Розділ 3. ЕЛЕКТРОННІ ЗАСОБИ ОБРОБКИ ЕКОНОМІЧНОЇ ІНФОРМАЦІЇ

Засвоївши матеріал, описаний у цьому розділі, студенти набудуть компетенцій щодо використання програмних комплексів Maxima та MatLab, а також отримають знання щодо мов як «C++», «Python» та «R».

Незважаючи на численні програмні комплекси, що дозволяють робити обробку економічних даних, у дослідників виникає проблема розуміння прийомів використання цих комплексів. Для більшості є достатнім хоча б поверхневий опис, щоб почати роботу з тим чи іншим комплексом. Ця мета реалізована в описі програмних комплексів Maxima та MatLab.

Але, якими б універсальними не були такі комплекси, дослідники вимагають все нових і нових можливостей, все нових і нових методів обробки, нових типів фільтрів даних, нових методів накопичення обробленої інформації. Тому без знання сучасних мов програмування, спеціально створених для цієї мети, неможлива якісна обробка економічної інформації.

Вашій увазі пропонується короткий опис таких мов як «C++», «Python» та «R», ознайомлення з якими дозволить, при бажанні самостійно вивчити їх глибше, узнати більше можливостей і краще працювати зі спеціальними даними, що їх накопичує і пропонує економіка.

3.1. Maxima

Сучасна економіка широко використовує математику, недарма більшість Нобелівських лауреатів з економіки отримали цю премію саме за використання досягнень обчислювальної математики в економіці. Але, за нашими

спостереженнями, сучасні економісти не часто застосовують вищу математику в чистому вигляді, так як не хочуть помилятися в складних математичних перетвореннях. Тепер всі ці обчислення автоматизовані.

Серед відкритих програм «комп'ютерної математики», які не потребують оплати ліцензії на використання, найширші можливості має Maxima, яка працює як під управлінням Windows, так і під управлінням Linux. Версія, яка існує на сьогодні - 5.15.0. Нові версії можна завантажувати на сайті http://maxima.sourceforge.net/ru/ або на моєму сайті http://www.polinskij.at.ua. Програма має декілька інтерфейсів. Консольна версія бідна візуальними можливостями: все математичні формули «відображаються» звичайними текстовими символами в кілька рядків дисплея, а графіки будуються у вигляді іксів. Але за рахунок цього різко знижуються вимоги до технічних параметрів комп'ютера - Maxima в консольному варіанті здатна працювати навіть на таких комп'ютерах, які на даний час і за комп'ютери ніхто не вважає. Крім консольної версії існують оболонка xMaxima, не набагато вимоглива до ресурсів, яка має систему меню і дозволяє будувати графіки будь-якими об'єктами, але математичні знаки імітуються, також як і в консольної версії, текстовими символами. Ще один інтерфейс, wxMaxima реалізований тільки для Windows. У цьому інтерфейсі реалізовані звичайні для сучасного користувача можливості вставляти в документ Maxima текст з буфера обміну і навпаки. Контекстне меню дозволяє редагувати набрані вираження. Система меню забезпечує обчислення всіх основних математичних дій в зручному віконному вигляді. Крім того, математичні формули відображаються звичайними графічними позначеннями.

3.1.1. Синтаксис Maxima

При старті (рис. 1.26) виводиться деяка інформація про систему і «мітка» (% i1). Ця мітка показує готовність програми до введення команд. Кожен введення і виведення позначаються програмою і потім можуть бути використані

знову. Символ% i (від input) використовується для позначення команд введення, а% o (від output) - при виведенні результатів обчислень.

Для ініціалізації процесу обчислень необхідно ввести команду, потім символ «;» (крапка з комою) і натиснути клавішу Enter. Якщо не потрібен виведення отриманої інформації на екран, то замість крапки з комою використовується символ $. Для арифметичних дій можна використовувати традиційні позначення: «+», «-», «*», «/» і «**» або «^» для введення в степінь. Наприклад, в осередок % i1 можна записати:

(% I1) (1/2 + 1/3 + 1/4) / (1/5 + 1/6 + 1/8);

(% O1) 130/59

Як випливає з останнього прикладу Maxima завжди показує результат в точної аналітичної формі. Якщо потрібно отримати результат обчислення у вигляді числа з плаваючою точкою, то після висловлення потрібно через кому задати дескриптор numer. Для цієї мети можна використовувати команду float, яка перетворює отриманий результат в потрібний вид:

(% I2) float (% o1);

(% O2) 2.203389830508475

На рис. 3.1. показано реалізацію описаних вище прикладів.

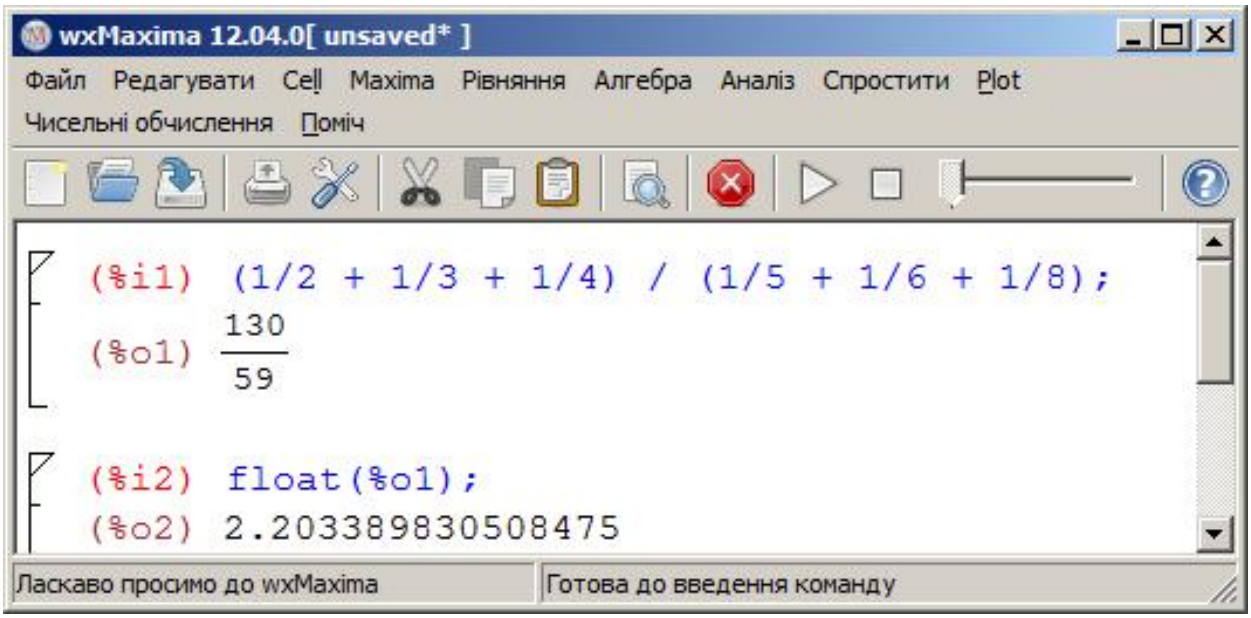


Рис. 3.1. Реалізація прикладів вирішення задач

Звернутися до результату останньої команди можна за допомогою символу%:

(% I3)% -12 / 59;

(% O3) 2.0

В цьому випадку % -12 / 59 - це те ж саме, що (i% o2) -12 / 59. Це позначення дозволяє звертатися до останнього результату, не відволікаючись на те, який у нього номер.

Для повтору останньої введеної команди, наприклад (% i3), в операційній системі Linux потрібно набрати комбінацію Alt + P, а в Windows досить натиснути стрілку вгору.

Maxima не звертає увагу на регістр введених символів в іменах вбудованих констант і функцій. Запис *sin (x)* еквівалентна запису *SIN (x)*. Регістр букв, важливий тільки при використанні змінних, наприклад, Maxima вважає *x* і *X* різними змінними.

Для стандартних математичних констант можуть використовуватися такі символи: % e – для підстави натуральних логарифмів,% i – для уявної одиниці (квадратний корінь з числа -1),% pi - для числа π. Позначення для деяких функцій: sqrt – корінь квадратний, log – натуральний логарифм.

Присвоєння значення для будь-якої змінної здійснюється за допомогою знака «:» (двокрапка).

(% I4) x: 2;

(% O4) 2

(% I5) y: 3;

(% O5) 3

(% I6) x + y;

(% O6) 5

При використанні елементарної математики будуть корисні прямі і зворотні тригонометричні функції: синус (sin x), косинус (cos x), тангенс (tan x), котангенс (cot x), арксинус (asin x), арккосинус (acos x), арктангенс (atan x), арккотангенс (acot x). Крім них є менш відомі функції - секанс (sec x = 1 / cos x)

і косеканс (csc x = 1 / sin x), а також гіперболічні функції - гіперболічний синус (sinh x =), гіперболічний косинус (cosh x =), гіперболічний тангенс (tanh x =), гіперболічний котангенс (coth x =). Розглянемо кілька прикладів обчислення тригонометричних функцій в Maxima:

(% I7) sin (% pi / 6) ^ 2 + cos (% pi / 6) ^ 2;

(% O7) 1

(% I8) sec (% pi / 3);

(% O8) 2

(% I9) asin (1/2);

(% O9)% pi / 6

(% I10) asin (sqrt (3) / 2);

(% O10)% pi / 3

Для спрощення тригонометричних виразів Maxima використовує функцію trigexpand. Функція trigreduce перетворює тригонометричний вираз в суму елементів, кожен з яких містить лише єдиний sin або cos. Приклади наведено на рис. 3.2.

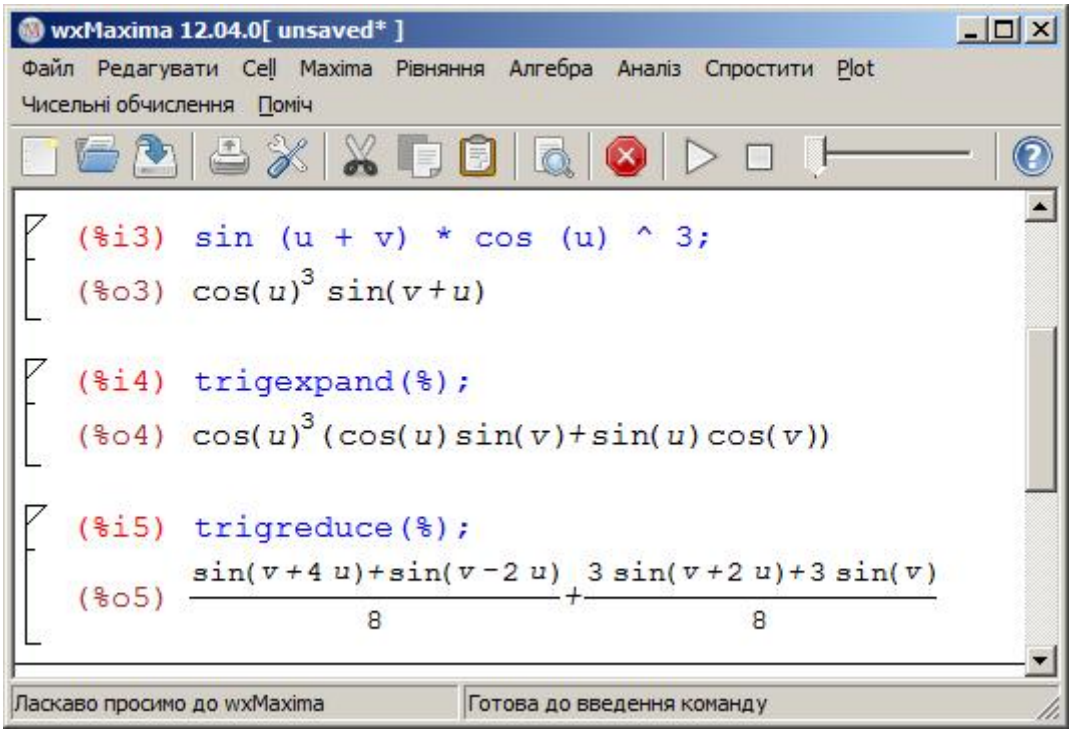


Рис. 3.2. Приклади розрахунку тригонометричних функцій

Функція trigsimp використовує тотожність sin2x + cos2x = 1, а також тотожність cosh2x - sinh2x = 1, щоб спростити вирази, які містять tan, sec і тому подібні в вирази, які містять sin, cos, sinh, cosh.

Будь-яке ім'я осередку % і або %o, або ім'я, яке присвоєно висловом з допомогою «:», можна очистити за допомогою функції kill. Можна також очистити всю пам'ять, набравши «kill (all)». Необхідно іноді це робити, так як імена змінних в сеансах роботи можуть повторюватися.

(% I14) kill (x);

(% O14) done

(% I15) x + y;

(% O15) x + 3

(% I16) kill (all);

(% O0) done

В останньому прикладі використана функція «kill (all)». Після цього нумерація осередків знову почнеться з одиниці

3.1.2. Функції Maxima

При обчисленні алгебраїчних рівнянь і нерівностей будуть корисні деякі функції Maxima (рис. 3.3):

– Функція expand, яка розкриває дужки і призводить вираз до канонічного вигляду. Наступний приклад демонструє розкриття дужок:

– Функція divide, яка знаходить приватне і залишок від ділення одного многочлена на інший:

– Функція gcd, яка визначає найбільший спільний дільник многочлена:

– Функція factor, яка здійснює розкладання многочлена на множники:

– Функція ratsimp, яка виносить за дужки найбільший спільний дільник:

Іноді для спрощення результату використовують функцію radcan.

Для обліку додаткових умов, які задаються нерівностями, використовують функцію assume (to assume – допускати):

(% I9) sqrt (x ^ 2);

(% O9) abs (x)

(% I10) assume (x <0);

(% O10) [x <0]

(% I11) sqrt (x ^ 2);

(% O11) -x

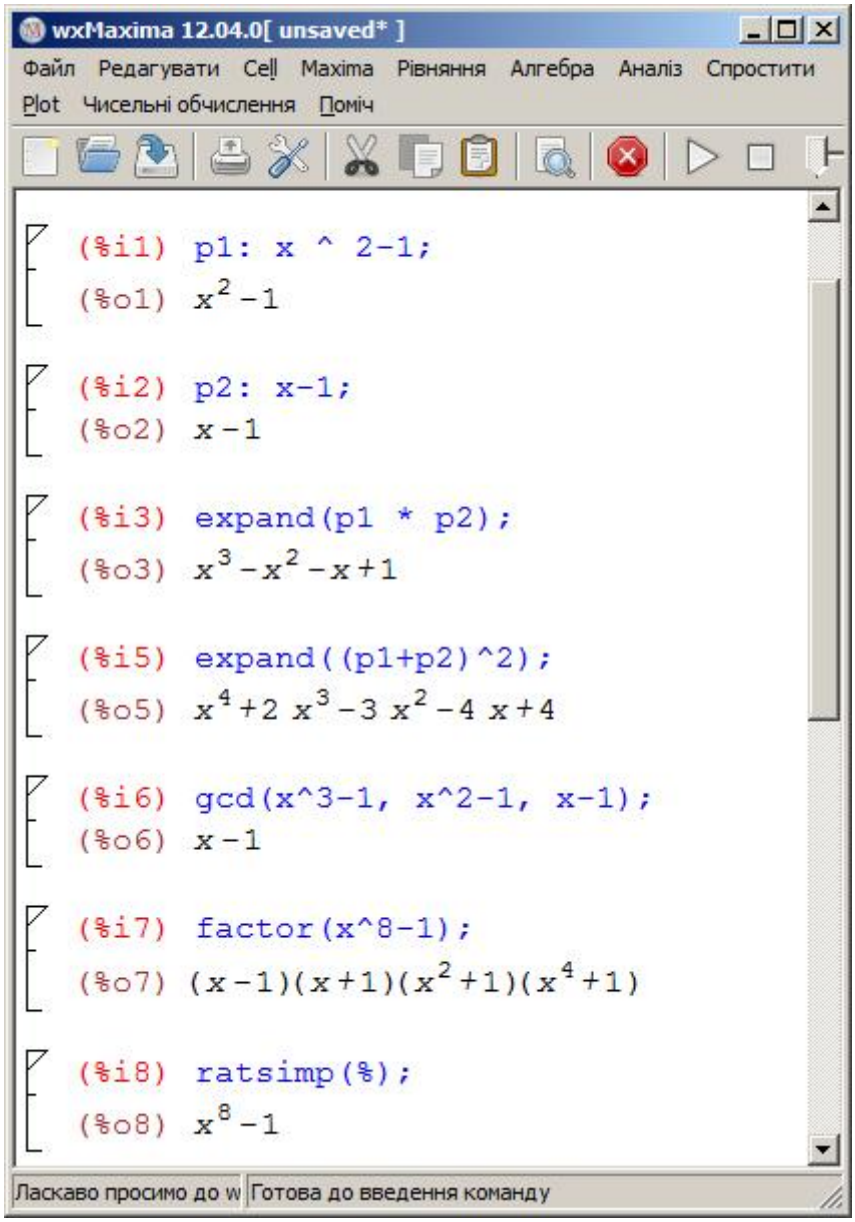


Рис. 3.3. Приклади роботи з функціями

Функція forget (to forget – забувати) знімає всі обмеження, накладені за допомогою assume:

```
(% I12) forget (x <0);
(% O12) [x <0]
(% I13) sqrt (x ^ 2);
(% O13) abs (x)
```

Для обчислення натурального логарифма в Maxima використовується функція log. Maxima не має вбудованої функції для десяткового логарифма або для інших підстав. Для переходу від однієї підстави логарифма до основи натурального логарифма використовуємо формулу. А для об'єднання логарифмів за формулами і використовується вбудована функція logcontract.

```
(% I14) log (% e ^ 2);
(% O14) 2
(% I15) 5 * log (a) + 6 * log (b);
(% O15) 6 * log (b) + 5 * log (a)
(% I16) logcontract (%);
(% O16) log (a ^ 5 * b ^ 6)
```

Символ «=» (так само) використовується при введенні підстановок або рівнянь. Наприклад, замінимо всі *x* на 5 / *z*:

```
(% I17) x ^ 4 + 3 * x ^ 3-2 * x, x = 5 / z;
(% O17) -10 / z + 375 / z ^ 3 + 625 / z ^ 4
```

Або вирішимо рівняння $y = x^2 - 58{,}5x + 607{,}5$:

```
(% I18) y: x ^ 2-58.5 * x + 607.5 $
(% I19) solve ([y], [x]);
'Rat' replaced 607.5 by 1215/2 = 607.5
'Rat' replaced -58.5 by -117/2 = -58.5
(% O19) [x = 45, x = 27/2]
```

В останньому прикладі для вирішення рівняння ми скористалися вбудованою функцією – solve. Цією ж функцією вирішують тригонометричні рівняння, наприклад,:

(% I20) solve ([tan (x) -sin (x) = 1-tan (x) * sin (x)], [x]);

'Solve' is using arc-trig functions to get a solution.

Some solutions will be lost.

В останньому виразі Maxima попереджає, що деякі рішення можуть бути загублені, оскільки не враховується періодичність функцій і навпаки, можуть бути рішення, які не входять до ОДЗ.

(% O20) [x =% pi / 4, x = -% pi / 2]

Якщо вираз має ім'я, можна де завгодно викликати його з цього імені:

(% I21) diff (y, x);

(% O21) 2 * x-58.5

(% I22) kill (all);

(% O0) done

Зрозуміло, що diff – це функція диференціювання. Крім обчислення похідних Maxima обчислює інтеграли, розкладає функції в ряди Тейлора, обчислює межі, диференціальні рівняння і дозволяє легко маніпулювати матрицями.

Однією такою особливістю є можливість програми Maxima не виконувати деяку функцію, а використовувати її в своєму математичному контексті. Наприклад, згадана функція диференціювання може бути використана для позначення похідної в диференціальному рівнянні; в цьому випадку обчислювати її не потрібно. Для таких потреб використовується знак апострофа:

(% I1) 'diff (y, x) = y;

(% O1) d / dx (y) = y

На противагу блокування обчислень, можна обчислити будь-який вираз примусово: таке примусова дія буде означати, що буде обчислено не тільки сам вираз, але і все входить у нього символи. Таке примусове обчислення робиться за допомогою функції ev:

(% I2) b: y ^ 2 $

(% I3) 'b / y;

(% O3) b / y

```
(% I4) y: 3 $ b; ev (b);
(% O5) y ^ 2
(% O6) 9
(% I7) u: x ^ 2-x = a;
(% O7) x ^ 2-x = a
(% I8) a: 0 $
(% I9) solve (ev (u), x);
(% O9) [x = 0, x = 1]
(% I10) solve (u, x);
(% O10) [[x = - (sqrt (4 * a + 1) -1) / 2, x = (sqrt (4 * a + 1) +1) / 2]
(% I11) ev (%);
(% O11) [x = 0, x = 1]
(% I12) kill (all);
(% O0) done
```

Кожне застосування ev обчислює вираз тільки на один рівень в глибину, але допускається застосовувати його до будь-якого виразу скільки завгодно разів:

```
(% I1) x: b $ b: a $ a: 3 $
(% I4) x;
(% O4) b
(% I5) ev (x);
(% O5) a
(% I6) ev (ev (x));
(% O6) 3
```

Для завершення роботи застосовується функція quit (); Довідка про будь-якої функції виводиться за допомогою команди describe (ім'я функції). Можна скористатися пунктом меню help. Процедура example (ім'я функції) демонструє приклади використання потрібної функції.

3.1.3. Матричні обчислення в Maxima

Maxima дозволяє легко маніпулювати матрицями. Для введення матриці необхідно після присвоювання імені через двокрапку написати функцію matrix, аргументами якої служать рядки матриці, розділені комами.

У наступному прикладі задаються дві матриці, які потім складаються (+) і перемножуються (*):

```
(% I1) A: matrix ([1,2],
                  [3,4]);
(% O1) matrix ([1,2],
               [3,4])
(% I2) B: matrix ([1,1],
                  [1,1]);
(% O2) matrix ([1,1],
               [1,1])
(% I3) A + B;
(% O3) matrix ([2,3], [4,5])
(% I4) A.B;
(% O4) matrix ([3,3],
               [7,7])
```

Функція determinant обчислює визначник матриці.

```
(% I5) determinant (A);
(% O5) -2
(% I6) determinant (matrix ([a, b],
                            [c, d]));
(% O6) a * d-b * c
```

Транспонування матриці здійснюється функцією transpose.

```
(% I7) transpose (A);
(% O7) matrix ([1,3],
```

[2,4])

Для отримання зворотного матриці використовується операція ^^ - 1 або функція invert.

(% I8) A ^^ - 1;
(% O8) matrix ([-2, 1],
[3/2, -1/2])
(% I9) invert (A);
(% O9) matrix ([- 2, 1],
[3/2, -1/2])

Як відомо, кожен елемент b_{ij} оберненої матриці $B = A^{-1}$ виходить розподілом алгебраїчного доповнення A_{ij} відповідного елемента вихідної матриці на її визначник | A |. Для того щоб винести 1 / |A| як співмножника застосовується функція detout.

(% I10) invert (% i1), detout;
(% O10) -matrix ([4, -2], [- 3,1]) / 2

Переконаємося в правильності отриманого результату, помноживши A на зворотну до неї матрицю:

(% I11) A.% o10;
(% O11) matrix ([1,0], [0,1])

Будьте уважні: в результаті виконання операції ^ -1 вийде матриця, кожен елемент якої обернений елементу вихідної, а не зворотна матриця.

(% I12) A ^ -1;
(% O12) matrix ([1, 1/2],
[1/3, 1/4])

Використання матриць дозволяє легко вирішувати системи лінійних рівнянь з декількома змінними. Нехай A - матриця коефіцієнтів системи, X - матриця невідомих, B - матриця вільних членів системи. Тоді матрицю X

знаходиться за формулою X = invert (A) .B, де операція «.» Означає матричне множення.

Для прикладу вирішимо наступну систему рівнянь матричним способом.

 x + 2y + z = 0,

 2x + y + z = 1,

 x + 3y + z = 0.

Спочатку очистимо пам'ять натиснувши kill (all). Далі заповнюємо відповідні матриці, а потім отримаємо матрицю результатів:

```
(% I13) kill (all);
(% O0) done
(% I1) A: matrix ([1,2,1],
                  [2,1,1],
                  [1,3,1]) $
(% I2) B: matrix ([0],
                  [1],
                  [0]) $
(% I3) invert (A) .B;
(% O3) matrix ([1],
               [0],
               [-1])
```

Для визначення рангу матриці використовується функція rank. Аргументом цієї функції є або матриця, або буква, якій присвоєно значення матриці. наприклад:

```
(% I4) rank (matrix ([2, 0, 2, 2],
                     [4, 4, 8,16],
                     [8, 4, 0, 2]));
(% O4) 3
```

Для вирішення систем рівнянь в програмі Maxima зручно використовувати функцію linsolve, де в якості першого аргументу через кому в квадратних дужках наводиться список рівнянь системи. Якщо вільні члени

рівнянь перенести в ліву частину, то відпадає необхідність введення знака рівності. Зрозуміло, що можна попередньо привласнити кожному рівняння будь-яку букву. Другим аргументом є список змінних. Ці змінні також наводяться в квадратних дужках через кому. Імена змінних повинні збігатися з тими, які введені в першому аргументі.

Розглянемо приклад застосування цієї функції при вирішенні такої системи рівнянь:

$$\begin{cases} 0{,}2x1 + 0{,}07x2 + 0{,}1x3 = 2480, \\ 0{,}1x1 + 0{,}01x2 + 0{,}4x3 = 3240, \\ 0{,}05x1 + 0{,}2x3 = 1600. \end{cases}$$

```
(% I5) linsolve ([0.2 * x1 + 0.07 * x2 + 0.1 * x3 = 2480,
                  0.1 * x1 + 0.01 * x2 + 0.4 * x3 = 3240,
                 0.05 * x1 + 0.00 * x2 + 0.2 * x3 = 1600.],
                   [X1, x2, x3]) $
`Rat 'replaced 0.2 by 1/5 = 0.2
`Rat 'replaced 0.07 by 7/100 = 0.07
`Rat 'replaced 0.1 by 1/10 = 0.1
`Rat 'replaced 0.1 by 1/10 = 0.1
`Rat 'replaced 0.01 by 1/100 = 0.01
`Rat 'replaced 0.4 by 2/5 = 0.4
`Rat 'replaced 0.05 by 1/20 = 0.05
`Rat 'replaced 0.2 by 1/5 = 0.2
(% O5) [x1 = 8000, x2 = 4000, x3 = 6000]
```

Функція linsolve вирішує такі системи лінійних рівнянь, де кількість рівнянь дорівнює кількості невідомих. Для вирішення перевизначених або недовизначених систем рівнянь, в яких кількість рівнянь відрізняється від кількості змінних, використовується функція algsys. Перший аргумент цієї функції – список рівнянь системи. Другий аргумент – список змінних.

Вирішимо лінійну систему рівнянь:

$$\begin{cases} x1 + x2 + x3 + 2x4 = 1, \\ x1 - 2x2 - 3x3 + x4 = -1, \\ 2x1 - x2 - 2x3 + 3x4 = 0. \end{cases}$$

```
(% I6) algsys ([x1 + x2 + x3 + 2 * x4 = 1,
                x1-2 * x2-3 * x3 + x4 = -1,
                2 * x1-x2-2 * x3 + 3 * x4 = 0],
                [x1, x2, x3, x4]);
(% O6) [[x1 =% r1, x2 =% r2, x3 = - (5 *% r2-% r1-3) / 7,
     x4 = - (% r2 + 4 *% r1-2) / 7]].
```

Як бачимо з прикладу Maxima використовує для визначення довільних постійних наступні символи: C1 =% r1, C2 =% r2.

3.1.4. Обчислення власних чисел і векторів матриці

Щоб знайти власні числа матриці використовують функцію eigenvalues. Ця функція виводить список, який складається з двох підсписків. Перший підсписок – це власні значення матриці A, а другий підсписок – це ступінь складності власних значень у відповідному порядку.

Функція eigenvalues викликає функцію solve, щоб знайти рішення характеристичного рівняння матриці. Іноді функція solve не здатна знайти рішення многочлена через те, що вони є комплексними числами.

Щоб знайти власні вектору матриці необхідно використовувати функцію eigenvectors. Ця функція створює список, в якому перші два підсписки, – це власні значення матриці і ступінь складності власних значень, а інші підсписки - це власні вектори матриці, які відповідають цим власним значенням.

```
(% I1) A: matrix ([1,1],
                  [8,3]) $
(% I2) eigenvalues (A);
```

(% O2) [[5, -1], [1,1]]

(% I3) eigenvectors (A);

(% O3) [[[5, -1], [1,1]], [1,4], [1, -2]]

3.1.5. Обчислення меж

Для автоматичного обчислення меж в пакеті Maxima використовується оператор limit, який має три основних параметри: функція, змінна, значення до якого наближається змінна. Зазвичай змінної є *x*, а в числової послідовності – *n*.

Наприклад, обчислимо межа числової послідовності

limit (((n + 3) ^ 3 + (n + 4) ^ 3) / ((n + 3) ^ 4- (n + 4) ^ 4) при n → ∞.

У рядку (% i1) вводимо оператор limit, а в дужках вводимо функцію, змінну, значення якого прагне змінна. Нагадаємо, що нескінченність позначається в Maxima як inf.

(% I1) limit (((n + 3) ^ 3 + (n + 4) ^ 3) / ((n + 3) ^ 4- (n + 4) ^ 4), n, inf);

(% O1) -1/2

3.1.6. Обчислення похідних

Для автоматичного обчислення похідної в пакеті Maxima використовується оператор diff, який має два основні параметри: функція і змінна, а також додатковий параметр, який визначає порядок похідної. Якщо потрібно спростити вираз, то використовується оператор ratsimp, а якщо вираз містить тригонометричні функції, то оператор trigsimp.

Наприклад, обчислимо другу похідну функції (рис. 3.4)

$$y = (3-x^2) * (\log (x))^2$$

Часткові похідні відрізняються від простих лише функцією по якій відбувається диференціювання.

Знайдемо приватні похідні першого порядку функції (рис. 3.4)

$$z = \mathrm{Log}(\mathrm{Exp}(x) + \mathrm{Exp}(y))$$

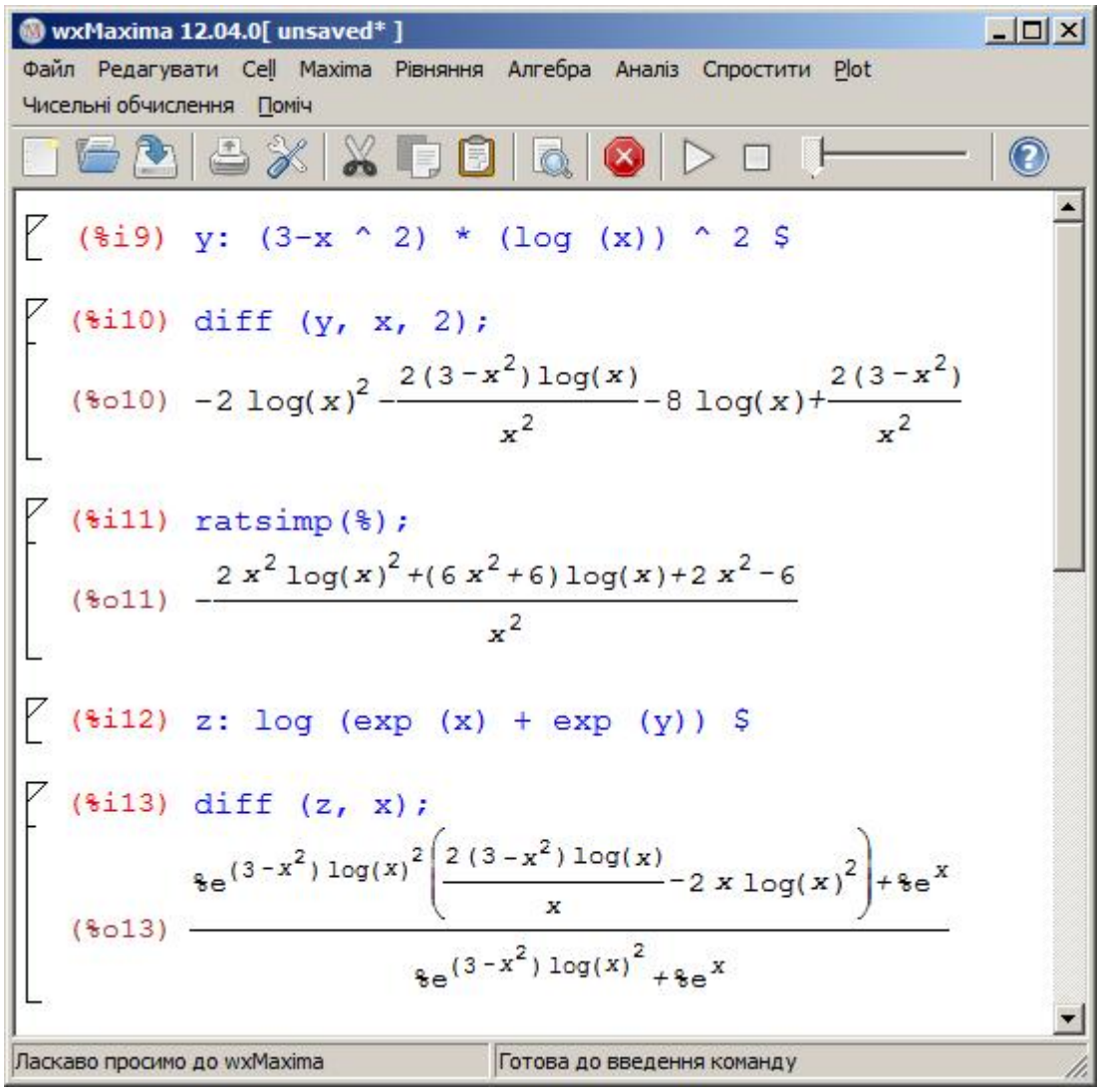


Рис. 3.4. Диференціювання функцій

Для розкладання функції за формулою Тейлора в пакеті Maxima використовується оператор taylor, який має чотири основних параметри: функція; змінна; точка для якої обчислюється ряд Тейлора; ступінь старшого члена в розкладанні.

Знайдемо, наприклад, розкладання функції ex за формулою Тейлора в околиці точки $x = 1$, обмежуючись $n = 4$.

(% I4) u: exp (x) $

(% I5) taylor (u, x, 1,4);

(% O5)% e +% e * (x-1) + (% e * (x-1) ^ 2) / 2 + (% e * (x-1) ^ 3)/
/ 6 + (% e *(X-1) ^ 4) / 24 + ...

3.1.7. Обчислення інтегралів

Для автоматичного обчислення інтегралів в пакеті Maxima використовується оператор integrate, який має два основні параметри: функція і змінна інтегрування.

(% I1) f: x ^ 2 / (4 * x ^ 6 + 1) $
(% I2) integrate (f, x);
(% O2) atan (2 * x ^ 3) / 6

Maxima в разі неоднозначного відповіді може задавати додаткові питання, як в наступному прикладі:

(% I3) integrate (x ^ n, x);
Is n + 1 zero or nonzero?

Пишемо відповідь на поставлене запитання і, як і при введенні будь-якої команди завершуємо набір крапкою з комою.

nonzero;
(% O3) x ^ (n + 1) / (n + 1)
(% I4) integrate (x ^ n, x);
Is n + 1 zero or nonzero?

Пишемо відповідь на поставлене запитання і, як і при введенні будь-якої команди завершуємо набір крапкою з комою.

zero;
(% O4) log (x)

Можна використовувати функцію assume для завдання додаткових умов (не забувайте потім видалити накладені обмеження):

(% I5) assume (notequal (n, -1)) $
(% I6) integrate (x ^ n, x);

(% O6) x ^ (n + 1) / (n + 1)

(% I7) forget (notequal (n, -1)) $

(% I8) integrate (x ^ n, x);

Is n + 1 zero or nonzero?

zero;

(% O8) log (x)

Обчислити невизначений інтеграл $\int \text{arctg}\sqrt{5x-1}dx$.

(% I9) u: atan (sqrt (5 * x-1)) $

(% I10) integrate (u, x);

(% O10) (2 * (((5 * x-1) * atan (sqrt (5 * x-1))) / 2 (sqrt (5 * x-1) -atan (sqrt (5 * x- 1))) / 2)) / 5

Щоб спростити останній вираз використовуємо оператор ratsimp.

(% I11) ratsimp (%);

(% O11) - (sqrt (5 * x-1) -5 * x * atan (sqrt (5 * x-1))) / 5

Для знаходження визначеного інтеграла слід вказати додаткові агрумент – межі інтегрування:

(% I12) integrate (x ^ 2, x, 0, 6);

(% O12) 72

(% I13) integrate (sin (x), x, 0,% pi);

(% O13) 2

Обчислення подвійного інтеграла:

(% I14) integrate (integrate (x * y, x, 1, 3), y, 0, 4);

(% O14) 32

Maxima допускає завдання і нескінченних меж інтегрування. Для позначення нескінченності використовується змінна inf:

(% I15) integrate (1 / x ^ 2, x, 1, inf);

(% O15) 1

(% I16) integrate (1 / (1 + x ^ 2), x, -inf, inf);

(% O16)% pi

(% I17) integrate (1 / x, x, 0, inf);

Integral is divergent

- an error. To debug this try debugmode (true);

В останньому прикладі система повідомила про неможливість обчислення інтеграла, тому що він розходиться (is divergent).

Якщо відповідь необхідно представити у вигляді числа з плаваючою точкою, то використовується оператор float (%), де% – номер рядка в якій потрібно перерахувати значення виразу.

(% I18) g: 1 / sqrt (2-x ^ 2) $

(% I19) integrate (g, x, 0,1);

(% O19)% pi / 4

(% I20) float (%);

(% O20) 0.78539816339745

Якщо відповідь необхідно представити у вигляді числа з плаваючою точкою, то використовується оператор float (%), де% - номер рядка в якій потрібно перерахувати значення виразу.

(% I18) g: 1 / sqrt (2-x ^ 2) $

(% I19) integrate (g, x, 0,1);

(% O19)% pi / 4

(% I20) float (%);

(% O20) 0.78539816339745

При обчисленні досить складних інтегралів відповідь не завжди буде представлена в найбільш простому вигляді. У наступному прикладі Maxima не може в символьному вигляді отримати відповідь, що дорівнює 5,295:

Обчислити інтеграл $2\pi\int_0^1 \sin\frac{\pi x}{2}\sqrt{1+\left(\frac{\pi}{2}\cos\frac{\pi x}{2}\right)^2}\,dx$.

(%i21) z: 2*%pi*sin(%pi*x/2)*sqrt(1+(%pi*cos(%pi*x/2)/2)^2)$

(%i22) integrate(z,x,0,1);

Is %pi^2 * sin(2*%pi*x) + 2 * %pi^2 * sin(%pi*x) + 16 *
sin(%pi*x) positive, negative, or zero?

```
positive;
(%o22)
2*%pi*integrate(sqrt((%pi^2*cos((%pi*x)/2)^2)/4+1)*sin((%pi*x)/2
),x,0,1)
```

Але спочатку необхідно зробити заміну змінної:

$$\left\langle \begin{matrix} \cos\frac{\pi x}{2} = t & -\frac{\pi}{2}\sin\frac{\pi x}{2}dx = dt \\ \text{при } x = 0 \quad t = 1 & \text{при } x = 1 \quad t = 0 \end{matrix} \right\rangle.$$

Тоді отримаємо вираз: $-4\int_1^0 \sqrt{1+\left(\frac{\pi}{2}t\right)^2}\,dt$.

```
(%i23) u:-4*sqrt(1+(%pi*t/2)^2)$
(%i24) integrate(u,t,1,0);
(%o24) (4*asinh(%pi/2)+%pi*sqrt(%pi^2+4))/%pi
(%i25) float(%);
(%o25) 5.294609402052086
```

3.1.8. Рішення диференціальних рівнянь

Для вирішення диференціальних рівнянь, лінійних відносно старшої похідної, в пакеті Maxima призначений оператор ode2.

Оператор ode2 вирішує звичайні диференціальні рівняння першого або другого порядку. Він використовує три аргументи: власне вираз для диференціального рівняння eqn, залежну змінну dvar, і незалежну змінну ivar. Все диференціальні рівняння містять похідні. Як відомо, в пакеті Maxima похідна вводиться за допомогою оператора diff. Оскільки при введенні виразу eqn не потрібно обчислювати похідну, то перед оператором diff використовується знак апострофа. Не потрібно також вказувати залежність змінної dvar від незалежної змінної ivar. Однак незалежна змінна повинна бути завжди вказана в якості

третьої аргументу. Для диференціальних рівнянь першого порядку константа інтегрування має вигляд% c, для диференціальних рівнянь другого порядку константами служать% k1 і% k2. У разі вдалого вирішення Maxima видає явне або неявне рішення для залежної змінної. Якщо з якоїсь причини ode2 не може отримати рішення, то з'являється напис false, тобто повідомлення про помилку.

Для диференціальних рівнянь, що мають тільки одне початкова умова (це диференціальні рівняння першого порядку) вирішити задачу Коші вдається за допомогою оператора ic1. Якщо є значення функції і її похідної в початковій точці (це диференціальні рівняння другого порядку) використовується оператор ic2. Якщо має місце гранична задача, тобто відомі координати двох крайніх точок диференціальних рівнянь, то використовується опція bc2.

Приклад 1: Вирішити задачу Коші для диференціального рівняння першого порядку $x^2 \cdot y' + 3 \cdot y \cdot x = \frac{\sin x}{x}$. Початкова умова: $y(\pi) = 0$.

```
(%i1) x^2*'diff(y,x) + 3*y*x = sin(x)/x$
(%i2) ode2(%,y,x);
(%o2) y=(%c-cos(x))/x^3
(%i3) ic1(%o2,x=%pi,y=0);
(%o3)  y=-(cos(x)+1)/x^3
```

Приклад 2: Вирішити задачу Коші для диференціального рівняння другого порядку $y'' + y \cdot (y')^3 = 0$. Початкові умови: а) $y(0) = 0; y'(0) = 2$; б) $y(0) = 1; y(1) = 3$.

```
(% I4) 'diff (y, x, 2) + y *' diff (y, x) ^ 3 = 0 $
(% I5) ode2 (%, y, x);
(% O5) (y ^ 3 + 6 *% k1 * y) / 6 = x +% k2
(% I6) ratsimp (ic2 (% o5, x = 0, y = 0, 'diff (y, x) = 2));
(% O6) - (2 * y ^ 3-3 * y) / 6 = x
(% I7) bc2 (% o5, x = 0, y = 1, x = 1, y = 3);
```

(% O7) (y ^ 3-10 * y) / 6 = x-3/2

Приклад 3: Вирішити задачу Коші для лінійного неоднорідного диференціального рівняння з постійними коефіцієнтами $x'' - 3x' - 4x = 4t$, де $x = x(t)$. Початкові умови: $x(0) = 3; x'(0) = 2$.

(% I8) 'diff (x, t, 2) -3 *' diff (x, t) -4 * x = 4 * t $

(% I9) ode2 (%, x, t);

(% O9) x =% k1 *% e ^ (4 * t) +% k2 *% e ^ (- t) - (4 * t-3) / 4

(% I10) ic2 (% o9, t = 0, x = 3, 'diff (x, t) = 2);

(% O10) x = (21 *% e ^ (4 * t)) / 20 + (6 *% e ^ (- t)) / 5 (4 * t-3) / 4

3.1.9. Побудова графіків

Функція plot2d показує графік одного або більше виразів, як функції однієї змінної або параметра.

Відображає один або кілька ділянок в двох вимірах. Коли вираження або ім'я функції використовуються для визначення ділянок, всі вони повинні залежати тільки від однієї змінної і використання x_range буде обов'язково, щоб вказати ім'я змінної і її максимальні і мінімальні значення.

Ділянка також може бути визначеною в дискретних або параметричних формах. Дискретна форма використовується для побудови набору точок з заданими координатами. Дискретна ділянку визначається список, починаючи з ключовим словом *discrete*, за яким слідує один або два списки значень. Якщо два списки дані, вони повинні мати однакову довжину; перший список буде інтерпретуватися як координати точок *x*, які будуть нанесені, і другий список, як координати *y*. Якщо тільки один список дається після дискретного ключового слова, кожен елемент в списку також повинен бути список з двома значеннями, які відповідають x і y координати точки.

Параметричну ділянку визначає список, починаючи з ключових словом *parametric*, за яким слід два вирази або імена функцій і діапазон для параметра. Діапазон для параметра повинен бути списком з ім'ям параметра з подальшим його мінімальним і максимальним значеннями. Графік покаже шлях, що проходить через крапку з координатами, наведеними двома виразами або функціями, як *Param* збільшується від *min* до *max*.

Діапазон для вертикальної осі є необов'язковим аргументом. Якщо цей параметр використовується, графік покаже точний вертикальний діапазон, незалежно від значення на горизонтальній осі. Якщо вертикальний діапазон не вказано, він буде створений відповідно до мінімальних і максимальних значеннями другої координати точок ділянки.

Всі інші варіанти повинні бути також списками, починаючи з ключовим словом, і потім одне або декількох значень.

Якщо є кілька ділянок, які будуть побудовані, легенда буде записана в ідентичність кожен з виразів. Етикетки, які повинні бути використані в цій легенді може бути дано з легендою варіант. Якщо цей параметр не використовується, Maxima буде створювати ярлики з виразів або імен функцій.

Аналогічно працює функція Plot3d, яка створює тривимірні графіки. Окремим типом функції, що будує тривимірні графіки є contour_plot. В ній об’ємна фігура представлена наборами замкнених ліній однакової висоти.

Наведемо приклади.

Побудова функції однієї змінної вимагає таку форму функції

```
(%i1) plot2d (sin(x), [x, -%pi, %pi])$
```

або таку

```
(%i1) plot2d (sec(x), [x, -2, 2], [y, -20, 20])$
```

На рис. 3.5 показано графік функції y = Sin(x).

Побудуємо контурний графік функції двох змінних виду $z = x^2 + y^2$, в діапазоні [-4; +4] для кожної змінної. Функція має вигляд

```
(%i1) contour_plot (x^2 + y^2, [x, -4, 4], [y, -4, 4])$
```

Графік функції наведено на рис. 3.6.

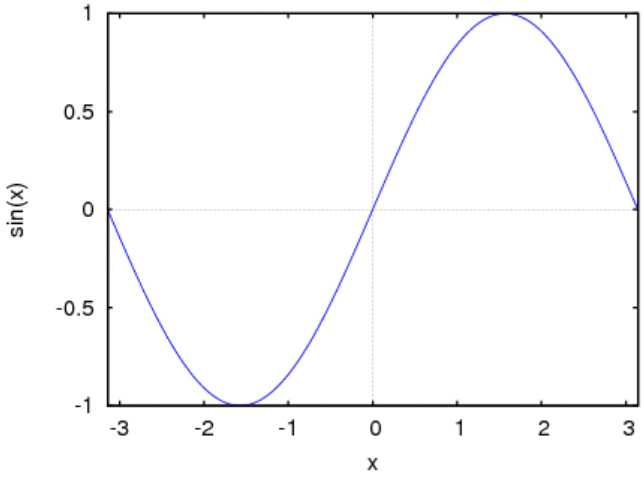


Рис. 3.5. Графік функції y = Sin(x)

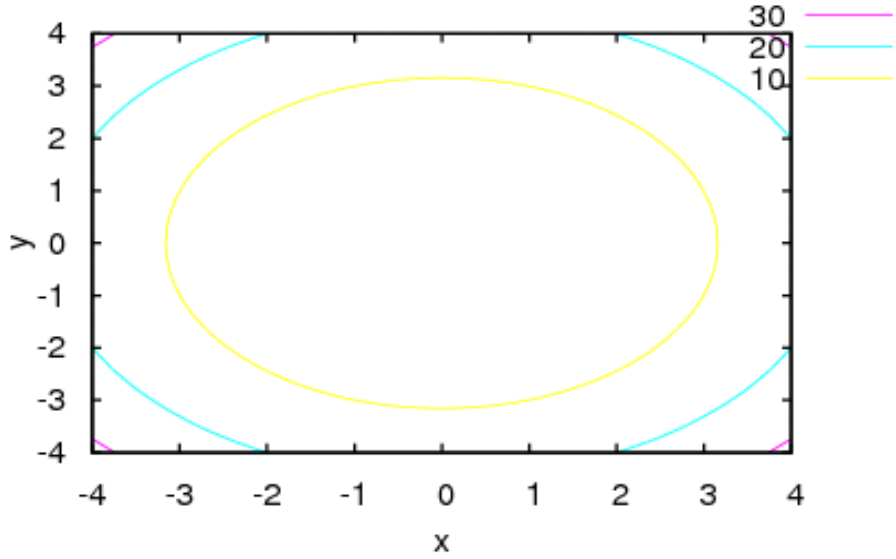


Рис. 3.6. Графік функції $z = x^2 + y^2$

Побудова тривимірного зображення функції $z = 2^{(-u^2 + v^2)}$

Функція має вигляд, а графік наведено на рис. 3.7.

```
(%i1) plot3d (2^(-u^2 + v^2), [u, -3, 3], [v, -2, 2])$
```

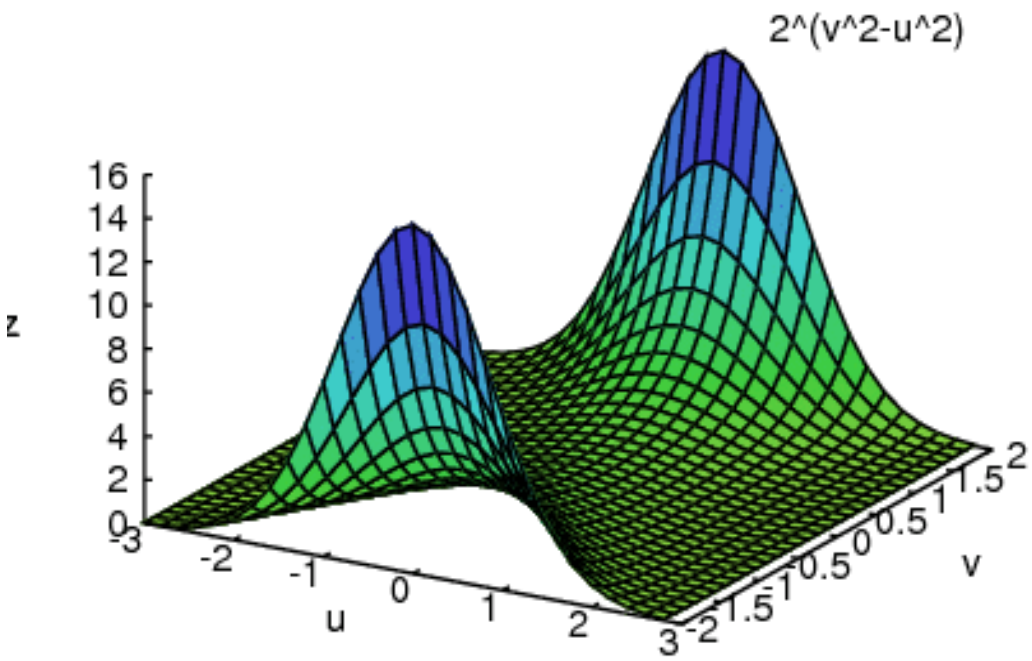


Рис. 3.7. Графік функції $z = 2^{(-u^2 + v^2)}$

3.2. MatLab

3.2.1. Робоче середовище MatLab

Щоб запустити програму двічі клацніть на іконку 

. Перед Вами відкриється робоче середовище, зображене на рис. 3.8.

Робоче середовище MatLab 6.x містить наступні елементи:

– панель інструментів з кнопками і списком;

– вікно з вкладками Launch Pad і Workspace, з якого можна отримати доступ до різних модулів ToolBox і до вмісту робочого середовища;

– вікно з вкладками Command History і Current Directory, призначене для перегляду і повторного виклику раніше введених команд, а також для установки поточного каталогу;

– командне вікно, в якому знаходиться запрошення до вводу і миготливий вертикальний курсор;

– рядок стану.

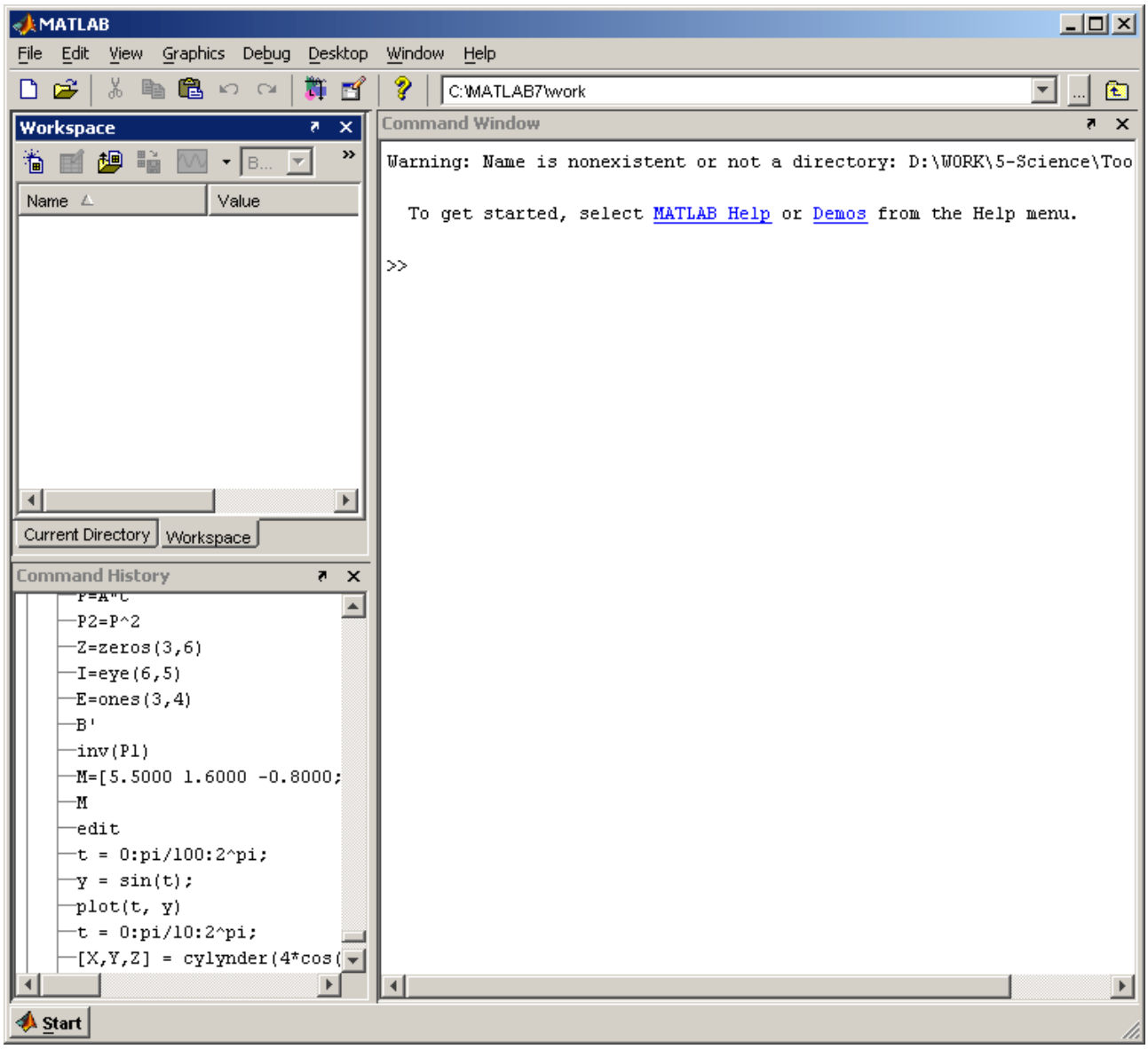


Рис. 3.8. Стартове вікно Matlab

Якщо в робочому середовищі MatLab 6.x відсутні деякі вікна, наведені на малюнку, то слід в меню View вибрати відповідні пункти: Command Window, Command History, Current Directory, Workspase, Launch Pad.

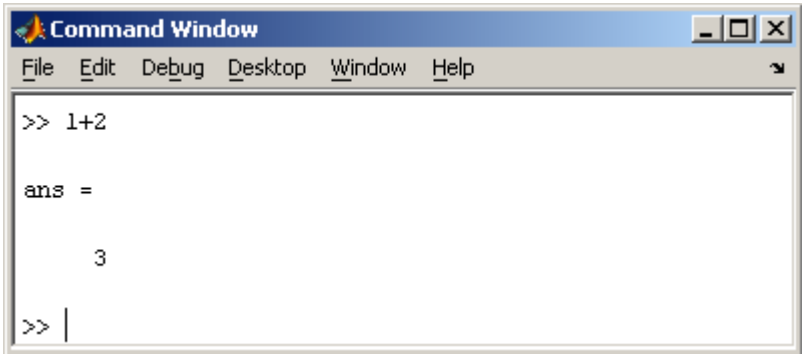
Команди слід набирати в командному вікні. Символ >>, що позначає запрошення до введення командного рядка, набирати не потрібно. Для перегляду робочої області зручно використовувати смуги скролінгу або клавіші Home, End,

для переміщення вліво або вправо, і PageUp, PageDown для переміщення вгору або вниз. Якщо раптом після переміщення по робочій області командного вікна пропала командний рядок з миготливим курсором, просто натисніть Enter.

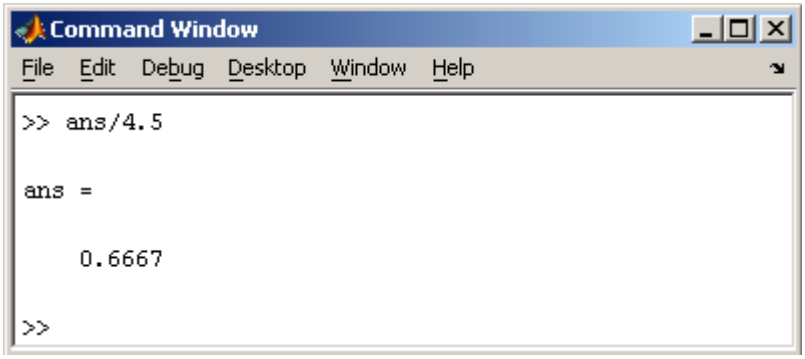
Важливо пам'ятати, що набір будь-якої команди або виразу повинен закінчуватися натисканням на Enter, для того, щоб програма MatLab виконала цю команду або вирахувала вираз.

3.2.2. Найпростіші обчислення

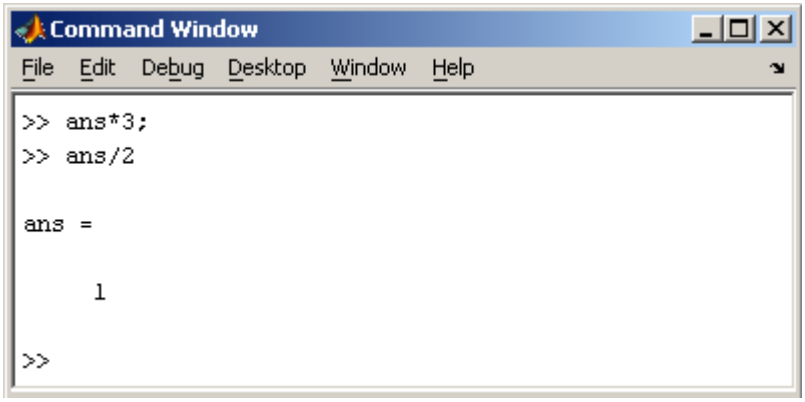
Наберіть в командному рядку 1 + 2 і натисніть Enter. В результаті в командному вікні MatLab відображається наступне:



Що зробила програма MatLab? Спочатку вона вирахувала суму 1 + 2, потім записала результат в спеціальну змінну ans і вивела її значення, що дорівнює 3, в командне вікно. Нижче відповіді розташована командний рядок з миготливим курсором, що позначає, що MatLab готовий до подальших обчислень. Можна набирати в командному рядку нові вирази і знаходити їх значення. Якщо потрібно продовжити роботу з попереднім виразом, наприклад, обчислити (1 + 2) /4.5, то найпростіше скористатися вже наявними результатом, який зберігається в змінної ans. Наберіть ans / 4.5 (при введенні десяткових дробів використовується крапка) і натисніть Enter, виходить:

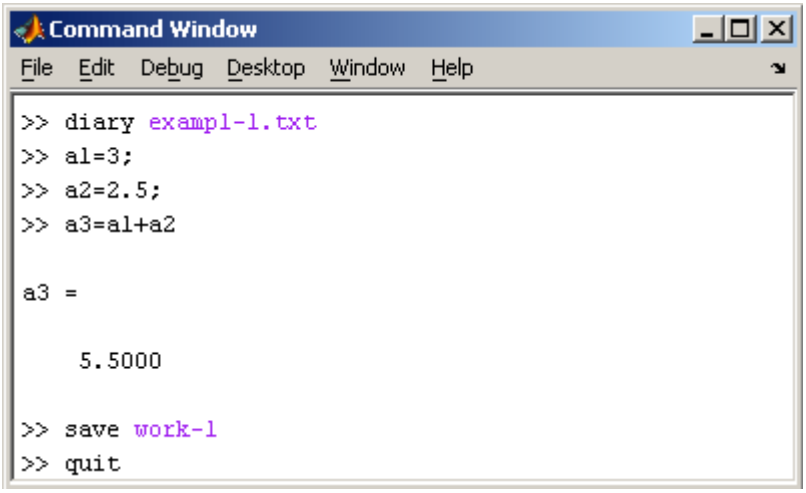


Виконання кожної команди в MatLab супроводжується відлунням. У наведеному вище прикладі– це відповідь ans = 0.6667. Часто відлуння ускладнює сприйняття роботи програми і тоді його можна відключити. Для цього команда повинна завершуватися символом крапка з комою.



Найпростіший спосіб зберегти всі значення змінних – використовувати в меню File пункт Save Workspase As. При цьому з'являється діалогове вікно Save Workspase Variables, в якому слід вказати каталог та ім'я файлу. За замовчуванням пропонується зберегти файл у підкаталозі work основного каталогу MatLab. Програма збереже результати роботи у файлі з розширенням mat. Тепер можна закрити MatLab. У наступному сеансі роботи для відновлення значень змінних слід відкрити цей збережений файл за допомогою підпункту Open меню File. Тепер всі змінні, визначені в минулому сеансі, знову стали доступними. Їх можна використовувати під нововведених командах.

У MatLab є можливість записувати виконувані команди і результати в текстовий файл (вести журнал роботи), який потім можна прочитати або роздрукувати з тек стового редактора. Для початку ведення журналу служить команда diary. Як аргумент команди diary слід задати ім'я файлу, в якому буде зберігатися журнал роботи. Набираються далі команди і результати їх виконання будуть записуватися я в цей файл, наприклад таку послідовність команд, що робить такі дії:



– відкриває журнал у файлі exampl-1.txt;

– обчислює;

– зберігає всі змінні в MAT файлі work-1.mat;

– зберігає журнал у файлі exampl-1.txt в підкаталозі work кореневого каталогу MatLab і закриває MatLab;

```
a1=3;
a2=2.5;
a3=a1+a2

a3 =

    5.5000

save work-1
quit
```

Якщо подивитися вміст файлу exampl-1.txt в якомусь текстовому редакторі, то у файлі виявиться наступний текст:

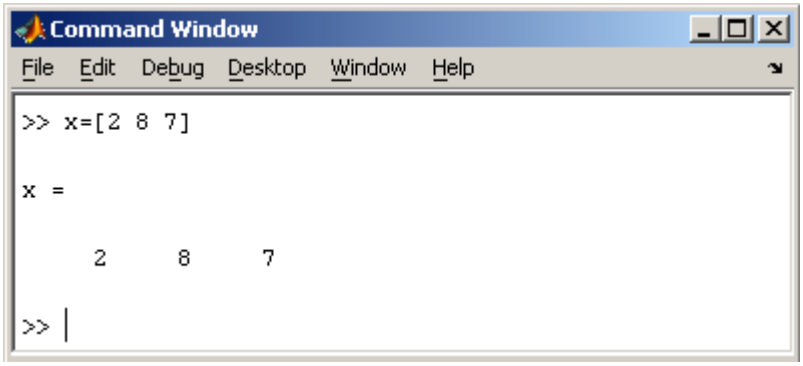
Вікно довідки MatLab з'являється після вибору опції Help Window в меню Help або натисканням кнопки питання на панелі інструментів. Ця ж операція може бути виконана при наборі команди helpwin. Для виведення вікна довідки з окремих розділів, наберіть helpwin topic. Вікно довідки надає Вам таку ж інформацію, як і команда help, але віконний інтерфейс забезпечує більш зручну зв'язок з іншими розділами довідки. Використовуючи адреса Web-сторінки фірми Math Works, ви можете вийти на сервер фірми і отримати саму останню інформацію по вас цікавлять. Ви можете ознайомитися з новими програмними продуктами або знайти відповідь на виниклі проблеми на сторінці технічної підтримки.

3.2.3. Матриці

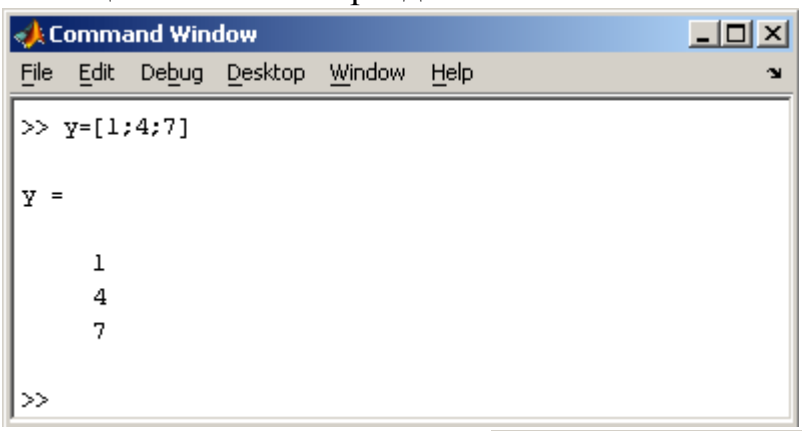
У MatLab можна використовувати скаляри, вектори і матриці. Для введення скаляра досить приписати його значення якоїсь змінної, наприклад p = 2.

Зауважимо, що MatLab розрізняє великі та малі літери, так що p і P – це різні змінні. Для введення масивів (векторів або матриць) їх елементи укладають у квадратні дужки. Так для введення вектору-рядка розміром 1 × 3,

використовується наступна команда, в якій елементи рядка відокремлюються пробілами або комами.

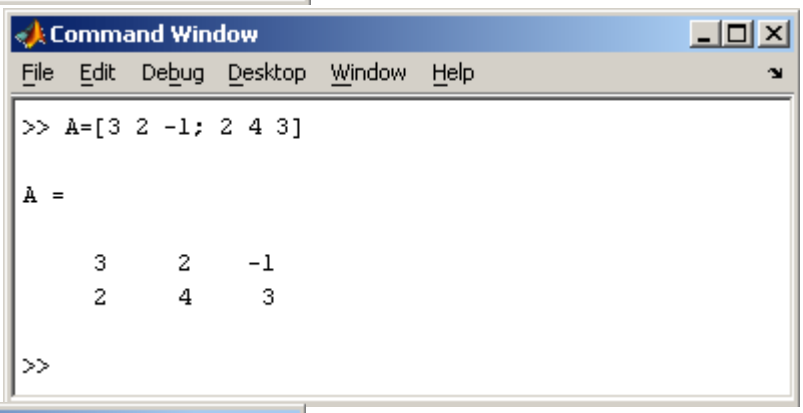


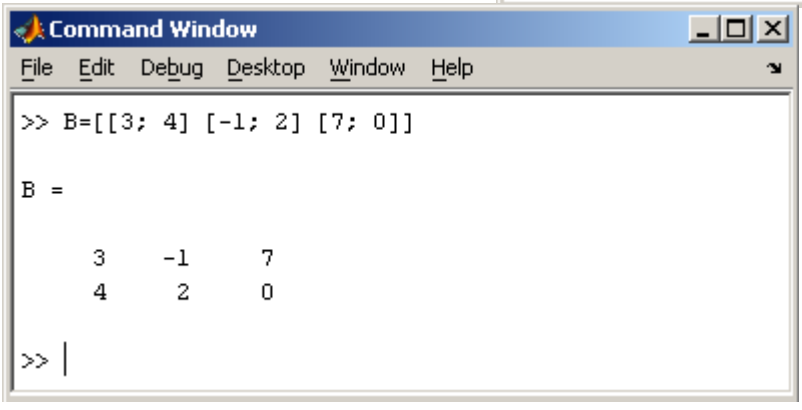
При введенні вектора-стовпця елементи розділяють крапкою з комою. Наприклад, так.



Або матрицю можна вводити по рядках. Розділяючи кожен рядок крапкою з комою.

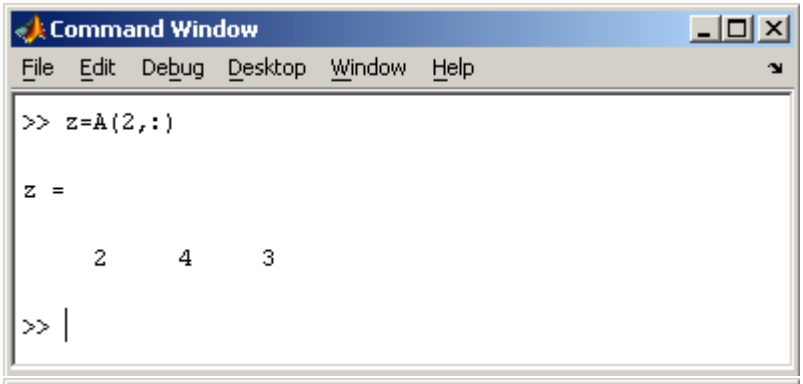
При введенні по стовпцях, нотація запису матриці міняється на наступну

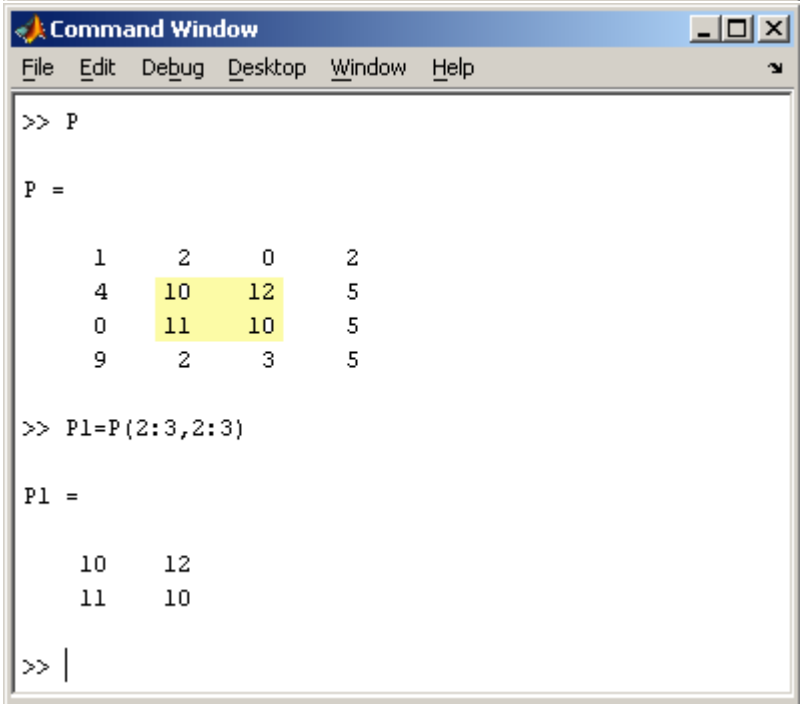




Доступ до елементів матриць здійснюється за допомогою двох індексів – номерів рядка і стовпця, укладених в круглі дужки, наприклад команда B (2,3) видасть елемент другого рядка і третього стовпця матриці B. Для виділення з матриці стовпця або рядка слід в якості одного з індексів використовувати номер стовпчика або рядка матриці, а інший індекс замінити двокрапкою. Наприклад, запишемо другий рядок матриці A в вектор z

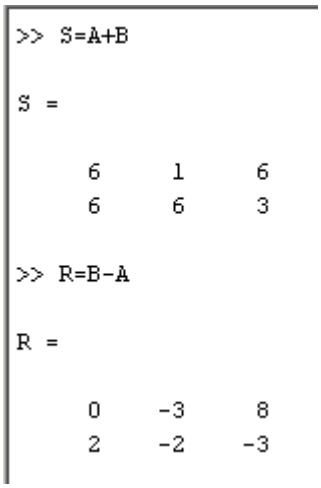
Також можна здійснювати виділення блоків матриць за допомогою двокрапки. Наприклад, виділимо з матриці P блок відзначений кольором.



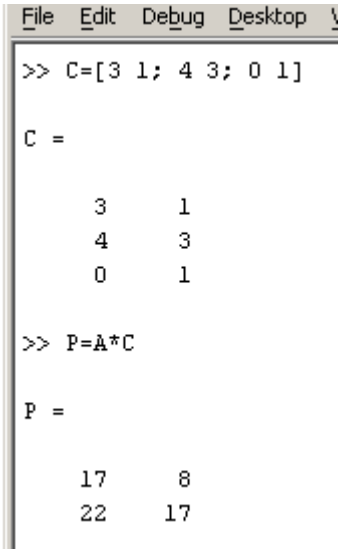


Якщо необхідно подивитися змінні робочого середовища, в командному рядку необхідно набрати команду whos.

При використанні матричних операцій слід пам'ятати, що для складання або віднімання матриці повинні бути одного розміру,



а при перемножуванні число стовпців першої матриці зобов'язана дорівнювати числу рядків другої матриці. Додавання і віднімання матриць, так само як чисел і векторів, здійснюється за допомогою знаків плюс і мінус, а множення - знаком зірочка *. Введемо матрицю розміром 3 × 2. Множення матриці на число теж здійснюється за допомогою зірочки, причому множити на число можна як справа, так і зліва. Зведення квадратної матриці на всю ступінь проводиться з використанням оператора ^.



Заповнення прямокутної матриці нулями проводиться вбудованою функцією zeros. Наприклад, команда >> Z= zeros (3,6) формує матрицю, що має три рядки та шість колонок, заповнених нулями.

Одинична матриця створюється за допомогою функції eye. Наприклад, команда >> I = eye(6,5), формує матрицю, у якої шість рядків та п’ять колонок, причому в головній діагоналі матриці стоять одиниці. Інші елементи матриці дорівнюю нулю.

Матриця, що складається з одиниць, утворюється в результаті виклику функції ones. Наприклад, >> E = ones(3,4).

транспонування матриці виробляється за допомогою апострофа '. Знаходження оберненої матриці проводиться за допомогою функції inv для квадратних матриць. Наприклад >> inv(P). Псевдообернена матрицю можна знайти за допомогою функції pinv. Наприклад >> pinv(P).

Більш докладно про обробку матричних даних можна дізнатися, якщо вивести список всіх вбудованих функцій обробки даних командою help datafun, а потім подивитися інформацію про потрібної функції, наприклад help max.

3.2.4. Об’єднання MatLab з Excel

Інтегрування MatLab і Excel дозволяє користувачеві Excel звертатися до численних функцій MatLab для обробки даних, різних обчислень і візуалізації результату. Надбудова excllink.xla реалізує дане розширення можливостей Excel. Для зв'язку MatLab і Excel визначені спеціальні функції.

Перед тим як налаштовувати Excel на спільну роботу з MatLab, слід переконатися, що Excel Link входить у встановлену на вашому комп’ютері версію MatLab. У підкаталозі exclink основного каталогу MatLab або підкаталогу toolbox повинен знаходитися файл з надбудовою excllink.xla. Запустіть Excel і в меню Tools виберіть пункт Add-ins. Відкриється діалогове вікно, що містить інформацію про доступні в даний момент надбудови. Використовуючи кнопку

Browse, вкажіть шлях до файлу excllink.xla. У списку надбудов діалогового вікна з’явиться рядок Excel Link 2.0 for use with MatLab зі встановленим прапорцем. Натисніть OK, необхідна надбудова додана в Excel.

Зверніть увагу, що в Excel тепер присутня панель інструментів Excel Link, яка містить три кнопки: putmatrix, getmatrix, evalstring. Ці кнопки реалізують основні дії, необхідні для здійснення взаємозв'язку між Excel і MatLab – обмін матричними даними, і виконання команд MatLab з середовища Excel. При повторних запусках Excel надбудова excllink.xla підключається автоматично.

Узгоджена робота Excel і MatLab вимагає ще кількох установок, які прийняті в Excel за замовчуванням (але можуть бути змінені). У меню Tools перейдіть до пункту Options, відкривається діалогове вікно Options. Виберіть вкладку General і переконайтеся, що прапорець R1C1 reference style вимкнений, тобто осередки нумеруються A1, A2 і т.д. На вкладці Edit повинен бути встановлений прапорець Move selection after Enter.

Запустіть Excel, перевірте, що виконані всі необхідні настройки так, як описано в попередньому розділі (MatLab повинен бути закритий). Введіть в осередку з A1 по C3 матрицю, для відділення десяткових знаків використовуйте точку відповідно до вимог Excel.

Виділіть на аркуші дані осередки і натисніть кнопку putmatrix, з'являється вікно Excel з попередженням про те, що MatLab не запущено. Натисніть OK, дочекайтеся відкриття MatLab.

З'являється діалогове вікно Excel з рядком введення, призначеним для визначення імені змінної робочого середовища MatLab, в яку слід експортувати дані з виділених осередків Excel. Введіть наприклад, M і закрийте вікно за допомогою кнопки OK. Перейдіть до командного вікна MatLab і переконайтеся, що в робочому середовищі створилась змінна M, яка містить масив, наприклад, три на три.

Виконайте деякі операції в MatLab з матрицею M, наприклад, оберніть її.

Виклик inv для обернення матриці, як і будь-якої іншої команди MatLab можна здійснити прямо з Excel. Натискання на кнопку evalstring, розташовану на

панелі Excel Link, призводить до появи діалогового вікна, в рядку введення якого слід набрати команду MatLab

IM = inv (M).

Результат аналогічний отриманому при виконанні команди в середовищі MatLab.

Поверніться в Excel, зробіть поточної осередок A5 і натисніть кнопку getmatrix. З'являється діалогове вікно з рядком введення, в якій потрібно ввести ім'я змінної, що імпортується в Excel. В даному випадку ім’я такої змінної є IM. Натисніть OK, в осередку з A5 по A7 введені елементи оберненої матриці.

Отже, для експорту матриці в MatLab слід виділити відповідні комірки аркуша Excel, а для імпорту досить вказати одну клітинку, яка буде верхнім лівим елементом імпортованого масиву. Інші елементи запишуться в осередку листа відповідно до розмірів масиву, переписуючи що містяться в них дані, тому слід дотримуватися обережності при імпорті масивів.

Вищеописаний підхід є найпростішим способом обміну інформацією між додатками - вихідні дані містяться в Excel, потім експортуються в MatLab, обробляються там певним чином і результат імпортується в Excel. Користувач переносить дані за допомогою кнопок панелі інструментів Excel Link. Інформація може бути представлена у вигляді матриці, тобто прямокутної області робочого листа. Комірки, розташовані в рядок або стовпець, експортуються, відповідно, в вектор-рядки і вектор-стовпці MatLab. Аналогічно відбувається і імпорт вектор-рядків і вектор-стовпців в Excel.

3.2.5. Програмування

Робота з командного рядка MatLab ускладнюється, якщо потрібно вводити багато команд і часто їх змінювати. Ведення щоденника за допомогою

команди diary і збереження робочого середовища незначно полегшують роботу. Найзручнішим способом виконання груп команд MatLab є використання M-файлів, в яких можна набирати команди, виконувати їх все відразу або частинами, зберігати в файлі і використовувати в подальшому. Для роботи з M-файлами призначений редактор M-файлів. З його допомогою можна створювати власні функції і викликати їх, в тому числі і з командного вікна.

Розкрийте меню File основного вікна MatLab і в пункті New виберіть підпункт M-file. Новий файл відкривається у вікні редактора M-файлів, яка зображена на рис. 3.9.

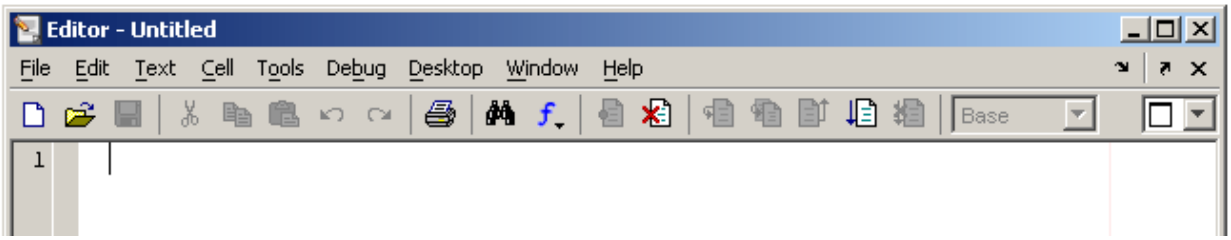
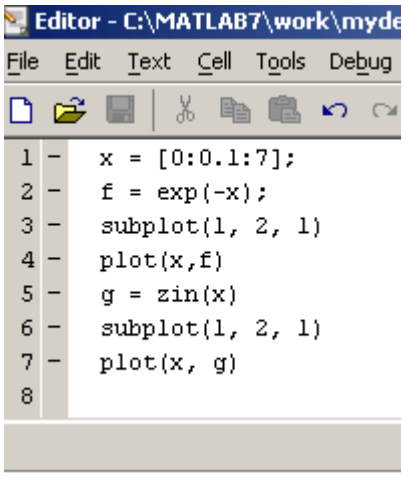


Рис. 3.9. Інтерфейс редактора M-файлів

M-файли в MatLab бувають двох типів: файл-програми (Script M-Files), що містять послідовність команд, і файл-функції, (Function M-Files), в яких описуються функції, визначені користувачем.

Наприклад, якщо набрати в редакторі команди, що призводять до побудови двох графіків на одному графічному вікні, як показано на прикладі ліворуч.



Якщо зберегти тепер файл з ім'ям mydemo.m в підкаталозі work основного каталогу MatLab, вибравши пункт Save as меню File редактора. Для запуску на виконання всіх команд, що містяться в файлі, слід вибрати пункт Run в меню Debug. На екрані з'явиться графічне вікно Figure 1, що містить графіки функцій.

Команди файл-програми здійснюють виведення в командне вікно. Для придушення виведення слід завершувати команди крапкою з комою. Якщо при наборі зроблена помилка і MatLab не може розпізнати команду, то відбувається виконання команд до неправильно введеної, після чого виводиться повідомлення про помилки в командне вікно.

Дуже зручною можливістю, що надається редактором М-файлів, є виконання частини команд. Закрийте графічне вікно Figure 1. Виділіть за допомогою миші, утримуючи ліву кнопку, або клавішами зі стрілками, утримуючи клавішу Shift, перші чотири команди і виконайте їх з пункту Text. Зверніть увагу, що в графічне вікно вивівся тільки один графік, відповідний виконаним: командам. Запам'ятайте, що для виконання частини команд їх слід виділити і натиснути клавішу F9.

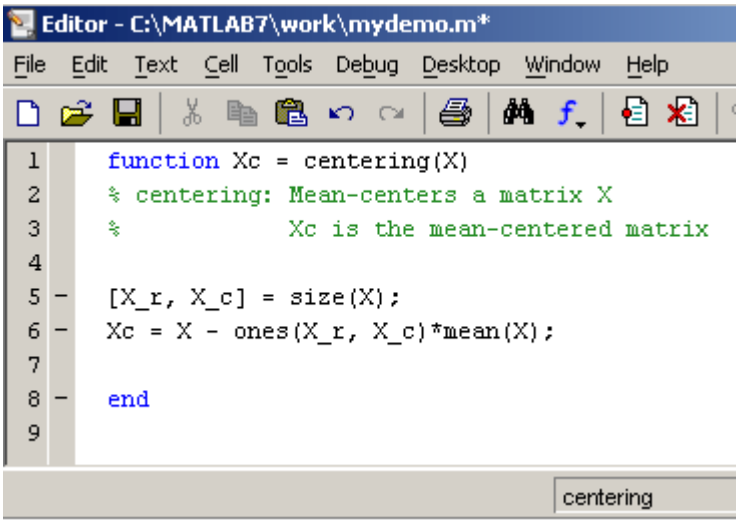
Окремі блоки М-файлу можна постачати коментарями, які пропускаються при виконанні, але зручні при роботі з М-файлом. Коментарі починаються зі знаку відсотка і автоматично виділяються зеленим кольором.

Відкриття існуючого М-файлу здійснюється за допомогою пункту Open меню File робочого середовища, або редактора М-файлів.

Розглянута вище файл-програма є лише послідовністю команд MatLab, вона не має вхідних і вихідних аргументів. Для використання чисельних методів і при програмуванні власних додатків в MatLab необхідно вміти створювати файл-функції, які виробляють необхідні дії з вхідними аргументами і повертають результат дії в вихідних аргументах. Розберемо декілька простих прикладів, що дозволяють зрозуміти роботу з файл-функціями.

Наприклад, проводячи попередню обробку даних багатовимірного аналізу часто застосовується центрування. Має сенс один раз написати файл-

функцію, а потім викликати його всюди, де необхідно проводити центрування. Відкрийте в редакторі М-файлів новий файл і наберіть наступний текст.

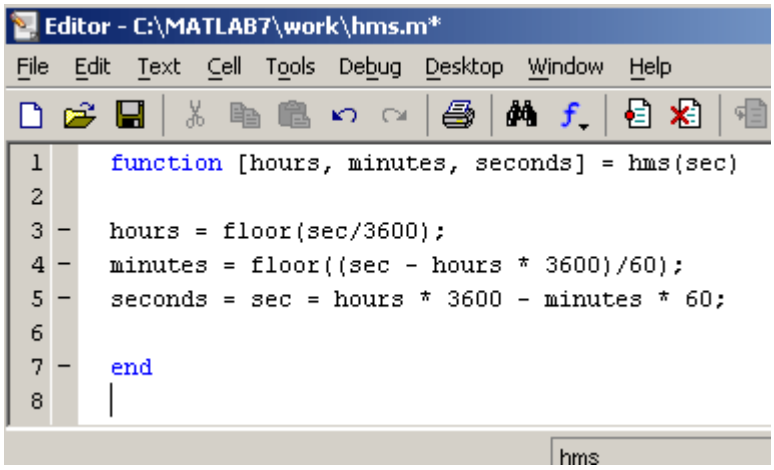


Слово function в першому рядку визначає, що даний файл містить файл-функцію. Перший рядок є заголовком функції, в якій розміщується ім'я функції і списку вхідних і вихідних аргументів. У прикладі ім'я функції centering, один вхідний аргумент X і один вихідний – X_c. Після заголовка слідують коментарі, а потім – тіло функції (воно в даному прикладі складається з двох рядків), де і обчислюється її значення. Важливо, що обчислене значення записується в X_c. Не забудьте поставити крапку з комою для запобігання виведенню зайвої інформації на екран. Тепер збережіть файл у робочому каталозі. Зверніть увагу, що вибір пункту Save або Save as меню File призводить до появи діалогового вікна збереження файлу, в полі File name якого вже міститься назва centering. Не змінюйте його, збережіть файл функцію в файлі з запропонованим ім'ям!

Тепер створену функцію можна використовувати так само, як і вбудовані sin, cos і інші. Виклик власних функцій може здійснюватися з файл-програми і з іншої файл-функції.

Можна написати файл-функції з кількома вхідними аргументами, які розміщуються в списку через кому. Можна також створювати і функції, які повертають кілька значень. Для цього вихідні аргументи додаються через кому в список вихідних аргументів, а сам список полягає в квадратні дужки. Хорошим

прикладом є функція, яка переводить час, заданий в секундах, в години, хвилини і секунди.

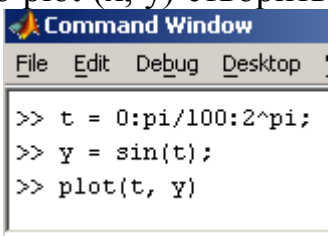


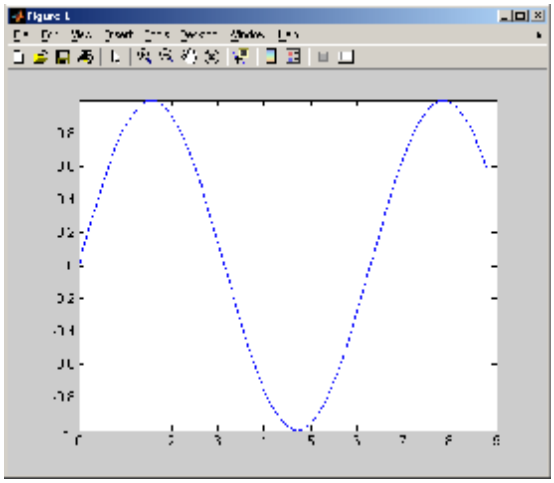
При виклику файл-функцій з кількома вихідними аргументами результат слід записувати в вектор відповідної довжини.

3.2.6. Створення графіка

MatLab має широкі можливості для графічного зображення векторів і матриць, а також для створення коментарів і друку графіків. Дамо опис кілька важливих графічних функцій.

Функція plot має різні форми, пов'язані з вхідними параметрами, наприклад plot (y) створює кусково-лінійний графік залежності елементів y від їх індексів. Якщо в якості аргументів задані два вектори, то plot (x, y) створить графік залежності y від x. Наприклад, для побудови графіка функції sin в інтервалі від 0 до 2p, зробимо наступне:

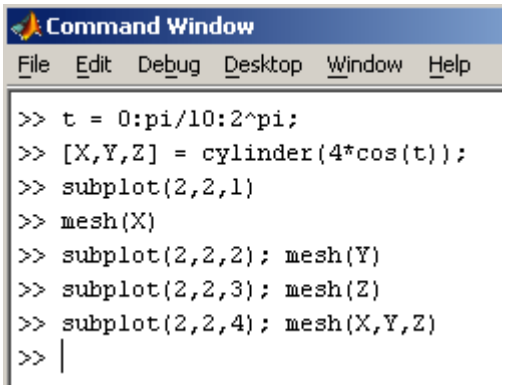




Програма побудувала графік залежності, який відображається у вікні Figure 1.

MatLab автоматично присвоює кожному графіку свій колір (крім випадків, коли це робить користувач), що дозволяє розрізняти набори даних.

Команда hold on дозволяє додавати криві на існуючий графік. Функція subplot дозволяє виводити безліч графіків в одному вікні.



На рис. 3.10. наведено приклад дії описаних команд.

Пункт Print в меню File і команда print друкують графіком MatLab. Меню Print викликає діалогове вікно, яке дозволяє вибирати загальні стандартні варіанти друку. Команда print забезпечує більшу гнучкість при виведенні вихідних даних і дозволяє контролювати друк з М-файлів. Результат може бути посланий прямо на принтер, обраний за замовчуванням, або збережений в заданому файлі.

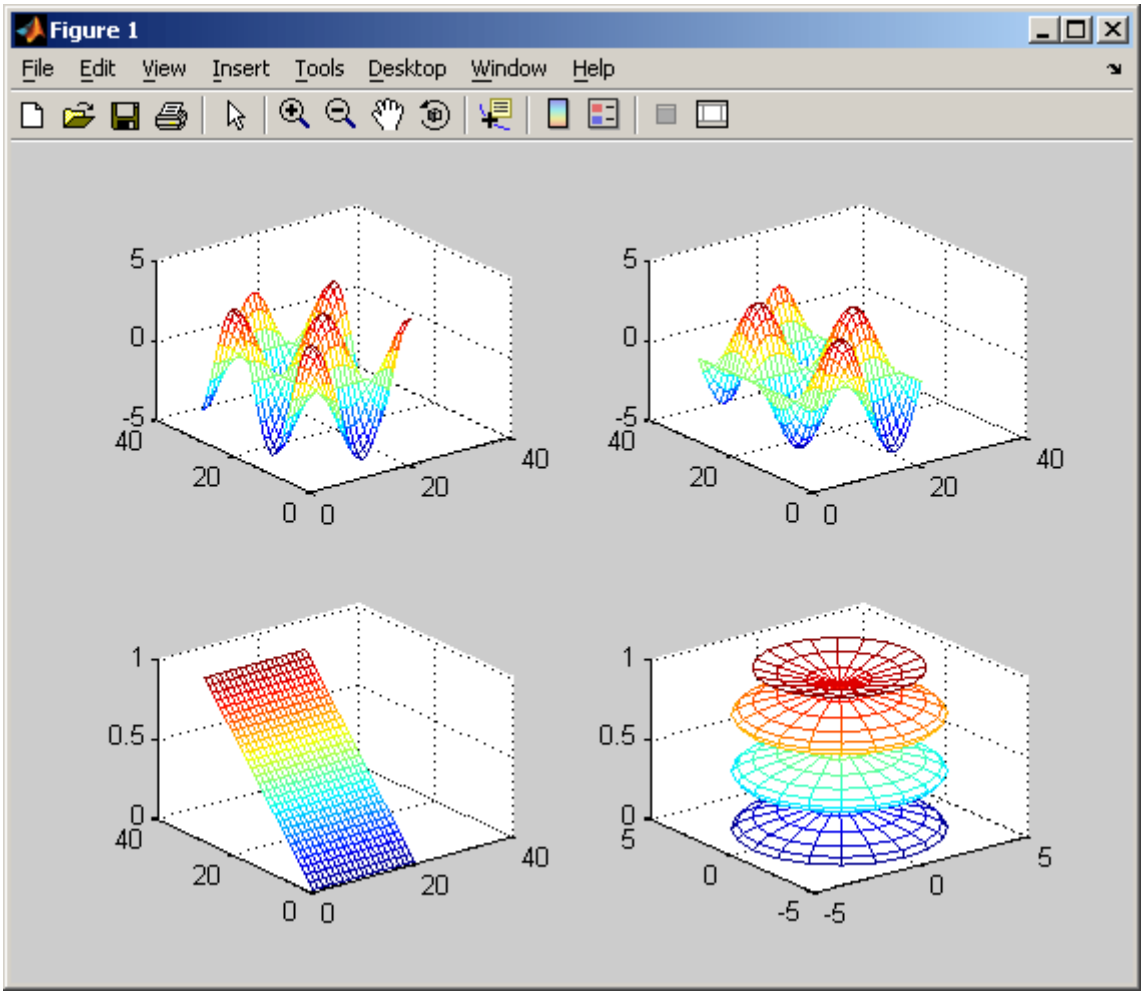


Рис. 3.10. Приклад дії команд побудови графіків

3.3. Visual Studio Express 2015 для Windows 10

За допомогою Visual Studio Express 2015 для Windows 10 можна створювати привабливі інноваційні універсальні додатки Windows. Цієї середи крім іншого входять: повнофункціональний редактор коду, потужний відладчик, спеціальний профілювальник і широкі можливості мовної підтримки для HTML5 / JavaScript, C ++, C # і Visual Basic, які дозволяють вам використовувати зручний для вас мову. Visual Studio Express 2015 для Windows також дозволяє завантажити емулятор пристроїв, який можна використовувати для тестування додатків на різних типах пристроїв без фізичного обладнання.

У цьому пункті ми розглянемо використання пакету для програмування на мові C#.

Популярне серед програмістів середовище програмування Visual C# 2008 Express можна встановити безкоштовно на комп'ютер з Інтернету на сайті:

https://www.visualstudio.com/ru/downloads/?rr=https%3A%2F%2Fwww.microsoft.com%2Fru-ru%2FSoftMicrosoft%2Fvs2015ExpressforW10.aspx

3.3.1. Установка пакета

При завантаженні Visual Studio 2017, ви отримаєте файл завантажувача, який в свою чергу встановлює інсталятор. Цей новий інсталятор включає в себе все, що потрібно, щоб налаштувати установку.

З папки Downloads, двічі клацніть файл завантажувач, який відповідає одній з таких ознак:

vs_enterprise.exe для Visual Studio Enterprise

vs_professional.exe для Visual Studio Professional

vs_community.exe для Visual Studio Community

Якщо ви отримали повідомлення управління обліковими записами користувачів про можливість запуску цього файлу, натисніть кнопку Так.

Далі потрібно підтвердити умови ліцензійної угоди Microsoft і Заяву про конфіденційність Microsoft, потім натиснути кнопку Продовжити.

З’являться кілька екранів, що показують хід установки. Після завершення установки, потрібно вибрати набори функцій або робочі навантаження, які потрібні. Це можуть бути одне або кілька робочих навантажень, які ви хочете; кожне навантаження містить функції, необхідні для мови програмування або платформи, якій ви віддаєте перевагу. На рис. 3.11 наведено зразок такого вікна. Для встановлення потрібних опцій, клацніть в лівому верхньому кутку прямокутника, що містить опцію.

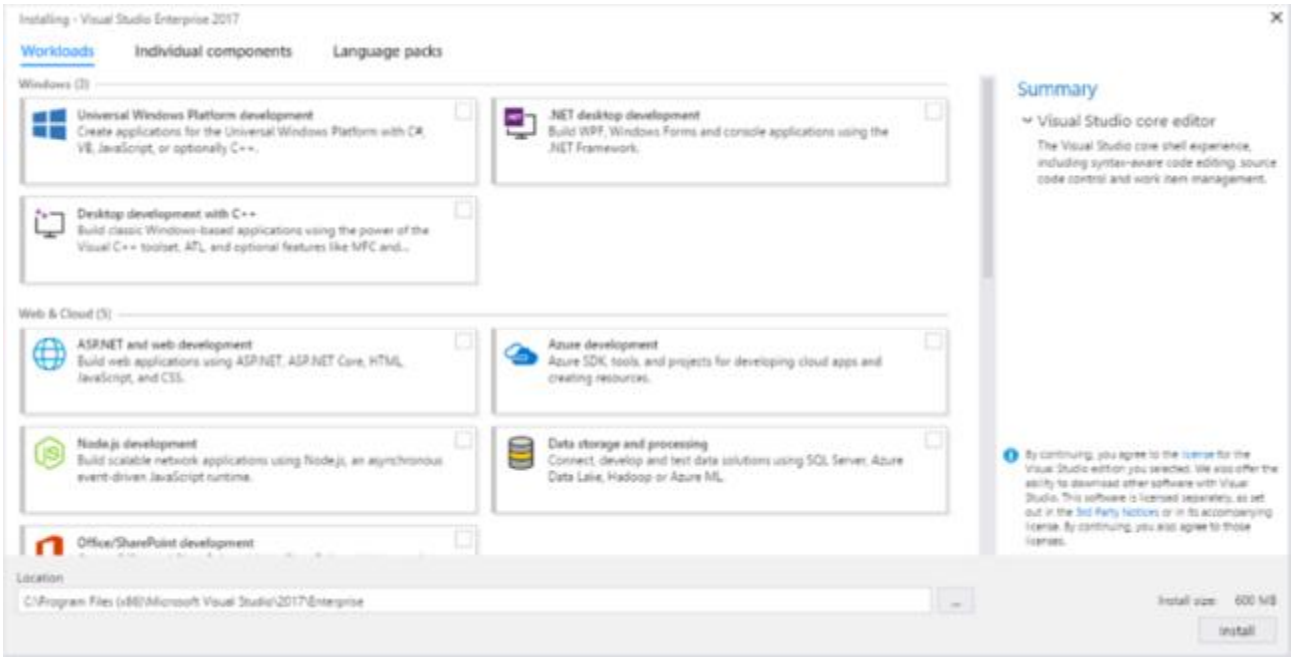


Рис. 3.11. Вибір опцій пакета

Наприклад, можна вибрати робоче навантаження розробки .NET робочого столу. Він поставляється за замовчуванням. Це редактор, який включає в себе базову підтримку редагування коду для більш ніж 20 мов, можливість відкривати і редагувати код з будь-якої папки без необхідності проекту, а також інтегрований контроль вихідного коду.

Після вибору необхідного робочого навантаження, натисніть кнопку Встановити. З'являться робочі вікна, що показують хід вашої візуальної установки Studio.

Після того, як нові робочі навантаження і компоненти встановлені, натисніть Запуск.

Якщо натиснути Індивідуальний варіант компонентів з Visual Studio Installer, з’являється можливість вибрати те, що потрібно, дотримуючись вказівок, що з’являться на екрані (рис. 3.12).

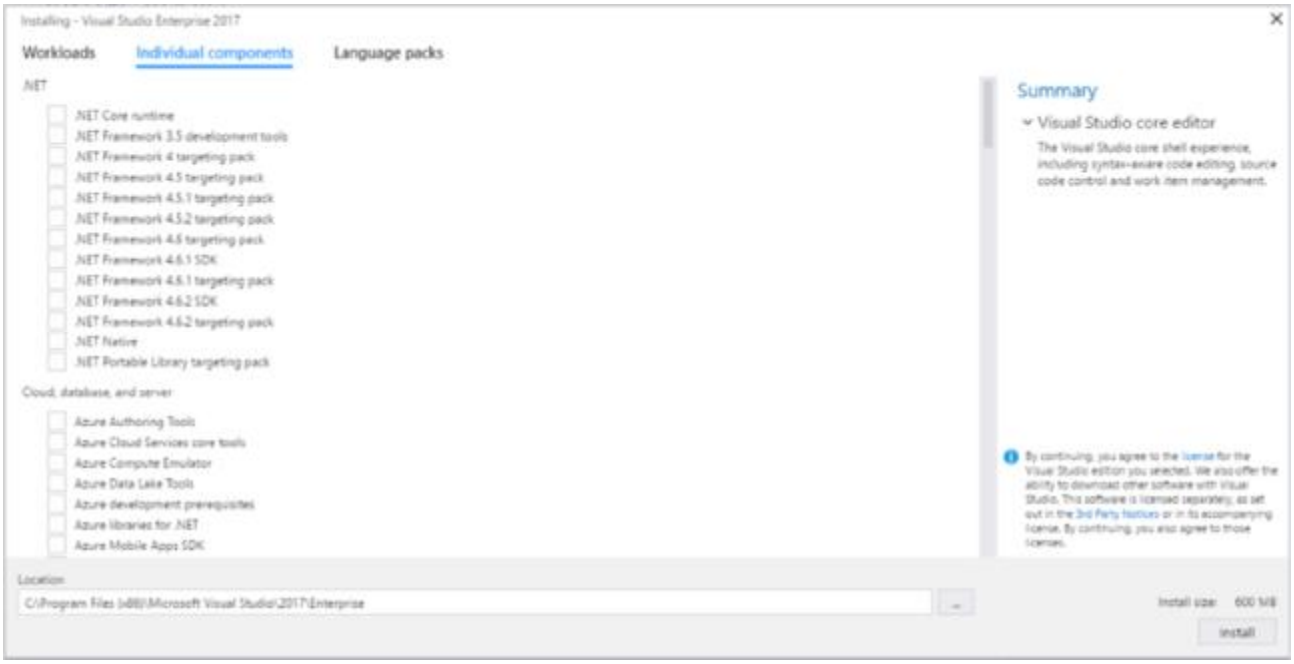


Рис. 3.12. Варіанти вибору опцій вручну

Для установки Visual Studio 2017 на мові за вашим вибором, виберіть опцію Мовні пакети з Visual Studio Installer, і дотримуйтесь інструкцій на екрані.

За замовчуванням програма установки намагається відповідати мові операційної системи, коли вона запускається в перший раз. Програма установки запам'ятовує цю установку. Ви можете змінити цю установку, запустивши програму установки з командного рядка. Наприклад, ви можете змусити програму установки працювати на англійській мові, використовуючи наступну команду: vs_installer.exe --locale EN-US. Установник запам'ятає цю настройку

при запуску в наступний раз. Програма установки підтримує такі мовні маркери: ZH-CN, ZH-TW, CS-CZ, EN-US, FR-FR, де-DE, он-IT, JA-JP, ко-KR, PL-PL, PT-BR, RU-RU, ES-ES, і TR-TR.

Варто пам’ятати, що Visual Studio встановлюється тільки на логічний диск C:\, тому перед установкою варто перевірити, чи достатньо пам’яті на жорсткому диску.

3.3.2. Інтерфейс програми Microsoft Visual C#

Вікно середовища програмування Microsoft Visual C# (рис.3.13) складається з меню (1), стандартної панелі інструментів (2), списку раніш виконуваних проектів (3), кнопок для відкриття та створення проектів (4), панелі для виводу повідомлень (5).

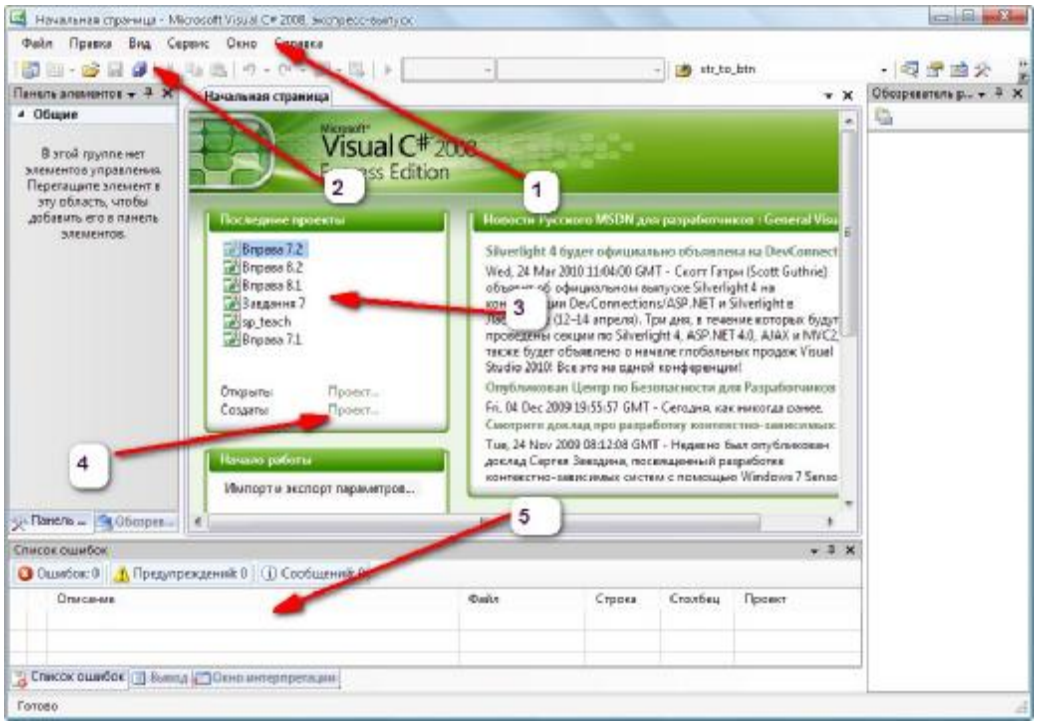


Рис. 3.13. Вікно середовища програмування Microsoft Visual C#

Створимо новий проект. Для цього клацнемо слово Проект.

У вікні (рис. 3.14) для створення програми з віконним графічним інтерфейсом виберемо **Приложение Windows Forms** (1), вкажемо назву проекту (2) та підтвердимо вибір клацнувши Ok (3).

Після створення нового проекту зовнішній вигляд програми зміниться (рис. 3.15) У вікні активізуються панелі:

Макет (3), форма проекту (4), панель елементів (5), закладки робочих полів (6), оглядач рішень (файлів проекту) (7), панель попереджень, помилок та повідомлень (8), панель властивостей візуальних елементів (9).

Макет (3). Панель призначено для вирівнювання візуальних елементів на формі проекту.

Форма проекту(4). Представляє вікно програми, на яке з панелі інструментів (5) переміщаються візуальні компоненти (кнопки, списки, текстові поля тощо). Всі візуальні компоненти встановлені на форму проекту мають маркери зміни розмірів, за допомогою яких можлива зміна їх розмірів та положення.

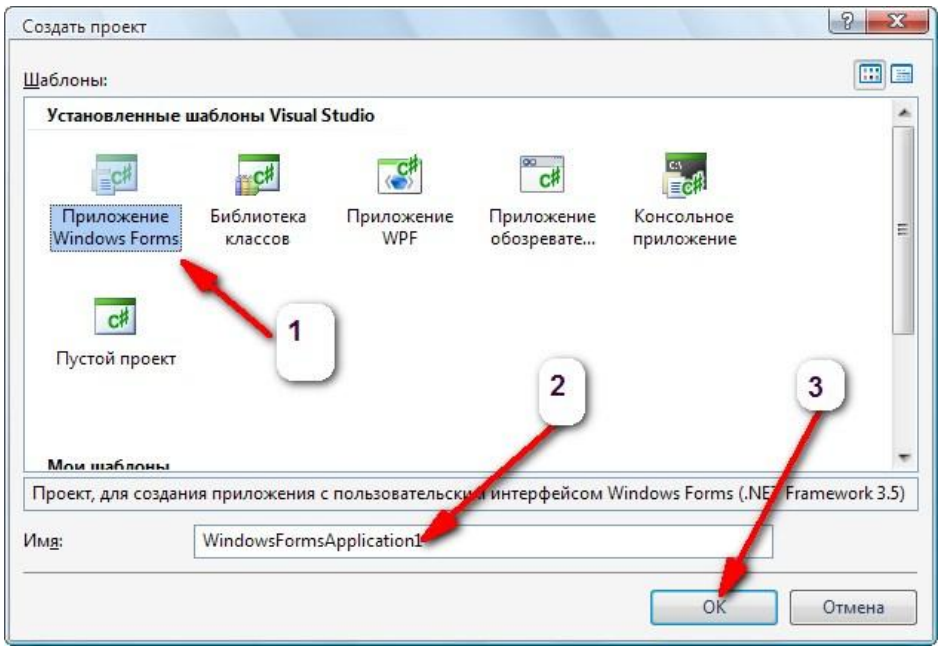


Рис. 3.14. Створення проекту

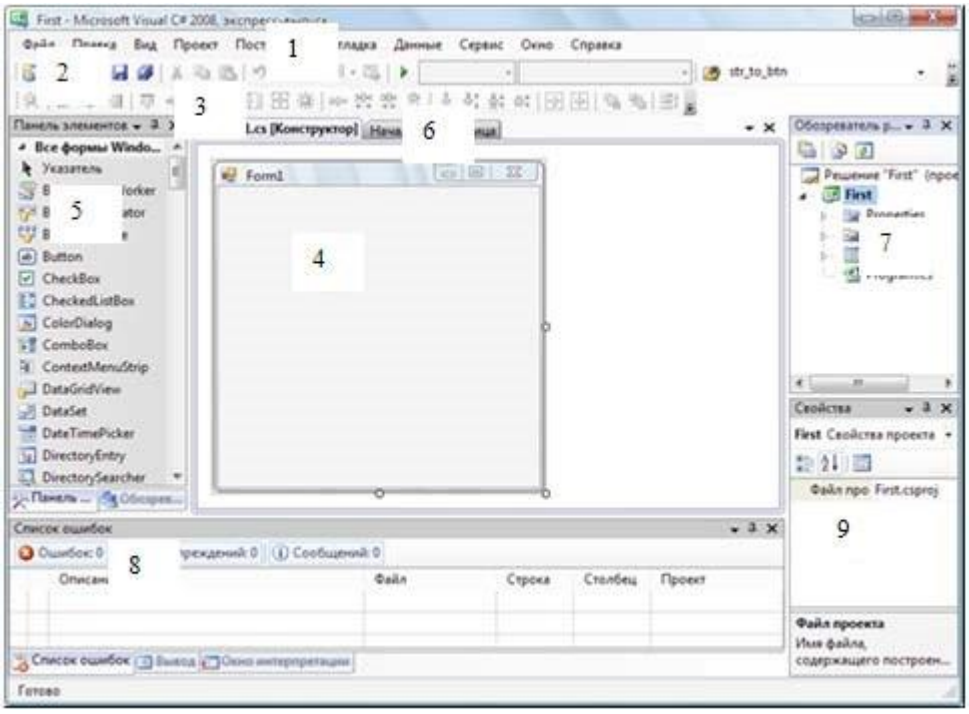


Рис. 3.15.

Панель елементів (5). На цій панелі зосереджено візуальні та невізуальні компоненти, з яких будується проект. Для вибору компоненту перетягніть його мишкою на вікно форми.

Закладки робочих полів (6) дозволяють перемикатися між відкритими файлами проекту.

Оглядач рішень (7). У даній панелі виводиться список файлів проекту. При подвійному клацанні по відповідному файлу, він відкривається на робочому полі. Інформація в цій панелі має ієрархічну структуру, яка вказує на її структуру та підпорядкованість.

У панель попереджень, помилок та повідомлень (8) виводиться інформація під час відлагоджування та компіляції проекту.

У панелі **Властивості (9)** виводиться список доступних властивостей для вибраного компоненту на вікні форми.

3.3.3. Створення консольного проекту в середовищі Microsoft Visual C#

Вивчення основ мови C# розпочнемо з консольного проекту. Що таке консольний проект? Більшість сучасних комп'ютерних програм створюється для виконання їх користувачем. Вікно програми має елементи керування (кнопки, текстові поля, списки, таблиці, перемикачі, прапорці). Такі програми назвемо Windows Forms. Створення проекту Windows Forms вимагає використання стандартних елементів операційної системи. На відміну від програм Windows Forms, консольний проект не використовує жодних елементів керування, а взаємодія з користувачем здійснюється безпосередньо через вивід інформації у вікно консолі та ввід з клавіатури. Далі називатимето вивід на екран – вивід в консоль, а ввід з клавіатури – ввід з консолі.

Для створення консольного проекту виберемо у ході створення нового проекту (Рис. 3.14) **Консольное приложение**.

Після створення даного проекту відкриється файл Program.cs (рис. 3.16). Панель елементів та властивостей залишиться неактивною, так як консольні проекти не використовують візуальних компонентів.

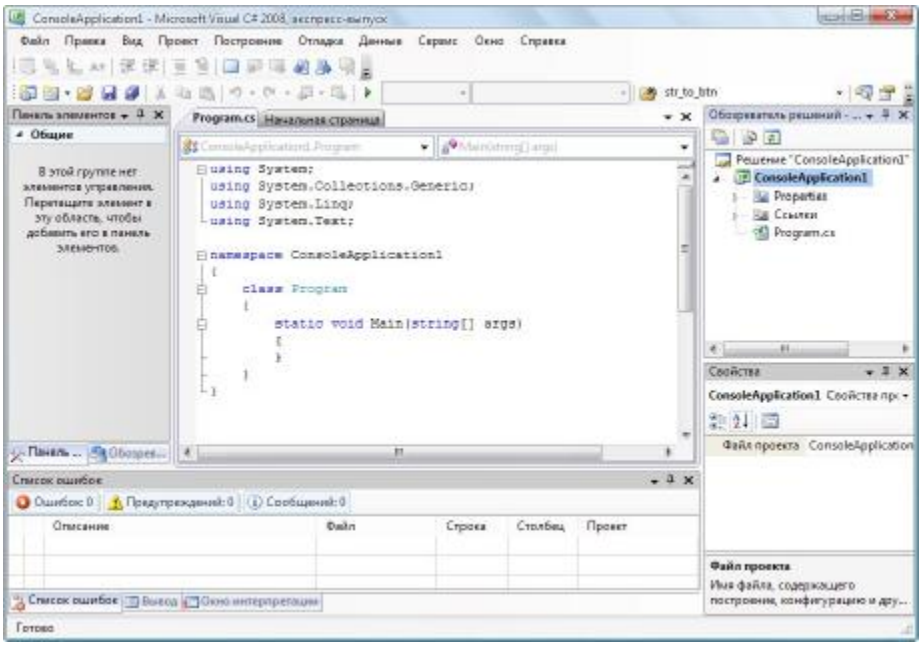


Рис. 3.16. Вікно консольного додатку

Автоматично буде написаний початковий код програми, який може бути дописаний користувачем.

Для збереження проекту виберемо Файл, Сохранить все… Для проекту буде створено окрему папку, в яку запишуться всі файли проекту рис. 3.17).

Для створення нового проекту слід вибрати з меню **File-> New-> Solution**.

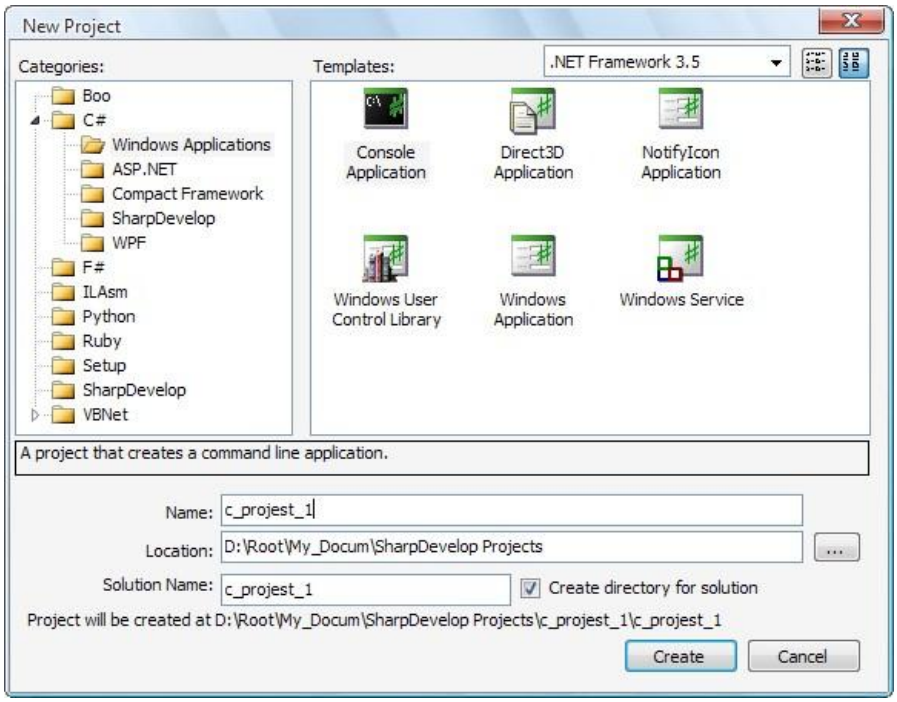


Рис. 3.17. Збереження створеного файлу

Вибрати у полі **Categories** мову проекту **C#-> Windows Applications**. У полі **Templates** вибрати тип проекту **Console.Application**. У поле **Name** увести англійськими літерами ім´я проекту та клацнути кнопку **Create**. Проект буде створено у папці, вибраній у полі **Location**.

3.3.4. Основи мови програмування C#

Платформа .NET Framework визначає середовище для підтримки, створення й виконання платформонезалежних гетерогенних додатків.

Особливостями даної платформи є незалежне від мови середовище виконання (Common Language Runtime, CLR) та бібліотека класів .NET.

Після компіляції створюється не виконуваний файл написаний у машинних кодах, а набір команд записаних проміжною мовою MSIL (Microsoft Intermediate Language). При запуску цього додатку в операційній системі його код написаний у MSIL транслюється JIT-компілятором (Justin-Time compiler - оперативний компілятор) в машинні коди, зрозумілі процесору комп'ютера.

Як не важко здогадатися, такий додаток буде працювати лише в операційних системах з установленою платформою NET Framework відповідної версії.

Усі мови програмування оперують певними даними. Дані можуть бути різних типів. Можна, наприклад, зберігати у пам'яті комп'ютера число, літеру, рядок літер, таблицю чисел, тощо. Платформа .NET використовує різні типи даних.

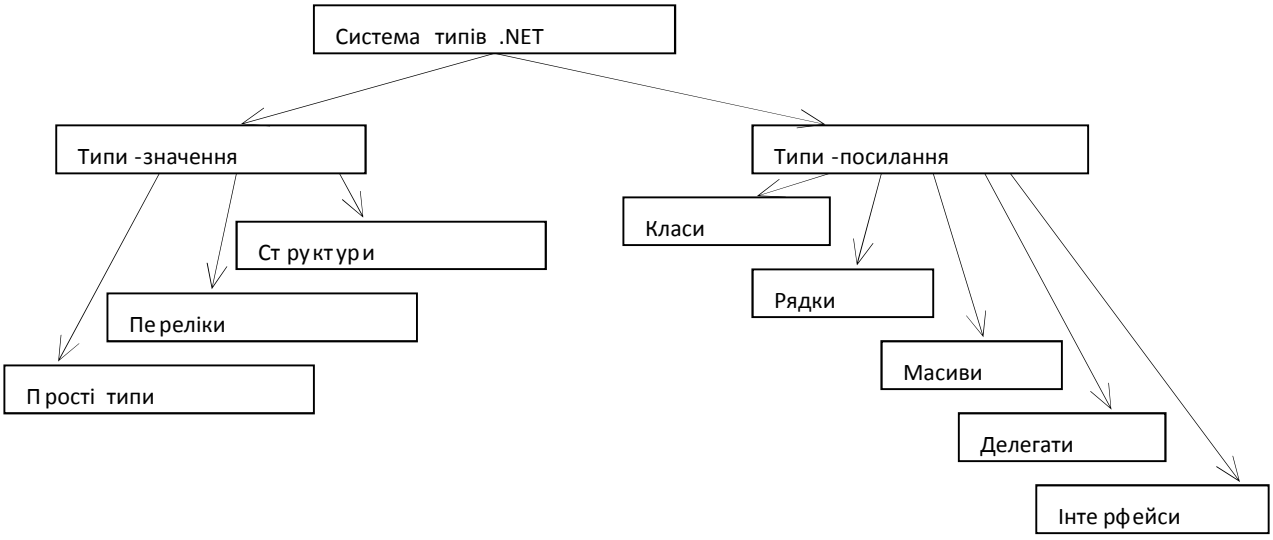


Рис. 3.18. Типи даних платформи .NET

Для збереження даних у програмі використовуються змінні та константи.

Змінною назвемо іменовану область пам'яті, в яку під час виконання програми можна записувати дані певного типу та зчитувати їх за потребою.

Константою назвемо іменовану область пам’яті, яка містить значення певного типу, яке можна зчитувати за необхідністю, але міняти не можна.

Для того, щоб використовувати змінну її треба оголосити та ініціювати. Для констант оголошення відбувається одночасно з ініціюванням.

Оголошення й ініціалізація змінних:

Тип_змінної ім'я_змінної[=значення];

Приклади:

```
int x; //оголошення змінної x
x=100; //ініціалізація змінної x
long w,z=100; //оголошення змінних w та z й ініціалізація z long q=100*z; //оголошення змінних з динамічною ініціалізацією
```

Оголошення й ініціалізація констант:

Тип_константи ім'я_константи[=значення]; Приклад

```
const double pi=3.14; //оголошення та ініціювання константи pi
```

C# - мова з суворим контролем типів. Є дві основні категорії вбудованих типів у C# - прості типи та типи-посилання. У табл. 3.1 наведено прості типи даних.

Таблиця 3.1

Основні прості типи даних в C#:

Ім’я типу	**Граничні значення**	**Приклад**
bool	**true** або **false**	true
char	Один символ	‘A’
sbyte	Число від -28 до 127	-16
byte	Число від 0 до 255 (8 розрядів)	32

Ім’я типу	Граничні значення	Приклад
short	Число від -32768 до 32767 (16 розрядів)	-256
ushort	Число від 0 до 65535 (16 розрядів)	128
int	Число від -2147483648 до 2147483648 (32 розряди)	-100000
uint	Число від 0 до 4294967295 (32 розряди)	200000
long	Число від -9223372036854775808 до 9223372036854775807 (64 розряди)	-5000000000L
ulong	Число від 0 до 18446744073709951615 (64 розряди)	10000000000L
float	Число від $1,5 \cdot 10^{-45}$ до $3,4 \cdot 10^{38}$ (32 розряди, точність до 7 десяткових розрядів)	375.95f
double	Число від $5,0 \cdot 10^{-324}$ до $1,7 \cdot 10^{308}$ (64 розряди, точність до 15-16 десяткових розрядів)	1.000000001d
Dsmal	Число від $1,0 \cdot 10^{-28}$ до $7,9 \cdot 10^{28}$ (128 розряди, точність до 15-16 десяткових розрядів)	75.95m
string	Послідовність символів Unicode	“Welcome”
object	Базовий для всіх інших типів тип	

Для оголошення імен змінних та констант не можна використовувати будь-які символьні послідовності.

Основні правила для вибору імен змінним та константам:

1. Першим символом має бути латинська літера, знак підкреслення «_», або символ «@».

2. Наступними символами імені можуть були будь-які символи латиниці, знак підкреслення чи цифри. Символи можуть були як з верхнього так і з нижнього регістру.

3. Для імен не можна використовувати ключові слова мови програмування (Main, int, if, for тощо). Звичайно вони виділяються кольором при їх наборі в редакторі.

Приклади допустимих імен: myBigScore, _qwer, a12

Приклади недопустимих імен: $qw, 54r, 1if, for, x-12

Слід розуміти, що мова C# є чутливою до регістру, і імена myWord та MyWord будуть різними.

За неписаним правилом програмістів імена простих змінних та констант мають починатися з символи нижнього регістру, імена складних змінних, методів та функцій з верхнього. Якщо ім'я складене з декількох слів, то перша літера другого слова пишеться великою.

Приклади імен змінних та констант: myWord, myString, myBestSolution.

Приклади імен методів та функцій: Fib(), Main(), Step(), MaxCount(), Max3().

Не рекомендується використовувати символ підкреслення для розділення частин імені (запис word_count виглядає незграбніше ніж wordCount).

3.3.5. Структура консольної програми

Ознайомившись з основними типами розглянемо структуру консольної програми на C#. Середовище програмування автоматично створить у новому проекті шаблон для майбутньої програми. Розберемо його.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace ConsoleApplication1
7   {
8   class Program
9   {
```

```
10    static void Main(string[] args)
11    {
12    }
13    }
14    }
```

Перші чотири рядки програми підключають модулі: System, System.Collections.Generic, System.Linq, System.Text. Модуль представляє спеціальну бібліотеку з інформацією про об'єкти, які використовуватиме наша програма. Дані модулі входять до системних і є частиною операційної системи. Програміст може написати та підключити після слова using свої модулі, проте це тема для іншої розмови. У середовищі SharpDevelop рядки 2-4 будуть відсутні.

У шостому рядку командою namespace ConsoleApplication1 створюється простір імен ConsoleApplication1, в якому описуватимуться всі об'єкти програми. Що таке **простір імен**? Це певна область коду, в якій всі об'єкти належать іншому батьківському об'єкту. Так, оголошення простору імен ConsoleApplication1 у шостому рядку означає, що код, уміщений у пару фігурних дужок (рядки 7 та 14) належатиме імені ConsoleApplication1. Метод Main, оголошений у десятому рядку також лежить у просторі імен ConsoleApplication1.

Символ «{» (фігурна дужка) починає блок команд, який закінчиться в 14 рядку. Пара символів «{}» називається операторними дужками, призначення яких обмежити блок команд. У даному випадку дужки з сьомого та чотирнадцятого рядків обмежують простір імен нашої програми.

У рядку 8 створюється клас Program. У кожній програмі на C# створюється обов'язковий метод Main().

Його створено в рядку 10. Слово void вказує на те, що дана програма не повертає результат обчислень у програму яка її викликала. Main – це стандартна назва основної програми (інша назва – **точка входу**). У дужках вказуються параметри, які передаються у дану з командного рядка запуску. Зверніть увагу на обов'язкове слово static перед заголовком методу Main().

Метод Main().є найголовнішим місцем у програмі, який і виконується при її запуску. Код програми слід вписувати після рядка 11 перед рядком 12.

Ввід та вивід інформації здійснюється за допомогою команд:

Write () вивід інформації в консоль.

WriteLine () вивід інформації в консоль та перехід на новий рядок.

Read () читання одного символу з консолі (клавіатури)

ReadLine () читання інформації з консолі та перехід на новий рядок.

Створимо програму, яка виведе повідомлення на екран.

Створивши порожній консольний проект після операторної дужки { в блоці основної програми додаємо код:

```
Console.WriteLine("Привіт. Я вивчаю C#");
Console.Read ();
```

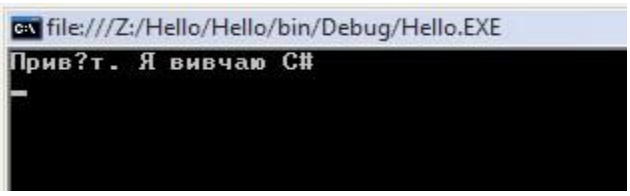
Приклад. Текст програми Hello

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   namespace Hello
6   {
7   class Program
8   {
9   static void Main(string[] args)
10  {
11  Console.WriteLine("Привіт. Я вивчаю C#");
12  Console.Read();
13  }
14  }
15  }
```

Запустимо програму клавішею F5, кнопкою з панелі інструментів, або з меню Отладка, Начать отладку.

На рисунку зображено вікно консолі, в яку команда виведе текст, набраний у лапках. Завершиться команда після натискання будь-якої клавіші з

клавіатури. Це реалізується командою Console.Read(), яка чекає натискання клавіші на клавіатурі.



Створимо програму, яка запитає в користувача ім'я та виведе для нього привітання.

Створимо порожній консольний проект.

В блоці основної програми додаємо код:

Приклад. Текст програми Dialog

```
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    namespace Dialog
6    {
7    class Program
8    {
9    static void Main(string[] args)
10   {
11   String s;
12   Console.WriteLine("Привіт. Як тебе звати?");
13   s = Console.ReadLine();
14   Console.WriteLine("Привіт. {0}. А я програма на C#.",s); 15
Console.Read();
16   }
17   }
18   }
```

У рядку 11 створюється змінна рядкового типу s.

Рядок 12 вміщає команду для виводу на екран тексту "Привіт. Як тебе звати?". Рядок 13 зчитує з клавіатури введений рядок та записує його до змінної s.

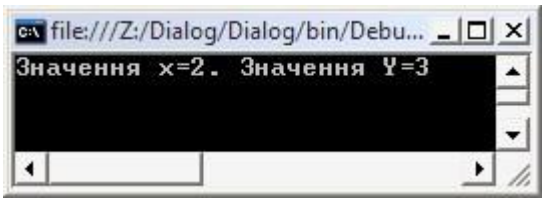
Рядок 14 виводить на екран текст "Привіт. {0}. А я програма на C#.". Замість {0} виводиться значення змінної s.

В платформі .NET прийнято стиль виводу, в якому при визначення рядкового літералу (виразу) з сегментами даних (підставленими значеннями

деяких змінних), значення яких залишається невідомим до часу виконання програми, дозволяється вказувати в літералі заповнювач з використанням фігурних дужок. В дужках вказується номер змінної (починаючи з 0), імена яких вказуються через коми після зазначеного літералу. Наприклад:

```
int x=2; int y=3;
Console.WriteLine("Значення x={0}. Значення Y={1}",x,y);
```

Після запуску отримаємо

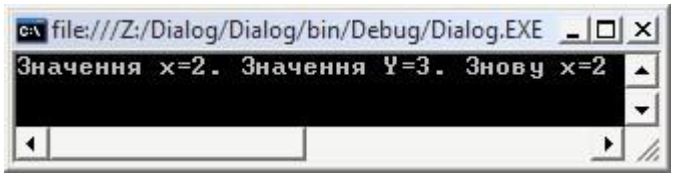



```
Console.WriteLine("Значення x={0}. Значення Y={1}. Знову x={0}",x,y);
```

Дозволяється повтор заповнювача в одному рядку-літералі. Наприклад, допустима така команда:

```
Console.WriteLine("Значення x={0}. Значення Y={1}. Знову x={0}",x,y);
```

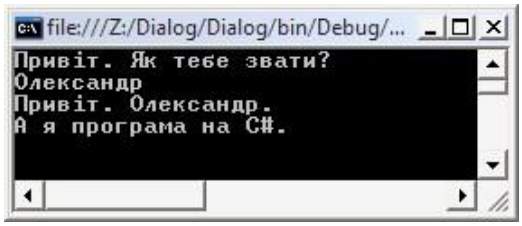
Після запуску отримаємо:



Змінимо 14 рядок у програмі з прикладу 2 додавши у рядок два символи

```
Console.WriteLine("Привіт. {0}. \nA я програма на C#.",s);
```

Після запуску програми отримаємо:



Як бачимо, «\n» не видно, але в тому місця, де його було вставлене рядок розірвався і його друга частина записалася з абзацу. Літерал «\n» називають керуючим символом переходу на новий рядок. Приведемо інші можливі керуючі символи:

\' Апостроф
\" Лапки
\\ Зворотній слеш
\a Системне сповіщення
\b Backspace
\f Початок наступної сторінки

\n Начало нового рядка
\r Return
\t Горизонтальний символ табуляції

3.3.6. Форматований вивід

Для виводу на консоль ми використовувати наступну конструкцію:

```
...
int x=23, y=-4;
...
Console.WriteLine("x={0}, y={1}", x, y); ...
```

Тут ми використовувати всередині лапок підстановочні знаки {0}, {1} і т. д. Змінні при цьому виводяться в форматі по замовчуванню. Для виводу в потрібному для нас форматі потрібно використовувати підстановочні знаки з параметрами. Ось деякі з них:

- d - десятковий формат. Дозволяє задати загальну кількість знаків (при необхідності число доповнюється зліва нулями).
- f - формат з фіксованою точністю. Дозволяє задати кількість знаків після коми.
- x - шістнадцятковий формат.
- c - грошовий формат (додає знак долара й показує два знаки після коми). □ e - вивід числа в експоненціальній формі.

Приклад використання форматованого виводу:

```
...
        int a=38;
        //Виводиться 0038
        Console.WriteLine("a={0:d4}", a);

        double pi=3.1415926;
        // Виводиться 3.14
        Console.WriteLine("pi={0:f2}", pi);
```

```
int b=255;            // Виводиться FF.
Console.WriteLine("b={0:X}", b);

int c=255;            // Виводиться ff.
Console.WriteLine("c={0:x}", c);


double d=1003.214;
// Виводиться $1, 003.14 в англійській версії Windows та
//1 003,14 р. в російській.
Console.WriteLine("d={0:c}", d);

double e=213.1;
// Виводиться 2.131000e+002
Console.WriteLine("e={0:e}", e);
```

У якості параметрів підстановочних знаків можна використовувати як малі, так і великі літери. Виключення – вивід числа в шістнадцятковому форматі (при використанні h цифри a..f будуть малими, при використанні H – великими).

У 13 рядку програми з прикладу 2 (s = Console.ReadLine()) зчитувалася інформація введена з клавіатури та запистувалася до рядкової змінної s Операція «=» називається **присвоювання.** Її зміст полягає в тому, що дані з правої від знака «=» частини записуються до змінної, розміщеної зліва від «=». У нашому випадку результат виконання функції Console.ReadLine()запишеться до змінної s.

3.3.6. Арифметичні операції

Задача більшості програм проводити обчислення. Із уроків математики ми знайомі з дійсними, цілими, натуральними числами. Над ними можна здійснювати операції додавання та віднімання, множення та ділення. До того ж існує багато функції (sin, cos, $\sqrt{}$ та інш)

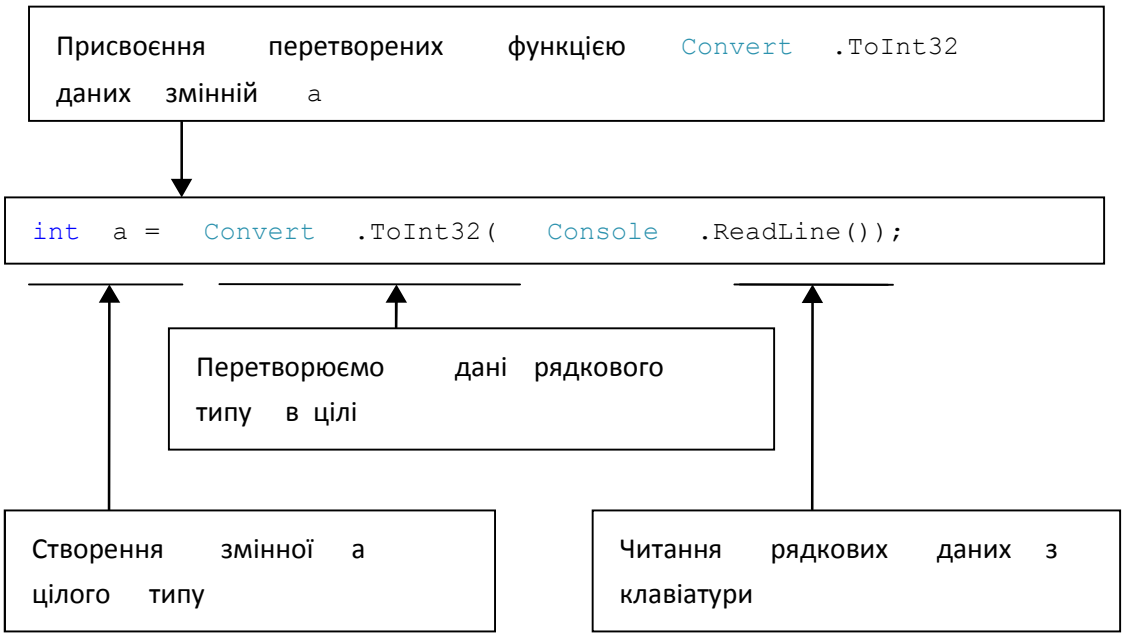
Приклад. Створимо програму, для розрахунку площі трикутника за відомими сторонами

Для розв’язання даної задачі спочатку уведемо значення довжин сторін a, b, c, обчислимо півпериметра трикутника p, та, скориставшись формулою Герона знайдемо площу трикутника. Виконавши обчислення, виведемо результати в консоль.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace triangle
7   {
8   class Program
9   {
10  static void Main(string[] args)
11  {
12  Console.WriteLine("Введіть сторону a");
13  int a = Convert.ToInt32(Console.ReadLine()); 14
Console.WriteLine("Введіть сторону b");
15          int b = Convert.ToInt32(Console.ReadLine()); 16
Console.WriteLine("Введіть сторону c");
17  int c = Convert.ToInt32(Console.ReadLine());
18  int p = (a + b + c) / 2;
19  Console.WriteLine("Півпериметр трикутника p={0}",p);
20  double s = Math.Sqrt(p*(p-a)*(p-b)*(p-c));
21  Console.WriteLine("Площа трикутника s={0}", s);
22  Console.ReadLine();
23  }
24  }
25  }
```

В рядку 12 виводиться в консоль рядок "Введіть сторону a".

Проаналізуємо рядок 13 .



Після зчитування даних про сторони трикутника в змінні a, b, c у рядку 18 обчислюється та присвоюється змінній p значення півпериметра, а в рядку 20 присвоюється змінній s значення площі трикутника.

З мови C++ мова C# взяла префіксні та постфіксні операції додавання. Операцію збільшення змінної X на 1 можна записати так

Звичайнай запис	Постфіксний запис
X=X+1;	X++;

Останній запис має перевагу у швидкості виконання та в лаконічнішому записі.

У наступному прикладі до значення змінної X додамо значення змінної Y, а результат запишемо до змінної X

Звичайнай запис	Постфіксний запис
X=X+Y;	X += Y;

Приклад 3.2. Програма, яка демонструє постфіксні та префіксні операції

```
1  using System;
2  using System.Collections.Generic;
```

```
3    using System.Linq;
4    using System.Text;
5
6    namespace operations
7    {
8    class Program
9    {
10   static void Main(string[] args)
11   {
12   int x = 4;
13   x++;
14   Console.WriteLine("Перше додавання X={0}",x); 15          x++;
16   Console.WriteLine("Друге додавання X={0}",x);
17   x--;
18   Console.WriteLine("Перше віднімання X={0}",x); 19          x += 5;
20   Console.WriteLine("Після збільшення на 5 X={0}",x);
21   Console.ReadLine();
22   }
23   }
24   }
```

Приклад. Інший варіант програми, яка демонструє постфіксні та префіксні операції

```
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5
6    namespace operations
7    {
8    class Program
9    {
10   static void Main(string[] args)
11   {
12   int x = 4;
13   Console.WriteLine("X={0}\n X={1}\n X={2}\n X={3}",x++,x++,x--,x--); 14
x = 4;
15          Console.WriteLine("X={0}\nX={1}\nX={2}\nX={3}", ++x, ++x, --x, --x);
16        Console.ReadLine();
17   }
18   }
```

```
19    }
```

Виконавши дану програму, розберемося з відмінностями в виконанні 13-го та 15-го рядків.

При виконанні операції x++ спочатку здійснюється вивід значення змінної на екран, після чого над змінною здійснюється операція додавання одиниці. Операція ++x спочатку додає до змінної одиницю, після чого виводить її значення на екран.

Наступні операції ілюструють відмінність між постфіксними та префікс ними операціями:

- x = ++y; //значення y збільшується на 1, після чого нове значення записується до змінної x;
- x = y++; // значення y записується до змінної x, після чого значення змінної y збільшується на 1.

Приклад 3.4. Ілюстрація відмінності між префікс ними та постфіксними операціями

```
1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5

6     namespace operations
7     {
8     class Program
9     {
10    static void Main(string[] args)
11    {
12    int x ; int y = 1;
13    x = ++y;
14    Console.WriteLine("X={0}\nY={1}", x, y);
15    Console.WriteLine();
16    y = 1;          17          x = y++;
18    Console.WriteLine("X={0}\nY={1}", x, y);
19    Console.ReadLine();
20    }
21    }
}
```

3.3.7. Класи, об’єкти та структури

Нехай нам потрібно створити програму для опису тварин в зоопарку.

Об’єктами назвемо тварин, які є в зоопарку.

Будь-який об’єкт може мати властивості, та методи. **Властивістю** назвемо деяку характеристику об’єкта, наприклад, маса, висота, кількість ніг тощо. Об’єкти можуть мати **нащадків.** Наприклад, об’єкт кінь є нащадком об’єкта ссавець. Нащадки можуть мати деякі властивості батьківського об’єкта. Проте інші властивості можуть відрізнятися.

Методом назвемо операцію, дію, яку можна здійснити з даним об’єктом. Наприклад, об’єкт кінь може мати методи **введення маси**, **вивід імені** тощо.

Класом назвемо групу об’єктів у нашій програмі. При описі класу, описуються об’єкти, які містяться в даному в ньому. Клас дає загальний опис об’єктів, вказує на що вони схожі.

Опишемо клас об’єктів Тварина (Animal). Він міститиме такі об’єкти: Кінь (Horse), Лев (Leo), Риба (Fish) тощо.

Опис класу здійснюється службовим словом class

```
class Animal
{
        public string kindOfAnimal;        public string name;
public int numberOfLegs;          public float mass;


}
```

У даному фрагменті створюються новий клас Animal. У ньому визначені властивості kindOfAnimal (вид тварини), name (Ім’я), numberOfLegs (кількість ніг), mass (маса). Службове слово public вказує на те, що дане поле буде доступне для всіх нащадків даного класу.

Приклад. Оголошення класу та створення об’єкта

```
1 using System;
2
```

```
3   namespace c1
4   {
5   class Program
6   {
7   class Animal
8   {
9   public string kindOfAnimal;
10  public string name;
11  public int numberOfLegs;
12  public float mass;
13  }
14  public static void Main(string[] args)
15  {
16  Animal Horse = new Animal();
17  Horse.name = "Буцефал";
18  Horse.mass = 300;
19  Console.WriteLine("Ім'я тварини {0}", Horse.name);
20  Console.WriteLine("Її маса {0} кг", Horse.mass);
21  Console.ReadLine();
22  }
23  }
24  }
```

У 7-13 рядках описується клас Animal. Слово public перед описом кожного з його властивостей (полів) вказує на доступність (відкритість) даного поля ззовні класу.

У рядку 16 створюється об'єкт Horse, який є екземпляром класу Animal.

Програма в 17-18 рядках записує до полів об'єкту Horse його ім'я та масу, а в рядках 19-20 виводить їх на екран.

Методом назвемо набір команд, властивих нашому об'єкту. Наприклад, команда для виводу на екран імені та маси тварини. Опишемо метод для виводу інформації про тварину в класі Animal.

Метод, на відміну від змінної, може мати параметри. Тому після його імені ставимо символи круглих дужок, в яких за необхідності передаються параметри.

```
public void ShowAnimal ()
{
```

```
    Console.WriteLine("Ім`я тварини {0}",this.name);
    Console.WriteLine("Її маса {0} кг",this.mass);
   }
```

Зверніть увагу на заміну імені об’єкта Horse на службове слово this. Об’єкт Horse під час опису класу Animal буде недоступним, тому виникне помилка. Під час же виконання методу в основній програмі замість слова this компілятор підставить ім’я об’єкта.

Приклад. Використання метода без параметрів

```
1 using System;
2
3     namespace c1
4     {
5     class Program
6     {
7     class Animal
8     {
9     public string kindOfAnimal;
10    public string name;
11    public int numberOfLegs; 12          public float mass;
13    public void ShowAnimal()
14    {
15    Console.WriteLine("Ім’я тварини {0}", this.name);
16    Console.WriteLine("Її маса {0} кг", this.mass);
17    }
18    }
19    public static void Main(string[] args)
20    {
21    Animal Horse = new Animal();
22    Horse.name = "Буцефал";
23    Horse.mass = 300;
24    Horse.ShowAnimal();
25    Console.Read();
26    }
27    }
28    }
```

В 13-17 рядках описано метод ShowAnimal (). Слово public вказує на доступність даного методу з інших частин програми. Слово void вказує на те, що даний метод не передає даних в основну програму.

В рядку 24 даний метод викликається з основної програми.

Розглянемо використання методів з параметрами. Створимо метод InputName та InputMass для вводу даних про ім’я та масу тварини. Додамо до опису класу Animal метод

```
Animal InputName (Animal s)
    {
                        s.name=Console.ReadLine();
                return s;
    }
```

Приклад. Використання метода з параметром

```
1 using System;
2
3     namespace c1
4     {
5     class Program
6     {
7     class Animal
8     {
9     public string kindOfAnimal;
10    public string name;
11    public int numberOfLegs; 12          public float mass;
13    public void ShowAnimal()
14    {
15    Console.WriteLine("Ім’я тварини {0}", this.name);
16    Console.WriteLine("Її маса {0} кг", this.mass);
17    }
18    public Animal InputName(Animal s)
19    {
20    s.name = Console.ReadLine();
21    return s;
22    }
23    }
24    public static void Main(string[] args)
25    {
26    Animal Horse = new Animal();
27    Horse = Horse.InputName(Horse);
```

```
28    Horse.mass = 300;
29    Horse.ShowAnimal();
30    Console.Read();
31    }
32    }
33     }
```

У 18-22 рядках описується метод **public** Animal InputName (Animal s) InputName з параметром

Після слова return в рядку 21 указується змінна, дані з якої повертаються з методу в основну програму.

У рядку 27 викликається метод InputName, в який передається параметр Horse, та повертається з нього нове значення Horse.

Структура має призначення подібне до класу, проте спосіб її створення та використання нею пам'яті відрізняється від класів. Клас – це свого роду посилання на адресу пам'яті, в якій зберігається дані. Структурою ж це власне самі значення, при оголошенні змінної типу, заданого структурою, створюються нові змінні. Приклад оголошення структури:

```
struct elem
{      long isn;      string type;      string name;    };
```

3.3.8. Логічний тип. Операції з даними логічного типу

Крім арифметичних виразів у мовах програмування ще один тип виразів – логічний. Логічним назвемо такий вираз, результатом обчислення якого може бути істина (True) або хиба (False).

Для опису такого типу даних використовується слово bool (скорочення від boolean), читається «Буль».

В якості операторів у логічних виразах використовуються операції порівняння:

Назва	Позначення
Більше	>
Менше	<
Менше рівне	<=
Більше рівне	>=
рівне	= =
не рівне	!=

Та логічні операції

Назва	Позначення
Умовне (логічне множення) І	&&
Умовне (логічне додавання) АБО	||
Заперечення (інверсія) НІ	!

Логічна операція && (І) дає істинний результат лише тоді, коли обидва операнди істинні.

Логічна операція || (АБО) дає істинний результат тоді, коли хоча б один з операндів істинний.

Логічна операція ! (НІ) завжди дає результат протилежний значенню операнда. Приклади простих логічних виразів:

Вираз	Результат
5==7	False
4==4	True
6>=6	True
6!=8	True

Замість чисел можуть використовуватися змінні, яким присвоєно числа.

Приклади складних логічних виразів

Вираз	Результат

5>2 && 3==5	False
5>2 || 3==5	True

При складанні як арифметичних, так і логічних виразів слід ураховувати пріоритет виконання різних операцій. Для зміни порядку використовуються дужки

Пріоритет	Категорія	Операція	Порядок
0	Первинні	(expr), x y, x->y, f(x), a[x], x++, x--, new, typeof(t), checked(expr), unchecked(expr)	Зліва направо
1	Унарні	+, -, !, ~, ++x, --x, (T)x, sizeof(t)	Зліва направо
2	Мультиплікативні (Множення)	*, /, %	Зліва направо
3	Адитивні (Додавання)	+, -	Зліва направо
4	Зсув	<< ,>>	Зліва направо
5	Відношеня, перевірка типів	<, >, <=, >=, is, as	Зліва направо
6	Еквівалентність	==, !=	Зліва направо
7	Логічне І (AND)	&	Зліва направо
8	Логічне виключення АБО (XOR)	^	Зліва направо
9	Логічне АБО (OR)	|	Зліва направо
10	Умовне логічне І	&&	Зліва направо
11	Умовне логічне АБО	||	Зліва направо
12	Умовний вираз	? :	Справа наліво
13	Присвоювання	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=	Справа наліво
	Склеювання з null	??	
14	Лямбда-оператор	=>	Справа наліво

Розглянемо задачу визначення можливості існування трикутника за довжинами його сторін

Приклад. Використання логічних виразів

```
1   using System;
2   namespace c_bool_1
3   {
```

```
4   class Program       {
5   public static void Main(string[] args)
6   {
7           Console.WriteLine("Уведіть довжини трьох сторін трикутника
                    щоразу натискаючи Enter");
8   double a=Convert.ToDouble(Console.ReadLine());
9   double b=Convert.ToDouble(Console.ReadLine());
10  double c=Convert.ToDouble(Console.ReadLine());
11  bool sol = a+b>c && a+c>b && b+c>a;
12  Console.WriteLine("Результат {0}",sol);
13  Console.ReadKey(true);
14  }
15  }
16  }
```

3.3.9. Операція розгалуження

Більшість задач, які ми розв'язуємо в повсякденному житті вимагає в нас вибору одного способу дій з двох чи більше альтернативних. Спосіб програмування, в якому реалізується операція вибору називають розгалуженням.

Для здійснення розгалуження в мові C# використовується оператор if (якщо).

Загальний вигляд оператора if:

if (<умова>) <команда_1>; else <команда_2>; - повна форма; if (<умова>) <команда_1>; - скорочена форма.

Напишемо фрагмент програми для визначення подільності введеного з клавіатури числа на 5.

Приклад. Використання розгалуження

```
1   using System;
2   namespace c_if1
3   {
4   class Program
5   {
```

```
6       public static void Main(string[] args)
7       {
8       Console.WriteLine("Введіть ціле число та натисніть Enter");
9       string s = Console.ReadLine();
10      int n = Convert.ToInt32(s);
11      if (n%5==0) Console.WriteLine("Число {0} ділиться без залишку на 5",
                                                    n);
12      else Console.WriteLine("Число {0} не ділиться без залишку на 5", n);
13      Console.ReadKey(true);
14      }
15      }
16      }
```

У рядку 9 зчитується з клавіатури рядок методом ReadLine(); до змінної s.

У 10 рядку рядкове значення змінної s конвертується в ціле число, яку записується до змінної n.

У 11-12 рядках записано оператор розгалуження, який аналізує зміст умови (n%5==0), і якщо вона істинна, виконує команду.

Console.WriteLine("Число {0} ділиться без залишку на 5",n), якщо ж не істинна, то команду Console.WriteLine("Число {0} не ділиться без залишку на 5",n).

Окремої уваги заслуговує написання умови. По перше, у 11 рядку записано математичну операцію пошуку залишку від ділення %. Результатом обчислення значення виразу n%5 буде залишок від ділення числа n на число 5. Два символа дорівнює називається операцією порівняння. Ці символи ставляться між змінними чи константами для перевірки їх на спів падання.

Операції порівняння в C#

Вираз	Опис
x==y	Повертає true (істину), якщо x дорівнює y, в іншому випадку повертає false (хибу)
x!=y	Повертає true (істину), якщо x не дорівнює y, в іншому випадку повертає false (хибу)

x>y	Повертає true (істину), якщо x більше y, в іншому випадку повертає false (хибу)
x<y	Повертає true (істину), якщо x менше y, в іншому випадку повертає false (хибу)
x>=y	Повертає true (істину), якщо x більше або дорівнює y, в іншому випадку повертає false (хибу)
x<=y	Повертає true (істину), якщо x менше або дорівнює y, в іншому випадку повертає false (хибу)

Якщо в задачі вимагається обирати не з двох, а з трьох чи більше варіантів, то після слова else першого умовного оператора можна вставити інший.

Для прикладу розглянемо просту алгебраїчну задачу по визначенню коренів квадратного рівняння виду $a \cdot x^2 + b \cdot x + c = 0$. Як відомо з курсу шкільної алгебри спочатку обчислюється дискримінант $D = b^2 - 4 \cdot a \cdot c$. Якщо $D < 0$, то рівняння не має коренів Якщо $D = 0$, рівняння має один корінь, який обчислюється за формулою $x = \frac{-b + \sqrt{D}}{2 \cdot a}$. Якщо $D > 0$, то рівняння має два корені, які обчислюються за формулами $x_1 = \frac{-b + \sqrt{D}}{2 \cdot a}$ та $x_2 = \frac{-b - \sqrt{D}}{2 \cdot a}$.

Слідуючи ідеології об’єктно-орієнтованого програмування створимо новий клас kvRivn з полями-властивостями a, b та c й методами:

- для вводу коефіцієнтів inputKoef;
- для обчислення та виводу коренів rozv.

Приклад. Використання розгалуження

```
27  К

 a=System.Convert.ToDouble(Console.ReadLine());
```

Розглянемо задачу про пошук більшого з трьох. Оформимо код у вигляді функції з трьома параметрами.

Приклад. Використання розгалуження

```
1   using System;
2   namespace c if3
3   {
4   class Program
5   {
6   static int max3(int a, int b, int c)
7   {
8   int max;
9   if (a > b) max = a; else max = b;
10  if (c > max) max = c;
11  return max;
12  }
13  public static void Main(string[] args)
14  {
15  Console.WriteLine("Введіть три числа, натискаючи щоразу Enter");
16  int a = Convert.ToInt32(Console.ReadLine()); 17          int b =
Convert.ToInt32(Console.ReadLine());
18  int c = Convert.ToInt32(Console.ReadLine());
19  Console.WriteLine("Максимальне число {0}", max3(a, b, c)); 20
Console.ReadKey(true);
21  }
22  }
23  }
```

Існує безліч задач, в яких вибирати доводиться не з двох варіантів, а з більшого числа. У такому випадку зручно використовувати оператор вибору switch .

```
        switch (caseSwitch)  // визначення ключа для вибору
        {
           case 1: // якщо caseSwitch дорівнює 1
              Console.WriteLine("Case 1"); // виводимо на екран "Case 1"
break; // виходимо з оператора switch      case 2: // якщо caseSwitch дорівнює 1
              Console.WriteLine("Case 2"); // виводимо на екран "Case 2"
break; // виходимо з оператора switch      default: // у всіх інших випадках
               Console.WriteLine("Default case"); // виводимо на екран "Default case"
break; // виходимо з оператора switch
        }
```

Розв'яжемо задачу визначення кількості днів у місяці не високосного року .

Приклад. Використання оператора вибору

```
1    using System;
2    namespace c if4
3    {
4    class Program
5    {
6    public static void Main(string[] args)
7    {
8          Console.WriteLine("Введіть номер місяця");
9    int month = Convert.ToInt32(Console.ReadLine());
10   switch (month)
11   {
12   case 1: case 3: case 5:  case 7:
13   case 8: case 10: case 12:
14   Console.WriteLine("У цьому місяці 31 день");
15   break;
16    case 2:
17    Console.WriteLine("У цьому місяці 28 днів");
18  break;
19   case 4:case 6: case 9:case 11:
20   Console.WriteLine("У цьому місяці 30 днів");
21   break;
22   default:
23            Console.WriteLine("Невірно введено номер місяця"); 24
break;
25   }
26   Console.ReadKey(true);
```

```
27    }
28    }
29    }
```

3.3.10. Масиви

Досі в задачах ми розглядували змінні, в кожну з яких можна записати лише одне значення в даний момент часу. Існує багато задач, в яких виникає необхідність збереження в пам’яті послідовності значень однієї й тієї ж величини.

Наприклад, розглянемо задачу про збереження в пам´яті та обробки результатів метереологічних вимірювань температури повітря протягом тижня.

Змінна t (температура) повинна прийняти 7 значень за кожен з днів вимірювання. Зрозуміло, можна було б описати 7 змінних: t1, t2, … , t7. Проте цей метод ускладнить процес обробки даних.

Масивом назвемо послідовність однотипних даних, які мають одне ім´я, а розрізняються за індексом.

Перед використанням масиву, як і будь-якої змінної, його треба створити. Нехай нам потрібно створити масив t з кількістю елементів 7, і кожен з елементів дійсного типу (**double**) . **double** [ ] t = **new double** [7];

Такий масив, який представляє послідовність із заданої кількості даних певного типу назвемо одновимірним. Для доступу до певного елемента масиву після імені масиву в квадратних дужках вказуватимемо числовий індекс. Найпершим елементом масиву в нашому випадку буде елемент з індексом 0 (нуль), тобто t[0]. Враховуючи, що елементів масиву 7, то останнім елементом буде елемент t[6]. При спробі звернутися до елемента з номером поза діапазону [0..6] виникне помилка, яка може привести до аварійного переривання програми.

```
t[0]=15 6; //запис до 1-го елементу масиву t значення 15 6 t[2]=10;
//запис до 3-го елементу масиву t значення 10
```

Console.**WriteLine**(t[1]); //вивід на екран значення 2-го елементу масиву t

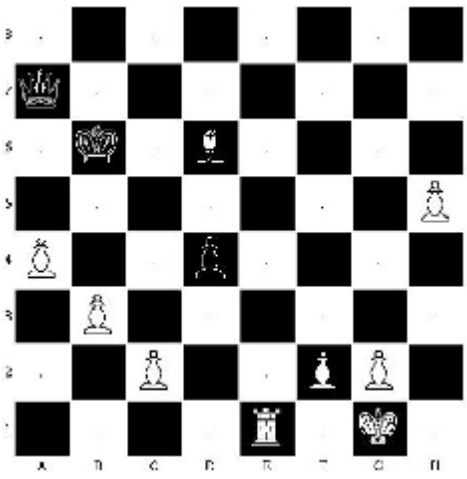
Значення до масиву можна записувати як операцією присвоювання, так і задавати одночасно з його створенням.

double [] t = **new double** [7] {11,13 5,15,10,9 5,13 1,15 4};

Щоб розібратися з багатовимірними масивами уявимо шахову дошку Кожна клітинка задається двома індексами: буквою (від a до h) та цілим числом (від 1 до 8).

Шахівниця – це приклад двовимірного масиву.

Процес створення двовимірного масиву схожий з одновимірним .



int[,] Array2 = new int[2, 3]; або int[,] Array2 = { { 1, 2, 3 }, { 4, 5, 6 } };

Для запису значення до елементу масиву Array2[1,2]=43;

Різновидом багатовимірних масивів є зубчасті масиви. У ньому кількість елементів у кожному рядку задається окремо. Прикладом задачі з використанням зубчастих масивів є масив місяців та днів у році.

Наступний фрагмент програми ілюструє різні способи створення масивів.

```
class TestArrays
 {
   static void Main()
   {
     // Оголошення одновимірного масиву        int[] array1 = new int[5];
```

```
            // Оголошення з одночасним заданням елементів одновимірного
масиву
            int[] array2 = new int[] { 1, 3, 5, 7, 9 };

            // Альтернативний спосіб оголошення масиву
            int[] array3 = { 1, 2, 3, 4, 5, 6 };

            // Оголошення двовимірного масиву
            int[,] multiDimensionalArray1 = new int[2, 3];

            // Оголошення з одночасним заданням елементів двовимірного
масиву
            int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

            // Оголошення зубчастого масиву
            int[][] jaggedArray = new int[6][];

            // Встановлення значення елементам першого рідка зубчастого
масиву
            jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
        }
    }
```

3.3.10.1. Уведення даних до масиву з клавіатури та вивід масиву в консоль

Масиви досить популярні при розробці програм. Без них неможливе збереження та обробка інформації.

Найпершою задачею є введення інформації до пам´яті комп´ютера. Інформацію можна читати з файла, вводити з клавіатури чи іншого пристрою вводу (сканер, веб-камера, звукова карта тощо).

Найпростішим та найактуальнішим для нас є введення інформації з клавіатури. Розглянемо процес зчитування з клавіатури цілих чисел та наступним занесенням його до масиву в пам´яті комп´ютера.

Так як масив, це послідовність однотипних даних, кожне з яких обробляється аналогічно до попереднього, то в основі нашої програми буде цикл.

Приклад 9.1. Уведення даних до одновимірного масиву з клавіатури

```
1    using System;
2    namespace c_vvidmass
3    {
4    class Program
5    {
6    public static void Main(string[] args)
7    {
8  Console.WriteLine("Скільки елементів у масиві?");
9    int n=Convert.ToInt32(Console.ReadLine());
10   int  [] Array = new int[100];
11
Console.WriteLine("Вводьте дані, щоразу натискаючи Enter");
12   for (int i=0;i<=n-1; i++)
13   Array[i]=Convert.ToInt32(Console.ReadLine());
14  Console.WriteLine("Ввід масиву завершено ");
15   for (int i=0;i<=n-1; i++)
16   Console.WriteLine("Array[{0}]={1}",i,Array[i]);
17   Console.ReadKey(true);
18   }
19   }
20   }
```

Масив уводиться з клавіатури в циклу, записаному в 12-13 рядках. У 15-16 рядках дані з уже введеного масиву виводяться на екран.

Приклад. Уведення даних до двовимірного масиву з клавіатури

```
1    using System;
2    namespace c_vvidmass2
3    {
4    class Program
5    {
6    public static void Main(string[] args)
7    {
8   Console.WriteLine("Скільки рядків у масиві?");
9            int rowCount=Convert.ToInt32(Console.ReadLine()); 10
     Console.WriteLine("Скільки стовпців у масиві?");
11   int colCount=Convert.ToInt32(Console.ReadLine());
12   int  [,] Array2 = new int[100,100];
```

```
13    for (int row=0;row<=rowCount-1; row++)
14    {
15  Console.WriteLine("Вводьте дані {0}-го рядка",row+1);
16    for (int col=0;col<=colCount-1; col++)
17    Array2[row,col]=Convert.ToInt32(Console.ReadLine());
18    }
19  Console.WriteLine("Ввід масиву завершено ");
20    for (int row=0;row<=rowCount-1; row++)
21    {
22    for (int col=0;col<=colCount-1; col++)
23    Console.Write(" {0:d3}",Array2[row,col]);
24    Console.WriteLine();
25    }
26    Console.ReadKey(true);
27    }
28    }
29    }
```

Уведення даних до масиву здійснюється в двох циклах for. У першому циклі (рядок 13) переглядаються почергово всі рядки таблиці, якою представлено масив, у другому циклі (рядок 16) переглядаються комірки в рядку (стовпці).

Також у двох циклах виводиться вміст масиву на екран (рядки 20-25).

У рядку 23 використовується форматований вивід результату.

3.3.10.2. Пошук у масиві елемента з заданими властивостями

Опишемо структуру children (діти) з полями: age (вік), name (ім´я), fullName (прізвище), form (клас).

Створимо масив типу children. Уведемо дані до масиву. Знайдемо та надрукуємо дані про учнів віком старші за 10 років.

Приклад. Пошук елемента з заданими властивостями

```
1     using System;
2     namespace c_find1
```

```
3   {
4   class Program
5   {
6   struct children //структура опису даних про учня
7   {
8   int age;
9   string name;
10  string fullName;
11  string form;
12  public void inputChild()
13  //Метод для вводу даних про учня
14  {
15  Console.WriteLine("Уведіть прізвище учня");
16  fullName=Console.ReadLine();
17  Console.WriteLine("Уведіть ім'я учня");
18  name=Console.ReadLine();
19  Console.WriteLine("Уведіть клас, в якому навчається учень");
20  form=Console.ReadLine();
21              Console.WriteLine("Уведіть вік учня");
22  age=Convert.ToInt32(Console.ReadLine());
23  }
24  public bool findChild(int ag)
25  //Метод для пошуку даних про учня
26  {
27  if (age>ag) return true; else return false;
28  }
29  public void printChild()
30  //Метод для друку інформації про учня
31  {
32  Console.WriteLine(fullName+" "+name);
33  }
34  }
35  public static void Main(string[] args)
36  {
37      Console.WriteLine("Уведіть кількість учнів");
38  int n=Convert.ToInt32(Console.ReadLine());
39  children [] child = new children[100];
40  for (int i=0; i<n; i++)
41  child[i] inputChild();
42  for (int i=0; i<n; i++)
43  if (child[i] findChild(10))
44  child[i] printChild();
```

```
45    Console.ReadKey(true);
46    }
47    }
48    }
```

У рядках 6-34 описується структура children з чотирма полями та трьома методами. У рядку 39 оголошується масив структур child. У рядках 40-41 в циклі викликається n разів метод для вводу даних про учня, у рядках 42-44 шукаються дані в масиві про учнів з віком більшим за 10 та знайдені записи друкуються за допомогою метода printChild() оголошеному в структурі.

3.3.10.3. Пошук найбільшого елемента

В прямокутній матриці визначити номет рядка та стовпця комірки, в якій знаходиться найбільший елемент.

Приклад. Пошук найбільшого елемента в таблиці

```
1     using System;
2     namespace c_findmax
3     {
4     class Program
5     {
6     public static void Main(string[] args)
7     {
8     int [,] arr=new int[,]{{4,6,2,3,1},{5,8,7,11,2},{9,3,5,4,1}};
9     int max_c=0;
10    int max_r=0;
11    for (int r=0;r<=2;r++)
12    for (int c=0;c<=4;c++)
13    if (arr[max_r,max_c]<arr[r,c])
14    {
15    max_c=c;
16    max_r=r;
17    }
18  Console.WriteLine("Найбільший елемент міститься\n" +
19    "у {0}-му рядку\n" +
20    "та {1}-му стовпці",max_r+1,max_c+1);
```

```
21      Console.ReadKey(true);
22      }
23      }
24      }
```

3.3.10.4. Пошук елемента в упорядкованому масиві

Нехай задано упорядкований за зростанням масив $a1, a2, \ldots, an$. Необхідно знайти на вивести на консоль номер елемента, значення якого дорівнює деякому числу. Застосуємо так званий бінарний пошук.

Ідея бінарного пошуку.

Ділимо масив навпіл, порівнюємо середній елемент масиву з шуканим. Можливі три варіанти. Для кожного з варіантів робимо відповідну дію.

1. Якщо елемент дорівнює шуканому, то друкуємо його номер та завершуємо програму.
2. Якщо елемент більший за шуканого, то ділимо навпіл першу частину та повторюємо алгоритм спочатку.
3. Якщо елемент менший за шуканого, то ділимо навпіл другу частину та повторюємо алгоритм спочатку.

Приклад. Пошук у відсортованому масиві (бінарний пошук)

```
1       using System;
2       namespace c_binary
3       {
4       class Program
5       {
6       public static void Main(string[] args)
7       {
8       int [] a ={3,6,9,12,17,26,29,36,39,43,51,67};
9                                   Console.WriteLine("Яке число
шукаємо?");
10      int x=Convert.ToInt32(Console.ReadLine());
11      int l=1; int r=12; int k=0; int k0;
```

```
12      while (a[k]!=x)
13      {
14      k0=k;
15      k=(r+l)/2;
16      if (x<a[k]) r=k-1;
17      if (x>a[k]) l=k+1;
18      if (k0==k) {k=-1; break;}
19      }
20
Console.WriteLine("Шуканий елемент під номером {0}",k+1);
21      Console.ReadKey(true);
22      }
23      }
24      }
```

3.3.10.5. Сортування масиву

Нехай дано масив $a_1, a_2, \ldots, a_n$. Треба перетворити його так, щоб у модифікованому масиві виконувалася умова $a_1 \leq a_2 \leq \ldots \leq a_n$.

Метод ***бульбашкового сортування*** Не раціональний, але простий для розуміння.

Фіксуємо перший елемент масиву й почергово порівнюємо з ним усі елементи розміщені правіше. Якщо правий елемент менший за перший, міняємо їх місцями. Повторюємо дану процедуру для другого елемента І так до тих пір, поки не дійдемо до кінця масиву

Приклад. «Бульбашкове сортування»

```
1       using System;
2       namespace c_sortBuble
3       {
4       class Program
5       {
```

```
6     public static void Main(string[] args)
7     {
8     int n=12; int tmp;
9     int [] a = {7,4,6,1,9,5,22,4,21,8,12,11};
10    for (int i=0;i<=n-2;i++)
11    for (int j=i+1; j<=n-1; j++)
12    if (a[j]<a[i])
13    {
14    tmp=a[i];
15    a[i]=a[j];
16    a[j]=tmp;
17    }
18  Console.Write("Відсортований масив ");
19    for (int i=0;i<=n-1;i++)
20    Console.Write(" {0}",a[i]);
21    Console.ReadKey(true);
22    }
23    }
24    }
```

Сортування здійснюється в рядках 10-17.

У рядках 14-16 здійснюється обмін елементів з номерами i та j.

У рядках 19-20 відсортований масив виводиться на консоль.

Є досить багато алгоритмів упорядкування послідовностей. Існують алгоритми впорядкування вже частково впорядкованих масивів, алгоритми, ефективніші за часом виконання, але такі, що використовують більше пам'яті комп'ютера, алгоритми, що впорядковують дуже великі за об'ємом послідовності, які не можна повністю завантажити в пам'ять комп'ютера тощо.

Неважко здогадатися, що всі ці програми можна використовувати і для рядкових величин Необхідно лише створити масив із елементів типу string.

3.3.11. Цикли

Для забезпечення можливості повторного використання фрагменту програми в усіх мовах програмування передбачено можливість створення циклів.

Розглянемо використання циклу for.

for (init-expression ; cond-expression ; loop-expression) statement ;

init-expression - задання початкового значення лічильника циклу.

cond-expression - умова продовження циклу, яка перевіряється на початку кожного повтору (ітерації).

loop-expression - операція, яка вказує те,яким чином змінюється значення лічильника циклу в кожній наступній ітерації.

statement - команда, яка виконується в тілі циклу задану кількість разів.

Лічильником циклу назвемо змінну величину, значення якої змінюється в кожній ітерації й від якого залежить продовжуватиметься цикл чи завершиться.

Напишемо програму для обчислення суми всіх чисел від n1 до n2.

Приклад. Використання циклів

```
1   using System;
2   namespace c_for1
3   {
4   class Program
5   {
6   public static void Main(string[] args)
7   {
8   Console.WriteLine("Введіть значення n1 та n2 натискаючи після
кожного Enter");
9   int n1 = Convert.ToInt32(Console.ReadLine());
10  int n2 = Convert.ToInt32(Console.ReadLine());
11  int s = 0;
12  for (int i = n1; i <= n2; i++)
13  s += i; //до значення s додати поточне значення i
14  Console.WriteLine("Шукана сума s={0}", s);
15  Console.ReadKey(true);
16  }
```

```
17  }
18  }
```

Цикл записано в рядках 12-13. В якості лічильника циклу створюється та ініціюється значенням n1 змінна i. Умовою виходу з циклу є i <= n2. Ця умова перевіряється щоразу на початку циклу, і якщо вона істинна, то виконується команда s += i; в тілі циклу. Зауваження щодо використання деяких арифметичних операцій

Операція	Аналог	Пояснення
i++	i=i+1	Збільшити i на одиницю
s += i	s=s+i	Збільшити s на величину i

Розв´яжемо задачу обчислення найбільшого спільного дільника з двох чисел. Цей алгоритм було вигадано Евклідом.

Опис алгоритму.

Якщо числа не одинакові, то від більшого віднімаємо менше, меншим замінюємо більше. Порівнюємо їх. Повторюємо до тих пір, поки числа не стануть рівними. Коли це трапиться, то це число й буде найбільшим спільним дільником.

Приклад. Використання циклів

```
1   using System;
2   namespace c for2
3   {
4   class Program
5   {
6   public static void Main(string[] args)
7   {
8   Console.WriteLine("Введіть значення a та b натискаючи після
кожного Enter");
9   int a = Convert.ToInt32(Console.ReadLine());
10  int b = Convert.ToInt32(Console.ReadLine());
11  for (int i = 1; a != b; i++)
12  {
13  if (a > b) a -= b;
14  else
15  if (a < b) b -= a;
```

```
16    }
17    Console.WriteLine("НСД={0}", a);
18    Console.ReadKey(true);
19    }
20    } 21 }
```

Зауваження щодо використання деяких арифметичних операцій.

Операція	Аналог	Пояснення
a -= b	a =a-b	Від значення змінної a відняти b, отриманий результат записати до змінної a
b -= a	b =b-a	Від значення змінної b відняти a, отриманий результат записати до змінної b

Доцільніше розв’язувати дану задачу за допомогою циклу while.

Загальний опис циклу while

while (expression) statement;

expression - умова продовження виконання циклу.

statement - команда, яка виконується в тілі циклу задану кількість разів.

Приклад. Використання циклу while

```
1     using System;
2     namespace c while1
3     {
4     class Program
5     {
6     public static void Main(string[] args)
7     {
8     Console.WriteLine("Введіть значення a та b натискаючи після
кожного Enter");
9     int a = Convert.ToInt32(Console.ReadLine());
10    int b = Convert.ToInt32(Console.ReadLine());
11    while (a != b)
12    {
13    if (a > b) a -= b;
```

```
14    else
15    if (a < b) b -= a;
16    }
17    Console.WriteLine("НСД={0}", a);
18    Console.ReadKey(true);
19    }
20    }
21    }
```

Загальний опис циклу do … while

```
do {
 statement
} while expression;
```

expression – умова продовження виконання циклу.

statement – команда, яка виконується в тілі циклу задану кількість разів.

Особливістю циклу do … while є обов´язковість виконання хоча б однієї ітерації, так як умова, яка може завершити цикл знаходиться в кінці самого циклу.

Розв´яжемо попередню задачу використовуючи цикл do … while.

Приклад. Використання циклу do…while

```
1     using System;
2     namespace c_do_while
3     {
4     class Program
5     {
6     public static void Main(string[] args)
7     {
8     Console.WriteLine("Введіть значення a та b натискаючи після кожного Enter");
9     int a = Convert.ToInt32(Console.ReadLine());
10    int b = Convert.ToInt32(Console.ReadLine());
```

```
11    do  {
12    if (a > b) a -= b;
13    else
14    if (a < b) b -= a;
15    } while(a != b);
16    Console.WriteLine("НСД={0}", a);
17    Console.ReadKey(true);
18    }
19    }
20    }
```

Цикл **foreach** дозволяє здійснювати ітерацію по кожному об´єкту в контейнерному класі. До контейнерних класів відносяться масиви, класи колекцій (**System.Collection**) та визначені користувачем класи колекції. Приклад цикла **foreach**:

```
int []  Ints = { 1, 2, 3 };  foreach (int temp in Ints)
{
 Console.WriteLine(temp);
}
```

Об´єкти циклу **foreach** доступні тільки для читання.

Розглянемо задачу, яка полягає в виводі на екран елементів масиву цілих чисел.

Приклад . Використання циклу foreach

```
1     using System;
2     namespace c_foreach
3     {
4     class Program
5     {
6     public static void Main(string[] args)
```

```
7    {
8    int[,] Array2 = { { 1, 4, 7 }, { 9, 1, 4 }};
9    foreach (int temp in Array2)
10   {
11   Console.WriteLine(temp);
12   }
13   Console.ReadKey(true);
14   }
15   }
16   }
```

Оператор goto дозволяє перейти до іншого рядка програми як вперед, так і назад. Для відмічання рядка, на який здійснюється перехід використовується мітка. Використання цього оператора разом із оператором if, також дозволяє створити цикли.

У прикладі, приведеному далі ілюструється використання оператора безумовного переходу goto.

Приклад . Використання циклу goto

```
1    using System;
2    namespace c_goto
3    {
4    class Program
5    {
6    public static void Main(string[] args)
7    {
8    Console.WriteLine("Цей рядок виконається!");
9    goto Label1;
10   Console.WriteLine("Цей рядок не виконається!");
11   Label1:
12   Console.WriteLine("Цей рядок виконається!");
13   Console.ReadKey(true);
14   }
15   }
```

```
16  }
```

У рядку 9 знаходиться оператор goto, який передає управляння рядку 12, який слідує за рядком 11, у якому стоїть мітка Label1 Рядок 10 виконуватися не буде.

Комбінуючи оператор goto та if можна створити цикл. Проте використанням оператора goto зловживати не слід, так як програма може стати дуже заплутаною і незрозумілою.

Є деякі обмеження на використання цього оператора. Його не можна, наприклад, використовувати для входу в цикл for.

Разом з циклами використовуються оператори **break** та **continue.**

Оператор **break** можна використовувати для виходу з циклу for, foreach, while, do while.

Оператор **continue** перериває лише поточну ітерацію, тобто виконання буде продовжене з наступної ітерації циклу, а не відбудеться виходу з циклу.

3.3.12. Функції

3.3.12.1. Функції без параметрів

Розглянуті нижче функції є нічим іншим як методами класу Program, визначеному на початку кожної програми на C#.

Виконаємо програму:

Приклад. Виклик функції без параметрів

```
1   using System;
2   namespace c_func
3   {
4   class Program
5   {
```

```
6     static void Write ()
7     {
8  Console.WriteLine("Текст виведений функцією ");
9     }
10    static void Main(string[] args)
11    {
12    Write () ;
13    Console.ReadKey();
14    }
15    }
16    }
```

В рядках 6-9 описується функція.

У рядку 6 зверніть увагу на слова **static** та **void.** Далі йде ім’я функції та порожня пара дужок.

Далі у фігурних дужках список операторів (у нашому випадку один), які виконуються в функції.

У рядку 12 здійснюється виклик функції. Опис функції (рядки 6-9) завжди повинен передувати виклику (рядок 12).

Можна помітити схожість структури рядків 6-9 та 10-14. У блоці 10-14 описано текст основної програми. Рядок 10 це можна назвати точною входу. Це тому, що з цього рядка починається виконання основної програми.

Як уже було вказано вище і Main (), і Write () визначаються з ключовими словами static та void.

Не будемо заглиблюватися у значення слова static, просто запам’ятаємо, що воно повинно бути обов’язковим у всіх функціях.

Ключове слово void означає те, що функція не повертає жодного значення.

Далі йде код, який викликає функцію. Він має такий вигляд:

Write (); //записується ім’я функції, далі вказуються порожні дужки.

Порожні дужки позначають те, що функція не потребує ніяких параметрів, які передавалися б їй з основної програми.

3.3.12.2. Функції з вихідним параметром

Найбільш простим способом для обміну даними з функцією є використання значення, що повертається. У коді функції відбуваються обчислення, після завершення результат цих обчислень повертається в основну програму для подальших обчислень.

Значення, що повертається має тип. Ім'я цього типу вказується при описі функції замість слова void.

Наприклад, можна створити функцію **GetString** (), яка повертає значення типу string описуватимемо цю функцію так:

```
static string GetString ()
    {
                        string s=Console.ReadLine();
            return s;   }
```

Використати дану функцію можна, наприклад, так:

```
Console.WriteLine("Уведіть слово");          string word=GetString();

Console.WriteLine("Довжина слова {0} {1} символів",word,word Length);
```

Розберемося з описом функції.

В заголовку функції після слова static указано тип значення, що повертається функцією string. Порожні дужки вказують на те, що функція не потребує вхідних параметрів. У коді функції після символу початку «{» оголошується рядкова змінна s, якій присвоюється результат виклику стандартної функції Console.**ReadLine**(). Рядок return s повертає обчислене значення рядка s в основну програму. Змінну s назвемо ***локальною***, так як вона описана у функції, і доступна лише з її коду.

Виклик функції з основної програми здійснюється командою string word=**GetString**(), яка присвоїть переданий функцією результат змінній word рядкового типу.

Декілька слів про оператор return. Це слово передує значенню, яке повертається функцією. У коді функції може бути й декілька таких операторів,

проте це можлива лише при використанні розгалужень, так як виконання цієї команди завершує функцію і повертає виконання в основну. Слово return можна використовувати й у функціях оголошених із словом void. У цьому випадку значення не повертатиметься, а дана команда просто завершить функцію. Функції з вхідними параметрами.

Якщо треба передати в функцію параметри, то їх потрібно оголосити в дужках після імені функції. При виклику функції в дужках слід записати значення, які передадуться в функцію.

Оголосимо функцію, яка отримавши два параметри типу double знайде з них більший і поверне результат у основну програму.

```
static double Max2(double x1, double x2)
    {
                                if (x1>x2) return x1; else return x2;
    }
```

У основній програмі здійснимо виклик даної функції, передавши в неї два числові параметри.

```
        Console.WriteLine("Уведіть два числа через пробіл");      string
myString=Console.ReadLine();
        string [] myWords;      myWords=myString.Split();    double
x1=Convert.ToDouble(myWords[0]);      double
x2=Convert.ToDouble(myWords[1]);
     Console.WriteLine("Найбільшим з двох є {0}",Max2(x1,x2));
```

У якості параметрів, що передаються у функцію може використовуватися список (масив) значень, кількість елементів якого не визначений.

Напишемо функцію для обчислення середнього арифметичного від уведеного списку чисел.

```
static double Average( double [] Arr)
    {
                    double sum=0;
                foreach (double t in Arr) sum+=t;          return sum/Arr
Length;
    }
```

Для перевірки роботи функції в основній програмі напишемо код.

```
            Console.WriteLine("Уведіть числа через пробіл");       string
myString=Console.ReadLine();
            string [] myWords;          myWords=myString.Split();
            double [] myDigits= new double[myWords.Length];       for (int i=0;
i<=myWords.Length-1;i++)
            {
                                        myDigits[i]=Convert.ToDouble(myWords[i]);
            }
        Console.WriteLine("Середнє арифметичне {0}",Average(myDigits));
```

Спочатку з клавіатури вводиться рядок чисел розділених пробілами. Далі за допомогою методу

Split() рядок конвертується в масив рядків myWords. За допомогою циклу for кожен елемент масиву myWords конвертується в масив чисел myDigits, який і передається як параметр до функції Average().

3.3.12.3. Масиви параметрів

У C# є можливість передавати всього один параметр для функції. Цей параметр назвемо ***масив параметрів***. Його задають останнім у виклику функції. Перед масивом параметрів указується ключове слова **params**.

Опис функції з масивом параметрів можна описати так:

```
        static int SumVals(params int[] vals)
            {
                    int sum = 0;                    foreach (int val in vals)
                    {
                                sum += val;
                    }
                            return sum;
            }
```

З основного коду програми викличемо функцію **SumVals**, передавши їй п’ять параметрів.

```
        int sum = SumVals(1, 5, 2, 9, 8) ;     Console.WriteLine("Сума = {0}", sum) ;
```

Зверніть увагу на те, що дану функцію можна було викликати як з одним, двома чи більше параметрів.

3.3.12.4 .Параметри змінні та параметри значення

У всіх прикладах досі ми використовувати параметри значення, які можна передати у функцію, проте якщо їх змінити у коді функції, то в основній програмі ці зміни ніяким чином не проявляться. Другими словами в функцію передасться копія значень змінних, самі ж змінні залишаться не модифікованими.

Розглянемо на прикладі

```
static void ShowDouble(int val)
    {
        val*=2;
                    Console.WriteLine("Подвоєне значення val={0}",val);
    }
```

Дана функція отримає з основної програми дані цілого типу, подвоїть їх та виведе отримане значення на консоль.

Викличемо дану функцію з основної програми виконавши код:

```
int myNumber=5;
Console.WriteLine("Значення до виклику функції {0}",myNumber);
ShowDouble(myNumber);
Console.WriteLine("Значення після виклику функції {0}",myNumber);
```

Результат виконання програми

```
Значення до виклику функції 5
Подвоєне значення val=10
Значення після виклику функції 5
```

Як бачимо, зміні, зроблені в функції не вплинули на значення змінної myNumber в основній програмі.

Що ж робити у випадку, коли потрібно в функції обчислити не одне, а більше значень?

У цьому випадку при передачі в функцію використовують не ***параметри значення***, а ***параметри змінні***. При цьому в функцію передадуться не копії

змінних, а посилання на області пам'яті, де зберігаються самі змінні. Тоді при модифікації параметрів у функції, самі змінні також модифікуються. Щоб вказати на те, що функція використовує параметри змінні, слід вказати при описі функції перед описом параметрів ключове слово **ref**.

Додамо слово **ref** у код функції та у код виклику функції.

```
static void ShowDouble(ref int val)
    {
                                        val*=2;Console.WriteLine("Подвоєне
значення val={0}",val);
    }
```

та

```
ShowDouble(ref myNumber);
```

Запустивши на виконання програму, отримаємо:

```
Значення до виклику функції 5
Подвоєне значення val=10
Значення після виклику функції 10
```

Цього разу виклик функції **ShowDouble** вплинув на значення **myNumber.**

При використанні параметрів-змінних слід враховувати, що:

1) в якості параметрів не можна використовувати константи (дані, оголошені з ключовим словом const);

2) змінна обов'язково повинна були ініційована (наприклад**, int n=4**; можна, **a int n**; - ні).

3.3.12.5. Вихідні параметри

Окрім використання параметрів змінних, для отримання даних з функцій можна використовувати ***вихідні параметри***, які помічаються ключовим словом **out**.

На відміну від параметра, відміченого ключем **ref**, параметри відмічені ключем out можна передавати в функцію без ініціалізації.

Створимо функцію для обчислення індекса максимального елемента масиву.

```
static int MaxValue(int [] intArray, out int maxIndex)
{
 int maxVal=intArray[0];
 maxIndex=0;
  for(int i=1;i<intArray Length;i++)
 {if(intArray[i]>maxVal)
                {
                                maxVal=intArray[i];
                                maxIndex=i;
                }
        }
 return maxVal;
}
```

Використати дану функцію можна так:

```
int [] myArray={1,8,3,6,2,5,9,3,0,2} ; int maxIndex;
                        Console.WriteLine("Максимальне значення myArray дорівнює
                                                {0}",MaxValue(myArray,out maxIndex));
Console.WriteLine("Перше входження цього значення зустрічається в
елементі {0}",maxIndex+1);
```

Після виконання отримаємо результат:

```
("Максимальне значення myArray дорівнює 9
Перше входження цього значення зустрічається в елементі 7
```

Зверніть увагу на використання ключових слів ref та out.

3.3.12.6. Область видимості змінних

При вивченні функції, напевне виникло питання про необхідності використання параметрів. Справа в тому, що для виключення великої кількості прикрих помилок, які до того ж досить важко знайти, кожна змінна видима лише в певній області коду. Наприклад, якщо змінна оголошена в деякій функції, то

отримати до неї доступ можна лише з цієї функції. У іншій функції можна оголосити змінну з такою самою назвою і вона буде іншою. Область коду, в якій оголошено деяку змінну назвемо областю видимості даної змінної .

Змінні чи константи, оголошені в класі Program назвемо ***глобальними***, а оголошені в функції або основній програмі Main ()— ***локальними.***

Глобальна змінна є видимою в усіх областях програми. Локальна змінна, оголошена в деякій функції є видимою лише в цій функції та лише у коді, записаному після її оголошення.

Приклад. Глобальні та локальні змінні

```
1   using System;
2   namespace c_func4
3   {
4   class Program
5   {
6   static string myString;
7   static void Write()
8   {
9   string myString="Локальна, визначена у Write ()";
10  Console.WriteLine("Виводимо у функції");
11  Console.WriteLine("Local myString={0}",myString);//Локальна
12  Console.WriteLine("Global myString={0}",Program.myString);

//Глобальна
13  }
14  public static void Main (string[] args)
15  {
16  string myString="Локальна, визначена у Main ()";
17  Program myString="Глобальна змінна ";
```

```
18    Write();
19    Console.WriteLine("\nВиводимо в програмі");
20    Console.WriteLine("Local myString={0}",myString);
21    Console.WriteLine("Global myString={0}",Program.myString);
22    Console.ReadKey();
23    }
24    }
25    }
```

Після запуску отримаємо в консолі:

```
Виводимо у функції
Local myString=Локальна у Write ()
Global myString=Глобальна змінна

Виводимо в програмі
Local myString=Локальна в Main ()
Global myString=Глобальна змінна
```

У рядку 6 оголошується глобальна змінна. У 17 рядку вона ініціюється перед викликом функції у рядку 18. До глобальної змінної можна отримати доступ вказавши як префікс ім'я класу, в якому вона оголошена: Program myString.

Змінна, оголошена в циклі також є локальною для цикла, і отримати до неї доступ можна лише з тіла циклу. Так при спробі виконати код:

```
        int i;
 for(i=0;i<10;i++)
        {
                                    string text="Рядок "+Convert.ToString(i);
                            Console.WriteLine("{0}",text);
        }
        Console.WriteLine("Останній текст, виведений в циклі:{0}",text);
```

Нам буде виведено повідомлення про помилку. Справа в тому, що змінна text оголошена та ініціалізована в циклі for, і вона не може бути використана поза цим блоком, так як є локальною для нього.

Якщо спробуємо виконати код:

```
        int i;   string text;
    for(i=0;i<10;i++)
        {
                                text="Рядок "+Convert.ToString(i);
                                Console.WriteLine("{0}",text);
        }
    Console.WriteLine("Останній текст, виведений в циклі:{0}",text);
```

результат також буже негативний, так як змінна text хоч і оголошена поза циклом, проте ініціалізована в циклі.

Якщо ж виконаємо код:

```
        int i;   string text="";
    for(i=0;i<10;i++)
        {
                                    text="Рядок "+Convert.ToString(i);
                                Console.WriteLine("{0}",text);
        }
    Console.WriteLine("Останній текст, виведений в циклі:{0}",text);
```

то програма успішно відкомпілюється й виведе на консоль:

```
Рядок 0
Рядок 1
…
Рядок 9
Останній текст, виведений в циклі:Рядок 9
```

3.3.12.7. Використання функцій у структурах

В структурах окрім полів можна використовувати й функції. На перший погляд незрозумілою є причина цього. Проте це може бути дуже корисним.

Розглянемо приклад. Опишемо структуру customerName, з полями-властивостями

```
firstName та lastName.  struct customerName
{     public string firstName,lastName;
}
```

В основній програмі опишемо масив.

customerName [] myCustomer=new customerName[5];

Якщо потрібно у вікно консолі вивести повне ім'я покупця, то у блоці опису структури опишемо функцію.

```
public string Name()
{
 return firstName+" "+lastName;
}
```

Приклад. Використання функцій у структурах

```
using System;
1     namespace c_struct
2     {
3     class Program
4     {
5     struct customerName
6     {
7     public string firstName,lastName;
8     public string Name()
9     {
10    return firstName+" "+lastName;
11    }
12    }
13    public static void Main (string[] args)
14    {
15    customerName [] myCustomer=new customerName[5];
16    myCustomer[0] firstName="Олександр";
17    myCustomer[0] lastName="Шевченко";
18    Console.WriteLine (myCustomer[0].Name());
 20                    }
 21          }
22 }
```

Пригадаймо, що у базових типах також є подібні внутрішні функції. Так для типу string визначені функції Trim (), Split () тощо.

3.3.13. Робота з файлами

Мова C# має функції для роботи з файлами. Як і інші комадни вводу-виводу вони розміщені в системному модулі System.IO, який треба підключити на початку програми командою.

```
using System.IO;
```

Доступними стають операції роботи з файлами та папками

Клас File. Створення, вилучення, копіювання та переміщення файла

Основні методи класа File

Create -	створеня файла
Exists -	перевірка існування файла
Delete -	вилучення файла
Move -	перейменування та переміщення файла
Copy -	копіювання файла

Приклад. Робота з файлами

```
using System; using System.IO; namespace contest
{
   class Test    {
      public static void Main()
      {
         // Створюємо файл
         File.Create("C:\\0.txt");          // Перевірка існування файла
if(File.Exists("C:\\1.txt"))
         {
            // Вилучення файла
            File.Delete("C:\\1.txt");
         }
         // Перейменування файла a.txt в b.txt
         File.Move("C:\\a.txt", "C:\\b.txt");          // Переміщення файла
         File.Move("C:\\c.txt", "D:\\c.txt");          // Копіювання файла
```

```
            File.Copy("D:\\z.txt", "D:\\x.txt");
        }
    }
}
```

Приклад. Читання з текстового файлу

```
1   using System;
2   using System.IO;
3   namespace c_file1
4   {
5   class Program
6   {
7   public static void Main(string[] args)
8   {
9   StreamReader sr = new StreamReader( "z:\\test.txt" );
10  while (sr.Peek()>-1) Console.WriteLine(sr.ReadLine());
11  sr.Close();
12  Console.ReadKey(true);
13  }
14  }
15  }
```

Передбачається існування файла "test.txt" на диску z:. Вміст цього файлу буде виведено в консоль. З кирилицею працює коректно при використанні кодування UTF-8.

Проаналізуємо код прикладу.

У рядку 9 створюється екземпляр класу StreamReader з іменем sr, який прив'язується до файлу з ім'ям z:\test.txt. Якщо вказати ім'я без повного шляху, то файл шукатиметься в папці, де буде розміщено виконуваний файл проекту.

У рядку 10 організовано цикл, який виконується до тих пір, поки метод **Peek()** повертатиме результат більший за -1. Ця умова виконуватиметься до тих пір, поки не досягнуто кінець файла.

Команда Console.**WriteLine**(sr.**ReadLine**()) зчитуватиме та виводитиме в консоль рядки файла. Як бачимо, функція для читання з файлу рядка, як і у випадку з клавіатурою має ім'я **ReadLine()**, відмінний лише батьківський клас (замість Console це ім'я нашого екземпляра sr для читання з файла.

Після завершення роботи з файлом його треба закрити. Це робиться методом **Close()**.

Приклад. Запис до текстового файлу

```
1   using System;
2   using System.IO;
3   namespace c_file2
4   {
5   class Program
6   {
7   public static void Main(string[] args)
8   {
9   StreamWriter  sw = new StreamWriter ( "z:\\test.txt" );
10  for (int i=0; i<=10; i++)
11  {
12  sw.WriteLine("Рядок {0}",i);
13  }
14  sw.Close();
15  Console.ReadKey(true);
16  }
17  }
18  }
```

Якщо файл z:\test.txt існує, то наша програма його вилучить перед створенням нового не запитавши в нас дозволу. Тому для збереження інформації

перед командою у рядку 9 бажано б було перевірити існування файлу методом Exists().

Створимо програму, яка прочитає з текстового файла три числа, уведені через пробіл та знайшовши середнє арифметичне, з точністю до сотих виведе результат до іншого текстового файлу

Приклад. Робота з текстовими файлами

```
1   using System;
2   using System.IO;
3   namespace c_file3
4   {
5   class Program
6   {
7   public static void Main(string[] args)
8   {
9   StreamReader  sr = new StreamReader ( "z:\\in.txt" );
10  StreamWriter  sw = new StreamWriter ( "z:\\out.txt" );
11  string s=sr.ReadLine();
12  string [] ss = s.Split();
13  double a=Convert.ToDouble(ss[0]);
14  double b=Convert.ToDouble(ss[1]);
15  double c=Convert.ToDouble(ss[2]);
16  double average=(a+b+c)/3;
17  sw.WriteLine ("{0:f2}",average);
18  sw.Close();
19  sr.Close();
20  }
21  }
22  }
```

Опис:

9 рядок – створення екземпляру sr класу StreamReader для читання з файлу.

10 рядок – створення екземпляру sw класу StreamWriter для запису до файлу.

11 рядок – читання до рядкової змінної s єдиного рядка з файлу.

12 рядок – роз’єднання рядка на слова та запис їх до масиву ss.

13-15 рядки – конвертація в double перших трьох елементів масиву ss та запис в змінні a,b,c.

16 рядок – обчислення середнього арифметичного.

17 рядок – форматований вивід середнього арифметичного до файлу.

18-19 рядки – закриття файлів.

3.4. Python

Основні властивості мови Python:

• гнучкий - підходить для розробки програм самого різного призначення і складності: від простих навчальних програм до професійних проектів.

• не буяє складними для освоєння конструкціями - на відміну від інших мов, про які ви могли чути (C, PHP, Pascal).

• не вимагає компіляції, т. Е. Програми на Python можна запускати на будь-яких платформах (Windows, Linux, MacOS, ...).

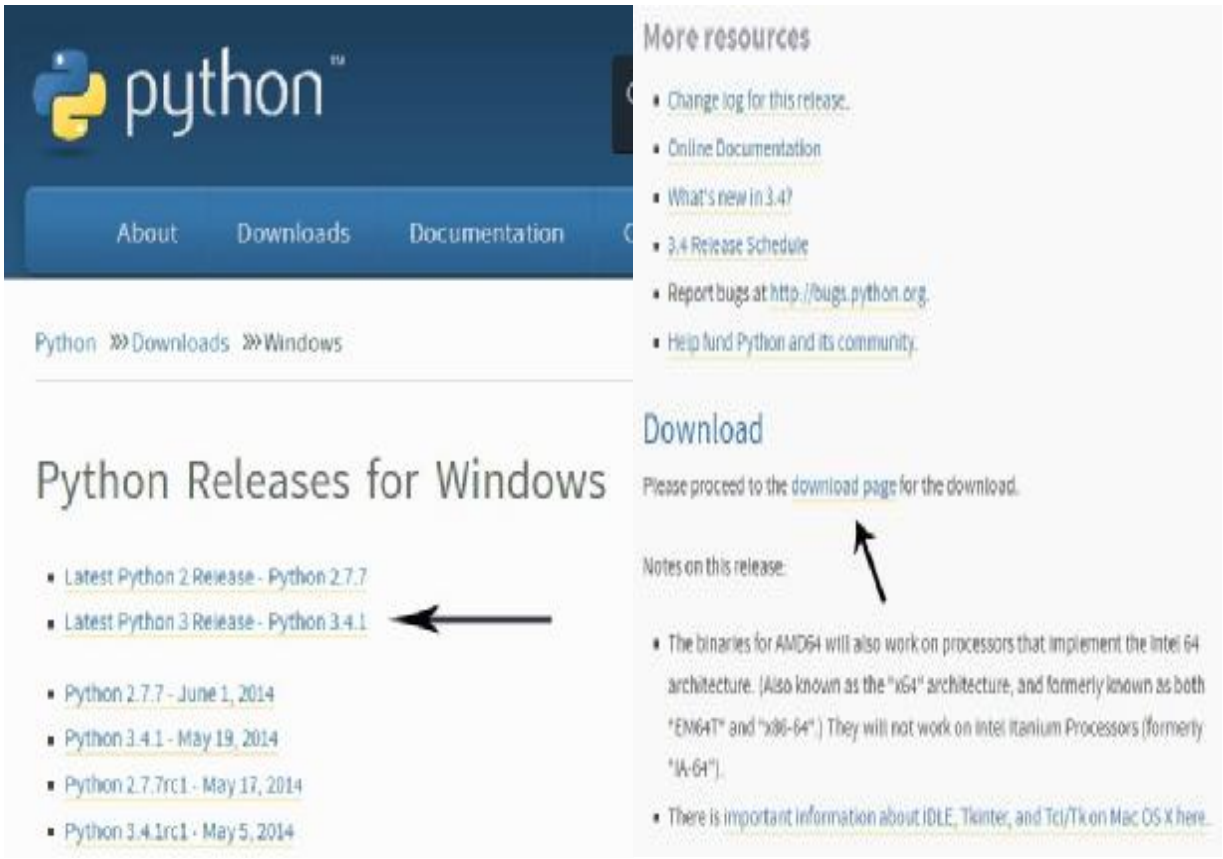
• підтримує безліч стилів програмування, на його прикладі можна освоїти будь-яку парадигму (ООП, структурне програмування, функціональне програмування). • містить ще безліч чудових можливостей.

Мова програмування Python використовується при розробці таких відомих інтернет-сервісів, як YouTube, Google, Yahoo !; Python застосовують при розробці програм в NASA (космічне агентство США) і CERN (Європейська організація з ядерних досліджень), а отже, добре підходить для обробки екномічних даних, отриманих з мережі Internet.

3.4.1. Установка Python на Windows

Завантажувати Python треба з офіційного сайту. Не рекомендується завантажувати інтерпретатор python з інших сайтів або через торрент, в них можуть бути віруси. Програма безкоштовна. Потрібно зайти на https://python.org/downloads/windows/, та обрати "latest python release" і взяти найсвіжішу версію python.

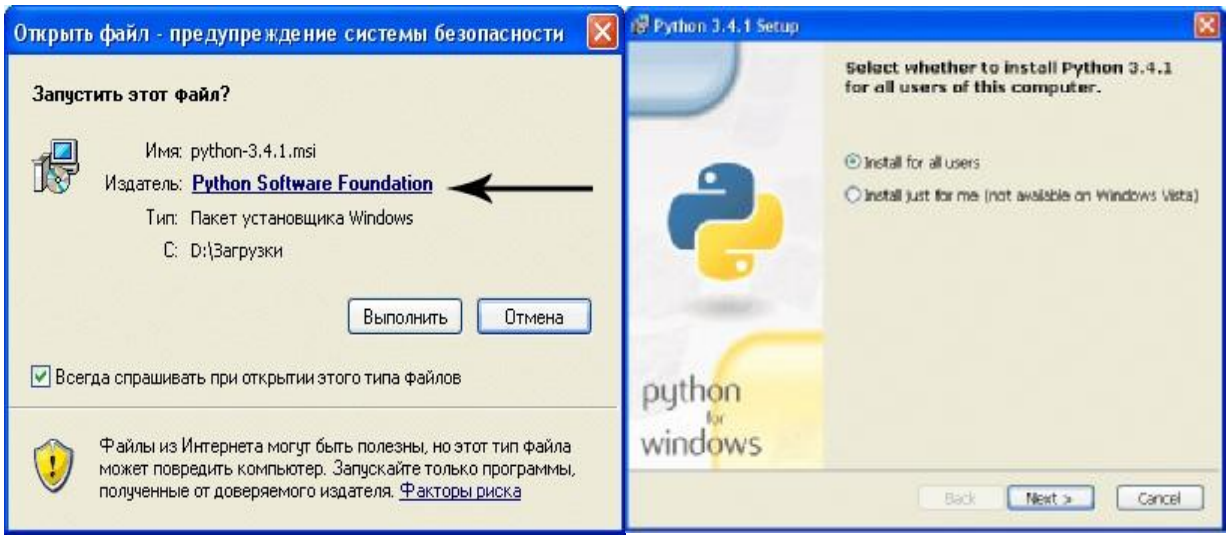
З'являється сторінка з описом даної версії Python (англійською). Потім крутимо в самий низ сторінки та відкриваємо "download page", як це показано на рисунках внизу.



Ви побачите список файлів, які можна завантажити. Нам потрібен Windows x86 MSI installer (якщо система 32-х бітна), або Windows x86-64 MSI installer (якщо система 64-х бітна). Більше з файлів нічого не потрібно.

Version	Operating System	Description	Date	MD5 Sum	File Size
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later		316a2f83edff73bbbcb2c84390bee2db	22776248
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later		534f8ec2f5ad5539f9165b3125b5e959	22692757
XZ compressed source tarball	Source release			6cafc183b4106476dd73d5738d7f616a	14125788
Gzipped source tarball	Source release			26695450087f8587b26d0b6a63844af5	19113124
Windows debug information files	Windows			9ce29e8356cf13f88e41f7595c2d7399	36744364
Windows x86 MSI installer	Windows			4940c3fad01ffa2ca7f9cc43a005b89a	24408064
Windows debug information files for 64-bit binaries	Windows			44a2d4d3c62a147f5a9f733b030490d1	24129218
Windows help file	Windows			6ff47ff938b15d2900f3c7311ab629e5	7297786
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors		25440653f27ee1597fd6b3e15eee155f	25104384

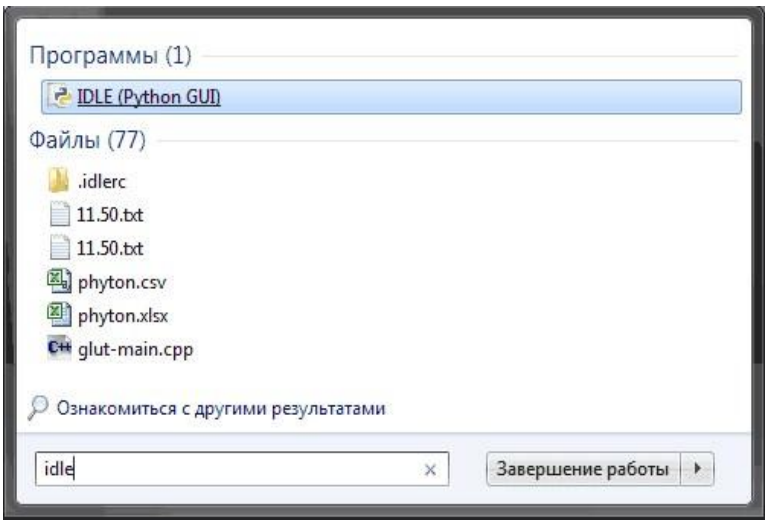
Чекаємо, поки python завантажиться. Потім відкриваємо завантажений файл. Якщо він підписаний Python Software Foundation, значить файл скачано вірно. Встановлюємо доступ для всіх користувачів або тільки для одного (на ваш розсуд).



Вибираємо папку для установки. Вибираємо компоненти, які будуть встановлені. Залиште компоненти за замовчуванням, якщо не впевнені.

3.4.2. Середовище розробки IDLE

Після завантаження та установки python відкриваємо IDLE (середовище розробки на мові Python, що поставляється разом з дистрибутивом).

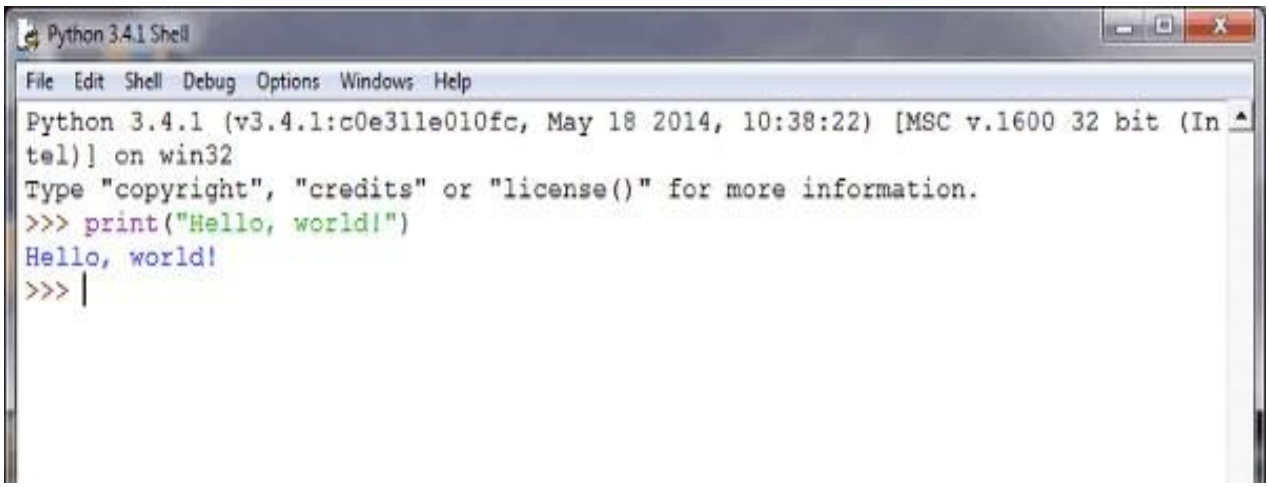


Запускаємо IDLE (спочатку запускається в інтерактивному режимі), після чого вже можна починати писати першу програму. Традиційно, першою програмою у нас буде "hello world".

Щоб написати "hello world" на python, достатньо всього одного рядка:

```
print ("Hello world!")
```

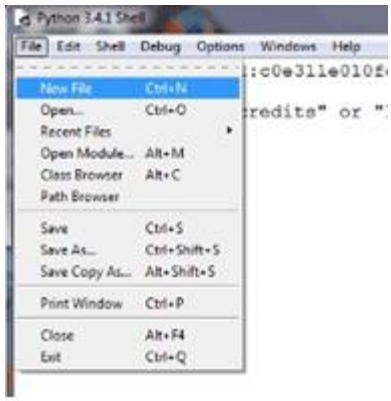
Вводимо цей код в IDLE і натискаємо Enter. Результат видно на зображенні нижче.



З інтерактивним режимом ми трохи познайомилися, можете з ним ще попрактикувася, наприклад, написати

print (3 + 4) print (3 * 5) print (3 ** 2)

Але, все-таки, інтерактивний режим не буде основним. В основному, ви будете зберігати програмний код в файл і запускати вже файл.



Для того, щоб створити нове вікно, в інтерактивному режимі IDLE виберіть File → New File (або натисніть Ctrl + N).

У вікні, введіть наступний код:

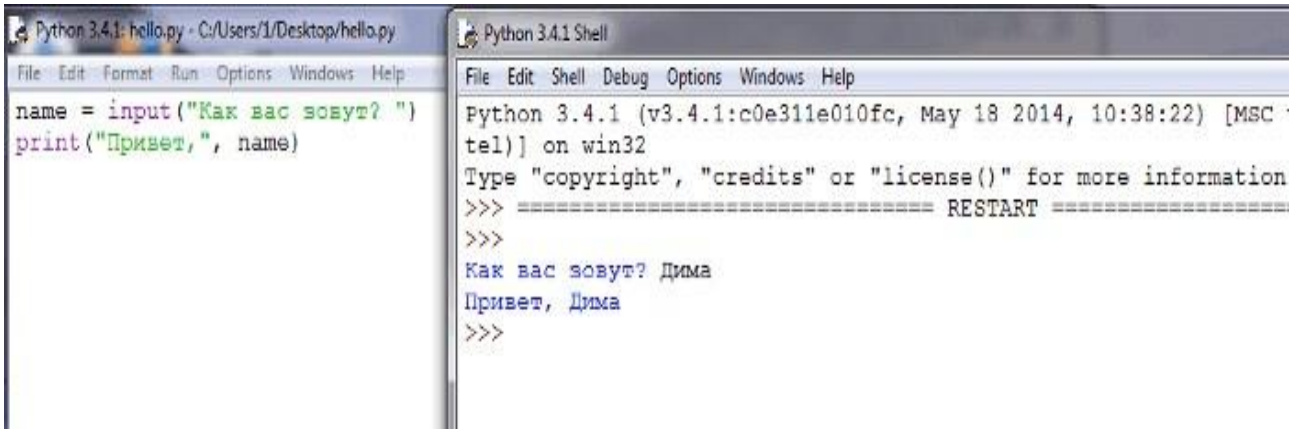
name = input ("Як Вас звати?") print ("Привіт,", name)

Перший рядок друкує питання ("Як Вас звати?"), Очікує, поки ви не надрукуєте що-небудь і не натиснете Enter і зберігає введене значення в змінній name.

У другому рядку ми використовуємо функцію print для виведення тексту на екран, в даному випадку для виведення "Привіт," і того, що зберігається в змінної "name".

Тепер натиснемо F5 (або виберемо в меню IDLE Run → Run Module і переконаємося, що те, що ми написали, працює. Перед запуском IDLE запропонує нам зберегти файл. Збережемо туди, куди вам буде зручно, після чого програма запуститься.

Ви повинні побачити щось на зразок цього (на скріншоті зліва – файл з написаної вами програмою, праворуч – результат її роботи).



3.4.3. Синтаксис

Кінець рядка є кінцем інструкції (крапка з комою не потрібно).

Вкладені інструкції об'єднуються в блоки по величині відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. І про зручність читання коду не забувайте. Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробіли (або знак табуляції).

Вкладені інструкції в Python записуються відповідно до одним і тим же шаблоном, коли основна інструкція завершується двокрапкою, слідом за яким розташовується вкладений блок коду, зазвичай з відступом під рядком основний інструкції.

Основна інструкція:
Вкладений блок інструкцій

Іноді можливо записати кілька інструкцій в одному рядку, розділяючи їх крапкою з комою:

a = 1; b = 2; print (a, b)

Але не робіть це занадто часто! Пам'ятайте про зручність читання коду.

Припустимо записувати одну інструкцію в декількох рядках. Досить її укласти в пару круглих, квадратних або фігурних дужок:

if (a == 1 and b == 2 and c == 3 and d == 4): # Не забуваємо про двокрапку print ('spam' * 3)

Тіло складовою інструкції може розташовуватися в тому ж рядку, що і тіло основний, якщо тіло є складовою інструкції і не містить складових інструкцій.. Наприклад:

if x> y: print (x)

3.4.4. Інструкція if-elif-else, перевірка істинності, тримісний вираз if / else

Умовна інструкція if-elif-else (її ще іноді називають оператором розгалуження) – основний інструмент вибору в Python. Простіше кажучи, вона вибирає, яку дію слід виконати, в залежності від значення змінних в момент перевірки умови.

Спочатку записується частина if з умовним виразом, далі можуть слідувати одна або більше необов'язкових частин elif, і, нарешті, необов'язкова частина else. Загальна форма запису умовної інструкції if виглядає наступним чином:

if test1: state1
elif test2:
state2 else:
state3

Простий приклад в якому на виході надрукується 'true', так як 1 – істина):

if 1:
print ('true')

```
else: print ( 'false') true
```

Трохи складніший приклад (його результат буде залежати від того, що ввів користувач):

```
a = int (input ()) if a <-5:
print ( 'Low') elif -5 <= a <= 5:
print ( 'Mid')
else:
print ( 'High')
```

Конструкція з декількома elif може також служити відмінною заміною конструкції switch - case в інших мовах програмування.

Взагалі в опеораторі if діє простий принцип:

Будь-яке число, не рівне 0, або непорожній об'єкт – істина.

Числа, рівні 0, порожні об'єкти і значення None – брехня.

Операції порівняння застосовуються до структур даних рекурсивно та повертають True або False

Логічні оператори and і or повертають істинний або помилковий об'єкт – операнд логічні оператори:

```
X and Y
```

Істина, якщо обидва значення X і Y істинні.

```
X or Y
```

Істина, якщо хоча б одне зі значень X або Y істинно.

```
not X
```

Істина, якщо X хибне.

Тримісний вираз if / else працює за наступною інструкцією:

```
if X:
A = Y
else:
A = Z
```

Хоч ця інструкці досить коротка, але, тим не менше, займає цілих 4 рядки. Спеціально для таких випадків і було придумано вираз if / else:

A = Y if X else Z

У даній інструкції інтерпретатор виконає вираз Y, якщо X істинно, в іншому випадку виконається вираз Z.

```
>>> A = 't' if 'spam' else 'f'
>>> A
' t '
```

3.4.5. Цикли for і while, оператори break і continue, команда else

While – один з найбільш універсальних циклів в Python, тому досить повільний. Виконує тіло циклу до тих пір, поки умова циклу істинно. Наприклад,

```
i = 5 while i <15:
print (i)
i = i + 2
5
7
9
11
13
```

Цикл for вже трішки складніший, трохи менше універсальний, але виконується набагато швидше циклу while. Цей цикл проходиться по будь-якому об'єкту, що піддається повтору, або як кажуть ітерації (рядку, списку, ...), і під час кожного проходу виконує тіло циклу.

```
for i in 'hello world': print (i * 2, end = '')
hheellllloo wwoorrlldd
```

Оператор continue починає наступний прохід циклу, минаючи тіло циклу, що залишилося (for або while)

```
>>> for i in 'hello world':
if i == 'o': continue
print (i * 2, end = '')
hheellll wwrrlldd
```

Оператор break достроково перериває цикл.

```
for i in 'hello world':
if i == 'o':
break
print (i * 2, end = '')
hheellll
```

Слово else, застосоване в циклі for або while, перевіряє, чи був проведений вихід з циклу інструкцією break, або ж "природним" чином. Блок інструкцій всередині else виконається тільки в тому випадку, якщо вихід з циклу стався без допомоги break.

```
for i in 'hello world':
if i == 'a':
break
else:
print ( 'Букви а в рядку немає')
Букви а в рядку немає
```

З прикладів добре видно, що результати роботи програми з’являються у тому ж вікні, що й сам текст програми.

3.4.6. Вбудовані функції Pyton

abs (x) – Повертає абсолютну величину (модуль числа).

all (послідовність) – Повертає True, якщо всі елементи справжні (або, якщо послідовність порожня).

any (послідовність) – Повертає True, якщо хоча б один елемент - істина. Для порожній послідовності повертає False.

ascii (object) – Як repr (), повертає рядок, що містить уявлення об'єкта, але замінює не ASCII символи на екрановані послідовності.

bin (x) – Перетворення цілого числа в двійкову рядок.

bool (x) – перетворення до типу bool, що використовує стандартну процедуру перевірки істинності. Якщо x є помилковим або опущений, повертає значення False, в іншому випадку вона повертає True.

bytearray ([джерело [, кодування [помилки]]]) – перетворення до bytearray. Bytearray - змінна послідовність цілих чисел в діапазоні 0 <= X <256. Викликана без аргументів, повертає порожній масив байт.

bytes ([джерело [, кодування [помилки]]]) – повертає об'єкт типу bytes, який є незмінною послідовністю цілих чисел в діапазоні 0 <= X <256. Аргументи конструктора інтерпретуються як для bytearray ().

callable (x) – Повертає True для об'єкта, що підтримує виклик (як функції).

chr (x) - Повертає односимвольних рядок, код символу якої дорівнює x.

classmethod (x) – Являє зазначену функцію методом класу.

compile (source, filename, mode, flags = 0, dont_inherit = False) – Компіляція в програмний код, який згодом може виконатися функцією eval або exec. Рядок не повинна містити символів повернення каретки або нульові байти. delattr (object, name) - Видаляє атрибут з ім'ям 'name'.

complex ([real [, imag]]) – перетворення до комплексного числа. dict ([object]) - перетворення до словника.

dir ([object]) – Список імен об'єкта, а якщо об'єкт не вказано, список імен в поточній локальній області видимості.

divmod (a, b) – Повертає приватне і залишок від ділення a на b.

enumerate (iterable, start = 0) – Повертає ітератор, при кожному проході надає кортеж з номера і відповідного члена послідовності.

eval (expression, globals = None, locals = None) – Виконує рядок програмного коду. exec (object [, globals [, locals]]) - Виконує програмний код на Python.

filter (function, iterable) – Повертає ітератор з тих елементів, для яких function повертає істину.

float ([X]) – перетворення до числа з плаваючою крапкою. Якщо аргумент не вказано, повертається 0.0.

format (value [, format_spec]) – Форматування (зазвичай форматування рядка). getattr (object, name, [default]) – витягує атрибут об'єкта або default.

frozenset ([послідовність]) – повертає незмінне безліч.

globals () – Словник глобальних імен.

hasattr (object, name) – Чи має об'єкт атрибут з ім'ям 'name'.

hash (x) – Повертає хеш зазначеного об'єкта.

help ([object]) – Виклик вбудованої довідкової системи. hex (x) - Перетворення цілого числа в шістнадцяткову рядок.

id (object) – Повертає "адреса" об'єкта. Це ціле число, яке гарантовано буде унікальним і постійним для даного об'єкта протягом терміну його існування.

input ([prompt]) – Повертає введену користувачем рядок.

int ([object], [підставу системи числення]) – перетворення до цілого числа.

isinstance (object, ClassInfo) – Істина, якщо об'єкт є екземпляром ClassInfo або його підкласом. Якщо об'єкт не є об'єктом даного типу, функція завжди повертає брехня.

issubclass (клас, ClassInfo) – Істина, якщо клас є підкласом ClassInfo. Клас вважається подклассом себе. iter (x) - Повертає об'єкт ітератора.

len (x) – Повертає число елементів в зазначеному об'єкті.

list ([object]) – створює список.

locals () – Словник локальних імен.

map (function, iterator) – Итератор, що вийшов після застосування до кожного елементу послідовності функції function.

max (iter, [args ...] * [, key]) – Максимальний елемент послідовності.

memoryview ([object]) – створює об'єкт memoryview.

min (iter, [args ...] * [, key]) – Мінімальний елемент послідовності.

next (x) – Повертає наступний елемент ітератора.

object () – повертає безликий об'єкт, який є базовим для всіх об'єктів. \

oct (x) – Перетворення цілого числа в вісімковий формат.

open (file, mode = 'r', buffering = None, encoding = None, errors = None, newline = None, closefd = True) – Відкриває файл і повертає відповідний потік.

ord (c) – Код символу.

pow (x, y, [r]) - (x ** y)% r. reversed (object) – Ітератор з розгорнутого об'єкта.

prompt – підказка користувачеві.

range ([start = 0], stop, [step = 1]) – арифметична прогресія від start до stop з кроком step.

repr (obj) – Представлення об'єкта. print ([object, ...], *, sep = "", end = '\ n', file = s

set ([object]) – створює меножину.

slice ([start = 0], stop, [step = 1]) – об'єкт зрізу від start до stop з кроком step.

str ([object], [кодування], [помилки]) – строкове представлення об'єкту. Використовує метод __str__.

tuple (obj) – перетворення до кортежу.

Нижче наведено перелік функцій, які діють на масиви:

abs, fabs – Обчислити абсолютне значення цілих, дійсних або комплексних елементів масиву. Для речових даних fabs працює швидше.

sqrt – Обчислити квадратний корінь з кожного елемента. Еквівалентно arr * * 0. 5

square– Обчислити квадрат кожного елемента. Еквівалентно arr ** 2

exp – Обчислити експоненту e кожного елемента.

log, logl0, log2, loglp – Натуральний (за основою *e*), десятковий, двійковий логарифм і функція log (1 + x) відповідно.

sign – Обчислити знак кожного елемента: 1 (для позитивних чисел), 0 (для нуля) або -1 (для негативних чисел).

ceil – Обчислити для кожного елемента найменше ціле число, що не менше його.

floor – Обчислити для кожного елемента найбільше ціле число, не більше його.

rint – Округлити елементи до найближчого цілого зі збереженням dtype.

modf – Повернути дробові і цілі частини масиву у вигляді окремих масивів

isnan – Повернути логічний масив, який показує, які значення є NaN (не числом).

isfinite, isinf – Повернути логічний масив, який показує, які елементи є кінцевими (НЕ inf і не NaN) або нескінченними відповідно.

cos, cosh, sin, tan, tanh sinh – Звичайні і гіперболічні тригонометричні функції.

arccos, arccosh, arcsinh, arctan, arcsin, arctanh – Зворотні тригонометричні функції.

logical_not – Обчислити значення істинності not x для кожного елемента. Еквівалентно -arr.

add – Скласти відповідні елементи масивів.

subtract – Відняти елементи другого масиву з відповідних елементів першого.

multiply – Перемножити відповідні елементи масивів.

divide, floor_divide – Ділення і ділення з відкиданням залишку.

Найбільш вживані функції з модуля numpy. linaig, який підключається до програми на Pyton, якщо вказати його назву:

diag – Повертає діагональні елементи квадратної матриці у вигляді одновимірного масиву або перетворює одновимірний масив в квадратну матрицю, в якій всі елементи, крім що знаходяться на головній діагоналі, дорівнюють нулю.

dot – Обчислює добуток матриць.

trace – Обчислює слід матриці – суму діагональних елементів.

det – Обчислює визначник матриці.

eig Обчислює власні значення і власні вектори квадратної матриці

inv – Обчислює зворотну матрицю

pinv – Обчислює псевдообернену матрицю Мура-Пенроуза для квадратної матриці.

qr – Обчислює QR-розкладання.

svd – Обчислює сингулярне розкладання (SVD).

solve – Вирішує лінійну систему Ax = B, де A – квадратная матриця.

lstsq – Обчислює рішення рівняння $y = xb$ за методом найменших квадратів.

3.4.7. Бібліотека turtle

Бібліотека turtle – це розширення мови Пітон, що дозволяє малювати на екрані нескладні малюнки.

Для цього потрібно підключити бібліотеку turtle. Програму, яка використовує цей графічний модуль треба починати командами (не забудьте про перші два рядки!):

```
import turtle turtle.reset ()
```

Для того, щоб затримати графічне вікно на екрані, необхідно закінчувати всі програми, які використовують модуль turtle командою:

turtle._root.mainloop ()

Треба відзначити, що на екрані ми побачимо трикутник.

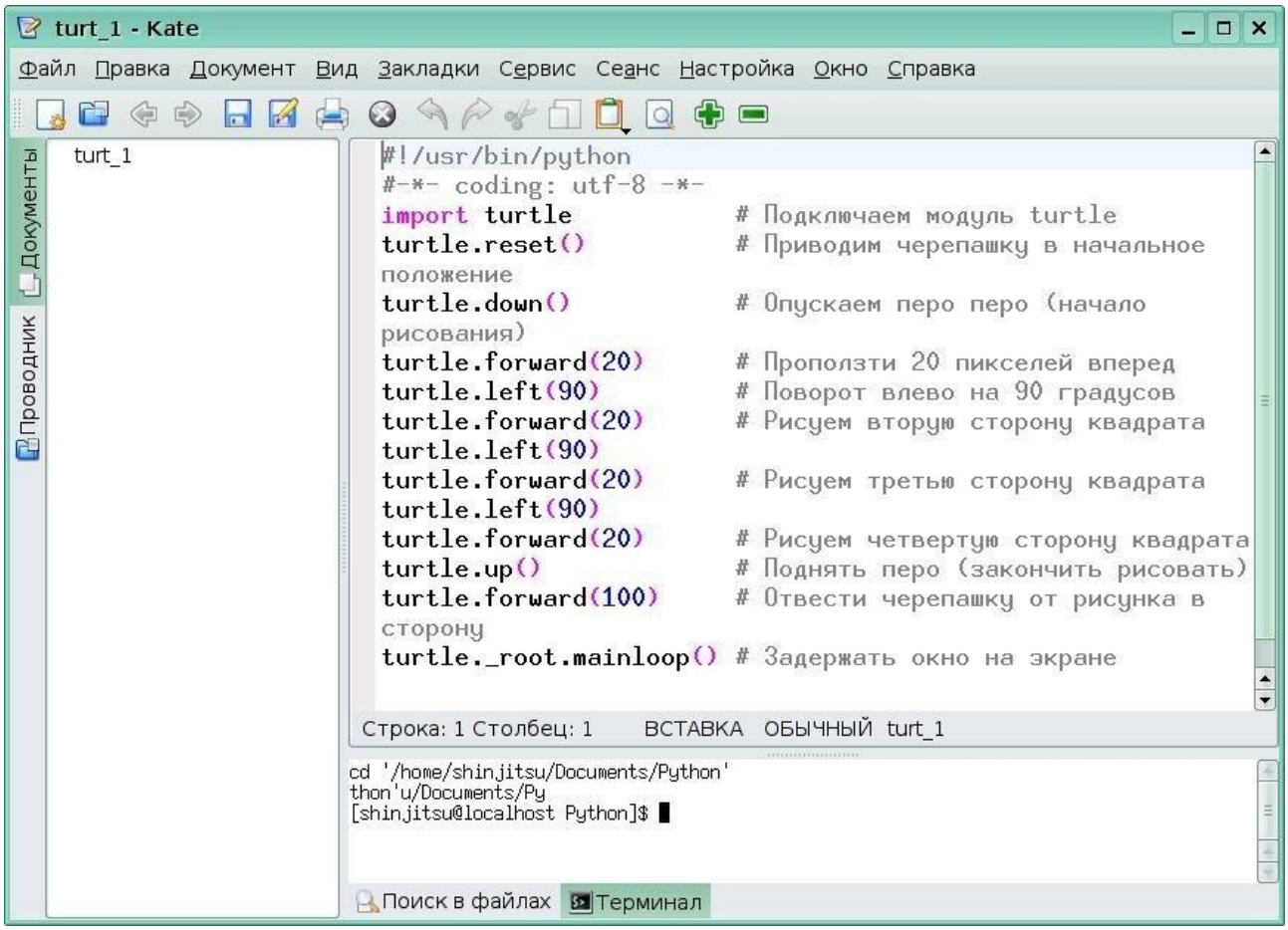
У Пітоні, як і в інших мовах програмування, необхідно використовувати коментарі. Вони теж позначаються знаком #, коментарі можна писати не тільки в окремому рядку, а й в тому ж рядку, правіше команди.

Пронумерувати рядки можна клавішею F11. Команди для малювання пишуться англійською мовою. Ось необхідний набір команд.

forward (a)	уперед на a кроків
backward (a)	назад на a кроків
left (ß)	ліворуч на ß градусів
right (ß)	праворуч на ß градісов
circle (r)	Намалювати окружність радіусом r, центр ліворуч - r>0, праворуч - r>0
circle (r,ß)	Намалювати дугу радіуса r та градусної міри ß
goto (x,y)	"перейти" в точку з координатами (x,y)
down ()	перо підніми
up ()	перо опусти
width (a)	нова ширина пера
color (s)	колір пера, "red" - червоний, "green" - зелений, "blue" - синій, "white" - білий, "black" - чорний, "yellow" - жовтий, "pink" - рожевий

fill (f)	Зафарбовує замкнуту область, перед роботою дайте команду *turtle.fill(1)*, а як закінчите розфарбування – *turtle.fill(0)*.
reset ()	сброс
clear ()	очистка екрану
write (s)	вивід тексту (=напиши)

На малюнку нижче показано приклад застосування цих команд.



3.4.8. Числа: цілі, речові, комплексні

Цілі числа мають тип (int).

Числа в Python 3 нічим не відрізняються від звичайних чисел. Вони підтримують набір звичайнісіньких математичних операцій:

x + y	Додавання
x - y	Віднімання
x * y	Множення
x / y	Ділення
x // y	Ціла частина від ділення
x % y	Решта від ділення
-x	Зміна знаку числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Зведення в степінь
pow(x, y[, z])	x^y по модулю (якщо модуль задано)

Також потрібно відзначити, що числа в python 3, на відміну від багатьох інших мов, підтримують довгу арифметику (проте, це вимагає більше пам'яті). Наприклад:

```
>>> 255 + 34
289
>>> 5 * 2
10
>>> 20/3
6.666666666666667
>>> 20 // 3
6
>>> 20% 3
2
>>> 3 ** 4
81
>>> pow (3, 4)
81
>>> pow (3, 4, 27)
0
>>> 3 ** 150
369988485035126972924700782451696644186473100389722973815184405301748249
```

Над цілими числами також можна виробляти бітові операції

x | y	Побітове або

x ^ y	Побітове або, що виключає
x & y	Побітове і
x << n	Бітовий зсув уліво
x >> y	Бітовий зсув управо
~x	Інверсія бітів

Функція int.bit_length () – кількість біт, необхідних для представлення числа в двійковому вигляді, без урахування знака і лідируючих нулів.

>>> n = -37 >>> bin (n)

'-0b100101'

>>> n.bit_length ()

6

Функція int.to_bytes (length, byteorder, *, signed = False) повертає рядок байтів, що представляють це число.

>>> (1024) .to_bytes (2, byteorder = 'big') b '\ x04 \ x00'
>>> (1024) .to_bytes (10, byteorder = 'big') b '\ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x04 \ x00'
>>> (-1024) .to_bytes (10, byteorder = 'big', signed = True) b '\ xff \ xff \ xff \ xff \ xff \ xff \ xff \ xff \ xfc \ x00'
>>> x = 1000
>>> x.to_bytes ((x.bit_length () // 8) + 1, byteorder = 'little') b '\ xe8 \ x03'
classmethod int.from_bytes (bytes, byteorder, *, signed = False) - повертає число з цього рядка байтів.
>>> int.from_bytes (b '\ x00 \ x10', byteorder = 'big') 16
>>> int.from_bytes (b '\ x00 \ x10', byteorder = 'little') 4096
>>> int.from_bytes (b '\ xfc \ x00', byteorder = 'big', signed = True)

Цілі числа мають позначення int, на приклад -1024.

>>> int.from_bytes (b '\ xfc \ x00', byteorder = 'big', signed = False) 64512

>>> int.from_bytes ([255, 0, 0], byteorder = 'big') 16711680

Системи числення у мові Pyton: двійкова, вісімкова, десятична та шістнадцяткова.Якщо потрібно переводити числа з однієї системи числення в іншу, то Python для цього надає кілька функцій:

int ([object], [підставу системи числення]) – перетворення до цілого числа в десятковій системі числення. За замовчуванням система числення десяткова, але можна задати будь-яку підставу від 2 до 36 включно.

bin (x) – перетворення цілого числа в двійкову рядок. hex (x) - перетворення цілого числа в шістнадцяткову рядок.

oct (x) – перетворення цілого числа в вісімкову рядок.

Приклади:

```
>>> a = int ('19 ') # Переводимо рядок в число
>>> b = int ('19 .5 ') # Рядок не є цілим числом Traceback (most recent call last): File "", line 1, in
ValueError: invalid literal for int () with base 10: '19 .5 '
>>> c = int (19.5) # Застосована до числа з плаваючою точкою, відсікає дробову частину >>> print (a, c)
19 19 >>> bin (19)
'0b10011'
>>> oct (19)
'0o23'
>>> hex (19)
'0x13'
>>> 0b10011 # Так теж можна записувати числові константи 19
>>> int ( '10011', 2)
19
>>> int ( '0b10011', 2)
19
```

Речові числа мають тип float. Речові числа підтримують ті ж операції, що і цілі. Однак (через представлення чисел в комп'ютері) речові числа неточні, і це може привести до помилок:

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.9999999999999999
```

Для високої точності використовують інші об'єкти (наприклад decimal і fraction).

Також речові числа не підтримують більшу розрядність, аніж завжди:

```
>>> a = 3 ** тисячі
>>> a + 0.1
Traceback (most recent call last): File "", line 1, in
OverflowError: int too large to convert to float
```

Простенькі приклади роботи з числами:

```
>>> c = 150
>>> d = 12.9
>>> c + d
162.9
>>> p = abs (d - c) # Модуль числа
>>> print (p)
137.1
>>> round (p) # Округлення 137
```

Додаткові методи роботи з числами представлені наступними функціями:

float.as_integer_ratio () – пара цілих чисел, чиє ставлення одно цього числа.

float.is_integer () – чи є значення цілим числом.

float.hex () – переводить float в hex (шестнадцатеричную систему числення).

classmethod float.fromhex (s) – перетворення float з шістнадцяткового числа.

```
>>> (10.5) .hex ()
'0x1.5000000000000p + 3'
>>> float.fromhex ( '0x1.5000000000000p + 3')
10.5
```

Крім стандартних виразів для роботи з числами (а в Python їх не так вже й багато), в складі Python є кілька корисних модулів.

Модуль math надає більш складні математичні функції.

```
>>> import math
>>> math.pi
```

3.141592653589793

>>> math.sqrt (85) 9.219544457292887

Модуль random реалізує генератор випадкових чисел і функції випадкового вибору.

```
>>> import random
>>> random.random ()
0.15651968855132303
```

3.4.9. Робота з рядками в Python. Літерали

Рядки в Python – впорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути представлено в текстовій формі.

Рядки можуть бути представлені в апострофах і в лапках

```
S = 'spam "s' S =" spam's "
```

Рядки в апострофа і в лапках – одне і те ж. Причина наявності двох варіантів в тому, щоб дозволити вставляти в літерали рядків символи лапок і апострофів, не використовуючи екранування.

Екрановані послідовності дозволяють вставити символи, які складно ввести з клавіатури.

Екранована послідовність	Призначення
\n	Перевод рядка
\a	Дзвінок
\b	Повернення курсора рядку на одну позицію ліворуч

\f	Перевод сторінки
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\N{id}	Ідентифікатор ID бази даних Юнікоду
\uhhhh	16-бітовий символ Юнікоду в 16-річному представленні
\Uhhhh. . .	32-бітовий символ Юнікоду в 32-річному представленні
\xhh	16-річне значення символу
\ooo	8- річне значення символу
\0	Символ Null (не є ознакою кінця рядку)

Якщо перед лапками, що відкривають рядок символів, стоїть символ 'r' (в будь-якому регістрі), то механізм подачі результатів роботи команнди на екран відключається.

S = r'C: \ newt.txt '

Але, незважаючи на призначення, "сирий" рядок не може закінчуватися символом зворотного слеша. Шляхи вирішення:

```
S = r '\ n \ n \\' [: - 1]
S = r '\ n \ n' + '\\'
S = '\\ n \\ n'
```

Головне достоїнство рядків в потрійних лапках в тому, що їх можна використовувати для запису багаторядкових блоків тексту. Усередині такого рядка можлива присутність лапок і апострофів, головне, щоб не було трьох лапок поспіль.

```
>>> c = " 'це дуже велика рядок, багатостроковий блок тексту' "
>>> c
'Це дуже великий \ nрядок, багатостроковий \ nблок тексту'
>>> print (c) це дуже великий рядок, багатостроковий блок тексту
```

Основні операції з рядками тексту наступні:

Конкатенація (додавання)

```
>>> S1 = 'spam'
```

```
>>> S2 = 'eggs'
>>> print (S1 + S2)
'Spameggs'
дублювання рядки
>>> print ( 'spam' * 3) spamspamspam
Довжина рядка (функція len)
>>> len ( 'spam') 4
Доступ за індексом
>>> S = 'spam'
>>> S [0]
'S' >>> S [2]
'A'
>>> S [-2]
'A'
```

Як видно з прикладу, в Python є можливість доступу по негативному індексу, при цьому відлік йде від кінця рядка.

З текстового рядка можні витягнути якусь його частину, або як кажкуть, «витягнути зріз рядка».

Оператор вилучення зрізу: [X: Y]. X – початок зрізу, а Y – закінчення; символ з номером Y в зріз не входить. За замовчуванням перший індекс дорівнює 0, а другий – довжині рядка.

```
>>> s = 'spameggs'
>>> s [3: 5]
'Me'
>>> s [2: -2]
'Ameg' >>> s [: 6]
'Spameg' >>> s [1:]
'Pameggs'
>>> s [:]
'Spameggs'
```

Крім того, можна задати крок, з яким потрібно витягувати зріз.

```
>>> s [:: - 1]
'Sggemaps'
>>> s [3: 5: -1]
''
>>> s [2 :: 2]
'Aeg'
```

При виклику методів необхідно пам'ятати, що рядки в Python відносяться до категорії незмінних послідовностей, тобто всі функції і методи можуть лише створювати новий рядок.

```
>>> s = 'spam'
>>> s [1] = 'b'
Traceback (most recent call last):
File "", line 1, in s [1] = 'b'
TypeError: 'str' object does not support item assignment
>>> s = s [0] + 'b' + s [2:]
>>> s
'Sbam'
```

Повний набір функцій та методів обробки рядків представлено у наступній таблиці

Функція або метод	Призначення
S = ‘str’; S = “str”; S = ‘’‘str’‘’; S = “””str”””	Літерали рядків
S = “s\np\ta\nbbb”	Екрановані послідовності
S = r”C:\temp\new”	Неформатованні рядки (пригнічують екранування)
S = b”byte”	Рядок байтів
S1 + S2	Конкатенація (складання ряків)
S1 * 3	Повтор рядків
S[i]	Звернення по індексу
S[i:j:step]	Витяг зрізу
len(S)	Довжина рядка
S.find(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого вхождення або 1
S.rfind(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього вхождення або 1
S.index(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого вхождення або викликає ValueError
S.rindex(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього вхождення або викликає ValueError
S.replace(шаблон, замена)	Заміна шаблону
S.split(символ)	Разбиття рядку по розділителю
S.isdigit()	Визначає, чи складається рядок із цифр
S.isalpha()	Визначає, чи складається рядок із букв

Функція або метод	Призначення
S.isalnum()	Визначає, чи складається рядок із цифр чи з букв
S.islower()	Визначає, чи складається рядок із символів в нижньому регістрі
S.isupper()	Визначає, чи складається рядок із символів в верхньому регістрі
S.isspace()	Визначає, чи складається рядок із символів, що невідображаються в тексті (прібел, символ переводу сторінки (‘\f’), “новий рядок” (‘\n’), “перевод каретки” (‘\r’), “горизонтальна табуляція” (‘\t’) и “вертикальна табуляція” (‘\v’))
S.istitle()	Визначає, чи починаються слова в рядку з заглавної букви
S.upper()	Перетворення строка до верхнього регістру
S.lower()	Перетворення строка до нижнього регістру
S.startswith(str)	Визначає, чи починаються рядок S з шаблону str
S.endswith(str)	Визначає, чи закінчується рядок S з шаблону str
S.join(список)	Збирання рядку зі списку з розділювачем S
ord(символ)	Перевод символу у код ASCII
chr(число)	Перевод коду ASCII у символ
S.capitalize()	Перевод першого символу рядка у верхній регистр, а всі іншиі – у нижній
S.center(width, [fill])	Повертає відцентрований рядок, по краям якого стоїть символ fill (пробіл за замовчуванням)
S.lstrip([chars])	Видаляє символи пробілів на початку рядка
S.rstrip([chars])	Видаляє символи пробілів у кінці рядка
S.strip([chars])	Видаляє символи пробілів на початку та у кінці рядка
S.swapcase()	Переводить символи нижнього региструк у верхній, а верхнього – у нижній
S.title()	Першу букву кожного слова переводить у верхній регистр, а всі інші – у нижній

Досить часто виникають ситуації, коли потрібно зробити рядок, підставивши в неї деякі дані, отримані в процесі виконання програми (призначений для користувача введення, дані з файлів і т.д.). Підстановку даних

можна зробити за допомогою форматування рядків. Форматування можна зробити за допомогою оператора%, і методу format.

Форматування рядків за допомогою методу format виконується, якщо для підстановки потрібно тільки один аргумент, то значення – сам аргумент:

```
>>> 'Hello, {}!'. Format ( 'Vasya')
'Hello, Vasya!'
```

А якщо кілька, то значеннями будуть усі аргументи з рядками підстановки (звичайних або іменованих):

```
>>> '{0}, {1}, {2}'. Format ( 'a', 'b', 'c')
'A, b, c'
>>> '{}, {}, {}'. Format ( 'a', 'b', 'c')
'A, b, c'
>>> '{2}, {1}, {0}'. Format ( 'a', 'b', 'c')
'C, b, a'
>>> '{2}, {1}, {0}'. Format (* 'abc')
'C, b, a'
>>> '{0} {1} {0}'. Format ( 'abra', 'cad')
'Abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'. Format (latitude = '37
.24N ', longitude =' - 115.81W ')
'Coordinates: 37.24N, -115.81W'
>>> coord = { 'latitude': '37 .24N ',' longitude ':' -115.81W '}
>>> 'Coordinates: {latitude}, {longitude}'. Format (** coord)
'Coordinates: 37.24N, -115.81W'
```

Однак метод format вміє більше. Ось його синтаксис:

поле заміни :: = "{" [ім'я поля] ["!" перетворення] [":" специфікація] "}"

ім'я поля :: = arg_name ("." ім'я атрибута | "[" індекс "]") *

перетворення :: = "r" (внутрішнє уявлення) | "S" (людське уявлення) специфікація :: = см. Нижче

наприклад:

```
>>> "Units destroyed: {players [0]}". Format (players = [1, 2, 3])
'Units destroyed: 1'
>>> "Units destroyed: {players [0]! R}". Format (players = [ '1',
'2', '3'])
"Units destroyed: '1'"
```

Тепер специфікація формату:

специфікація :: = [[fill] align] [sign] [#] [0] [width] [,] [. precision] [type]
заповнювач :: = символ крім '{' або '}' вирівнювання :: = "<" | ">" | "=" | "^"
знак :: = "+" | "-" | "" Ширина :: = integer точність :: = integer
тип :: = "b" | "C" | "D" | "E" | "E" | "F" | "F" | "G" | "G" |"N" | "O" | "S" | "X" | "X" | "%"

Вирівнювання проводиться за допомогою символу-заповнювача. Доступними параметрами вирівнювання є:

Флаг	Значення
‘<’	Символы, що заповнюють, будуть праворуч (вирівнювання об’єктк по лівому краю) (за замовчуванням).
‘>’	Вирівнювання об’єкту по правому краю.
‘=’	Заповновач буде після знаку, але перед цифрами. Працює тільки з числовыми типами.
‘^’	Вирівнювання по центру.

Опція “знак” используется только для чисел и может принимать следующие значения:

Флаг	Значення
‘+’	Знак має бути використаний для всіх чисел.
‘-‘	‘-‘ для негативних, нічого для для негативних,
‘Пробіл’	‘-‘для негативних, прібел для для негативних,

Поле “тип” може приймати наступні значення:

Тип	Значення
‘d’, ‘i’, ‘u’	Десяткове число.
‘o’	Число в восьмеричній системі счислення.
‘x’	Число в шестнадцятковій системі числення (букви у нижньому регістрі).
‘X’	Число в шестнадцятковій системі числення (букви у верхньому регістрі).
‘e’	Число з плаваючою точкою з експонентою (експонента у нижньому

	регістрі).
‘E’	Число з плаваючою точкою з експонентою (экспонента у верхньому регістрі).
‘f’, ‘F’	Число з плаваючою точкою (зивчайний формат).
‘g’	Число з плаваючою точкою з експонентою (експонента у нижньому регістрі), якщо воно меньше, за -4 або точність, інакше – звичайний формат.
‘G’	Число з плаваючою точкою з експонентою (экспонента у верхньому регістрі, якщо воно меньше, за -4 або точність, інакше – звичайний формат.
‘c’	Символ (рядок із из одного символу або число – код символу).
‘s’	Рядок.
‘%’	Число множиться на 100, відображається число з плаваючою точкою, а за ним знак %.

Наведемо декілька прикладів, для пояснення того, як використаовувати прийомии форматування тесту:

```
>>> coord = (3, 5)
>>> 'X: {0 [0]}; Y: {0 [1]} '. Format (coord)
'X: 3; Y: 5 '
>>> "repr () shows quotes: {! R}; str () does not: {! S}". Format
( 'test1', 'test2') "repr () shows quotes: 'test1'; str () does not:
test2 "
>>> '{: <30}'. Format ( 'left aligned')
'Left aligned'
>>> '{:> 30}'. Format ( 'right aligned')
'Right aligned'
>>> '{: ^ 30}'. Format ( 'centered')
'Centered'
>>> '{: * ^ 30}'. Format ( 'centered') # use '*' as a fill char
'********** centered **********'
>>> '{: + f}; {: + F} '. Format (3.14, -3.14) # show it always
'+3.140000; -3.140000 '>>>' {: f}; {: F} '. Format (3.14, -3.14)
# show a space for positive numbers
'3.140000; -3.140000 '>>>' {: -f}; {: -f} '. Format (3.14, -3.14) #
show only the minus - same as' {: f}; {: F} '
'3.140000; -3.140000 '
>>> # format also supports binary numbers
>>> "int: {0: d}; hex: {0: x}; oct: {0: o}; bin: {0: b}". Format
(42)
'Int: 42; hex: 2a; oct: 52; bin: 101010 '
```

```
>>> # with 0x, 0o, or 0b as prefix:
>>> "int: {0: d}; hex: {0: #x}; oct: {0: #o}; bin: {0: #b}". Format (42)
'Int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010 '
>>> points = 19.5
>>> total = 22

>>> 'Correct answers: {: .2%}'. Format (points / total)
'Correct answers: 88.64%'
```

3.4.10. Функції і методи списків

Списки в Python – це впорядковані змінювані колекції об'єктів довільних типів (майже як масив, але типи можуть відрізнятися).

Щоб використовувати списки, їх потрібно створити. Створити список можна декількома способами. Наприклад, можна обробити будь-який об'єкт (наприклад, рядок) вбудованою функцією list:

```
>>> list ( 'список')
['перелік']
```

Список можна створити і за допомогою літералу:

```
>>> s = [] # Порожній список
>>> l = [ 's', 'p', [ 'isok'], 2]
>>> s
[]
>>> l
[ 'S', 'p', [ 'isok'], 2]
```

Як видно з прикладу, список може містити будь-яку кількість будь-яких об'єктів (в тому числі і вкладені списки), чи не містити нічого.

І ще один спосіб створити список – це генератори списків. Генератор списків – це спосіб побудувати новий список, застосовуючи вираз до кожного елементу послідовності. Генератори списків дуже схожі на цикл for.

```
>>> c = [c * 3 for c in 'list'] >>> c
```

['Lll', 'iii', 'sss', 'ttt']

Можлива і більш складна конструкція генератора списків:

```
>>> c = [c * 3 for c in 'list' if c! = 'I']
>>> c
[ 'Lll', 'sss', 'ttt']
>>> c = [c + d for c in 'list' if c! = 'I' for d in 'spam' if d! = 'A'] >>> c
[ 'Ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

Але в складних випадках краще користуватися звичайним циклом for для генерації списків.

Для списків доступні основні вбудовані функції, а також методи списків:

Метод	Що робить
list.append(x)	Додає елемент у кінець списку
list.extend(L)	Разширяє список list, додаючи в кінець всі елементи списку L
list.insert(i, x)	Вставляє на *i*-ий элемент значеннння *x*
list.remove(x)	Видаляє перший елемент у списку, що має значення *x*
list.pop([i])	Видаляє *i*-ий елемент і повертає його. Якщо індекс не вказано, видаляється останній елемент
list.index(x, [start [, end]])	Повертає положення першого елементу від start до end зі значением *x*
list.count(x)	Повертає кількість елементів зі значенням *x*
list.sort([key = функция])	Сортує список на основі функції
list.reverse()	Рогортає список
list.copy()	Поверхнева копія списку
list.clear()	Очищає список

Потрібно відзначити, що методи списків, на відміну від строкових методів, змінюють сам список, а тому результат виконання не потрібно записувати в цю змінну. Наприклад:

```
>>> l = [1, 2, 3, 5, 7]
>>> l.sort ()
>>> l
[1, 2, 3, 5, 7]
>>> l = l.sort ()
```

```
>>> print (l) None
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print (a.count (333), a.count (66.25), a.count ('x'))
2 1 0
>>> a.insert (2, -1)
>>> a.append (333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index (333)
1
>>> a.remove (333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse ()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort ()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

3.4.11. Індекси і зрізи

Як і в інших мовах програмування, взяття за індексом означає, вказати номер рядка та стовпця у масиві:

```
>>> a = [1, 3, 8, 7]
>>> a [0]
1
>>> a [3]
7
>>> a [4]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Як і в багатьох інших мовах, нумерація елементів починається з нуля. При спробі доступу до неіснуючого індексу виникає виняток IndexError.

В даному прикладі змінна a була списком, однак взяти елемент за індексом можна і у інших типів: рядків, кортежів.

В Python також підтримуються негативні індекси, при цьому нумерація йде з кінця, наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a [-1]
7
>>> a [-4]
1
>>> a [-5]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

В Python, крім індексів, існують ще й зрізи:

item [START: STOP: STEP] – бере зріз від номера START, до STOP (не включаючи його), з кроком STEP. За замовчуванням START = 0, STOP = довжині об'єкта, STEP = 1. Відповідно, якісь (а можливо, і все) параметри можуть бути опущені.

```
>>> a = [1, 3, 8, 7]
>>> a [:]
[1, 3, 8, 7]
>>> a [1:] [3, 8, 7] >>> a [: 3] [1, 3, 8]
>>> a [:: 2]
[1, 8]
```

Також всі ці параметри можуть бути і негативними:

```
>>> a = [1, 3, 8, 7]
>>> a [:: - 1]
[7, 8, 3, 1]
>>> a [: - 2]
```

```
[1, 3]
>>> a [-2 :: - 1] [8, 3, 1]
>>> a [1: 4: -1] []
```

В останньому прикладі вийшов порожній список, так як START <STOP, а STEP негативний. Те ж саме відбудеться, якщо діапазон значень виявиться за межами об'єкта:

```
>>> a = [1, 3, 8, 7]
>>> a [10:20] []
```

Також за допомогою зрізів можна не тільки отримувати елементи, але і додавати і видаляти елементи (зрозуміло, тільки для змінних послідовностей).

```
>>> a = [1, 3, 8, 7]
>>> a [1: 3] = [0, 0, 0]
>>> a
[1, 0, 0, 0, 7]
>>> del a [: - 3]
>>> a
[0, 0, 7]
```

3.4.12. Кортежі (tuple)

Кортеж, по суті – це незмінний список.

Навіщо потрібні кортежі, якщо є списки? Ось приклади з поясненнями:

1. Захист від дурня. Тобто кортеж захищений від змін, як навмисних (що погано), так і випадкових (що добре).

2. Менший розмір:

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a .__ sizeof __ ()
36
>>> b .__ sizeof __ ()
```

3. Можливість використовувати кортежі в якості ключів словника:

```
>>> d = {(1, 1, 1): 1}
>>> d
{(1, 1, 1): 1}
>>> d = {[1, 1, 1]: 1}
Traceback (most recent call last):
File "", line 1, in d = {[1, 1, 1]: 1}
TypeError: unhashable type: 'list'
```

З кортежами працювати приблизно так само, як і зі списками.

Створюємо порожній кортеж:

```
>>> a = tuple () # За допомогою вбудованої функції tuple ()
>>> a
()
>>> a = () # За допомогою литерала кортежу
>>> a
()
>>>
```

Створюємо кортеж з одного елемента:

```
>>> a = ( 's')

>>> a 's'
```

Але в цьому випадку вийшов рядок. Як отримати кортеж? Наступним чином:

```
>>> a = ( 's',)
>>> a
( 'S',)
```

Вся справа - у комі. Самі по собі дужки нічого не значать, точніше, означають те, що всередині них знаходиться одна інструкція, яка може бути відділена пробілами, перенесенням рядків та ін. Кортеж можна створити і так:

```
>>> a = 's',
>>> a
( 'S',)
```

Але все-таки не захоплюйтеся, і ставте дужки, тим більше, що бувають випадки, коли дужки необхідні.

Ну і створити кортеж з об'єкту можна за допомогою все тієї ж функції tuple ()

```
>>> a = tuple ( 'hello, world!')
>>> a
( 'H', 'e', 'l', 'l', 'o', ',', '', 'w', 'o', 'r', 'l', 'd', '! ')
```

Всі операції над списками, що не змінюють список (додавання, множення на число, методи index () і count () і деякі інші операції). Можна також по-різному змінювати елементи місцями і так далі.

Наприклад, гордість програмістів на Python – поміняти місцями значення двох змінних:

```
a, b = b, a
```

3.4.13. Словники (dict) і робота з ними. методи словників

Словники в Python – це невпорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями.

Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами. Поперше, за допомогою литерала:

```
>>> d = {}
>>> d
{}
>>> d = { 'dict': 1, 'dictionary': 2}
>>> d
{ 'Dict': 1, 'dictionary': 2}
```

По-друге, за допомогою функції dict:

```
>>> d = dict (short = 'dict', long = 'dictionary') >>> d
{ 'Short': 'dict', 'long': 'dictionary'}
>>> d = dict ([(1, 1), (2, 4)])
>>> d
```

```
{1: 1, 2: 4}
```

По-третє, за допомогою методу fromkeys:

```
>>> d = dict.fromkeys ([ 'a', 'b'])
>>> d
{ 'A': None, 'b': None}
>>> d = dict.fromkeys ([ 'a', 'b'], 100)
>>> d
{ 'A': 100, 'b': 100}
```

По-четверте, за допомогою генераторів словників, які дуже схожі на генератори списків.

```
>>> d = {a: a ** 2 for a in range (7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Тепер спробуємо додати записів в словник і витягти значення ключів:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d [1]
2
>>> d [4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d [ '1']
Traceback (most recent call last):
File "", line 1, in d [ '1']
KeyError: '1'
```

Як видно з прикладу, привласнення по новому ключу розширює словник, привласнення за існуючим ключу перезаписує його, а спроба вилучення неіснуючого ключа породжує виключення. Для уникнення виключення є спеціальний метод (див. Нижче), або можна перехоплювати виняток.

Що ж можна ще робити зі словниками? Так то ж саме, що і з іншими об'єктами: вбудовані функції, ключові слова (наприклад, цикли for і while), а також спеціальні методи словників.

Методи словників описані настиупними функціями:

dict.clear () – очищає словник.

dict.copy () – повертає копію словника.

classmethod dict.fromkeys (seq [, value]) – створює словник з ключами з seq і значенням value (за замовчуванням None).

dict.get (key [, default]) – повертає значення ключа, але якщо його немає, не кидає виняток, а повертає default (за замовчуванням None).

dict.items () – повертає пари (ключ, значення).

dict.keys () – повертає ключі в словнику.

dict.pop (key [, default]) – видаляє ключ і повертає значення. Якщо ключа немає, повертає default (за замовчуванням кидає виняток).

dict.popitem () – видаляє і повертає пару (ключ, значення). Якщо словник порожній, кидає виняток KeyError. Пам'ятайте, що словники невпорядковані.

dict.setdefault (key [, default]) – повертає значення ключа, але якщо його ні, не кидає виняток, а створює ключ з значенням default (за замовчуванням None).

dict.update ([other]) – оновлює словник, додаючи пари (ключ, значення) з other. Існуючі ключі перезаписувати. Повертає None (не нова словник!).

dict.values () – повертає значення в словнику.

3.4.14. Множини (set і frozenset)

Множина в Python – "контейнер", що містить елементи, які не повторюються і розташовані у випадковому порядку.

Створюємо множини:

```
>>> a = set () >>> a set ()
>>> a = set ( 'hello') >>> a
{ 'H', 'o', 'l', 'e'}
>>> a = { 'a', 'b', 'c', 'd'}
>>> a
{ 'B', 'c', 'a', 'd'}
```

```
>>> a = {i ** 2 for i in range (10)} # генератор множин
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {} # А так не можна!
>>> type (a)
<Class 'dict'>
```

Як видно з прикладу, множдини мають той же літерал, що і словники, але порожню множина за допомогою літералу створити не можна.

Множини зручно використовувати для видалення повторюваних елементів:

```
>>> words = [ 'hello', 'daddy', 'hello', 'mum']
>>> set (words)
{ 'Hello', 'daddy', 'mum'}
```

З множинами можна виконувати безліч операцій: знаходити об'єднання, перетин, тощо. Ось перелік осмновних функцій, що обробляють множини:

len (s) – число елементів у множині (розмір множини).

x in s – перевірка, чи належить *x* множині *s*.

set.isdisjoint (other) – істина, якщо set і other не мають спільних елементів. Якщо set = other – всі елементи set належать other, інакше, всі елементи other належать set.

set.issubset (other) або set <= other – всі елементи set належать other.

set.issuperset (other) або set> = other – аналогічно. set.union (other, ...) або set | other | ... - об'єднання декількох множин. set.intersection (other, ...) або set & other & ... – перетин.

set.difference (other, ...) або set – other – ... - множина усіх елементів set, які не належать жодному з other.

set.symmetric_difference (other); set ^ other – множина з елементів, що зустрічаються в одній множин, але не зустрічаються в обох.

set.copy () – копія множини.

set.update (other, ...); set | = other | ... – об'єднання.

set.intersection_update (other, ...); set & = other & ... – перетин.

set.difference_update (other, ...); set - = other | ... – віднімання.

set.symmetric_difference_update (other); set ^ = other set.add (elem) – додавання елемента в множину.

set.remove (elem) – видаляє елемент із множини або видає. KeyError, якщо такого елемента не існує.

set.discard (elem) – видаляє елемент, якщо він знаходиться у множині.

set.pop () – видаляє перший елемент з множини. Так як множини не впорядковані, не можна точно сказати, який елемент буде першим.

set.clear () – очищення множини.

Єдина відмінність set від frozenset полягає в тому, що set – змінюваний тип даних, а frozenset – ні. Приблизно схожа ситуація зі списками і кортежами.

```
>>> a = set ( 'qwerty')
>>> b = frozenset ( 'qwerty')
>>> a == b
True
>>> type (a - b)
>>> type (a | b)
>>> a.add (1)
  >>> b.add (1)
Traceback (most recent call last):
File "", line 1, in
b.add (1)
AttributeError: 'frozenset' object has no attribute 'add'
```

3.4.15. Функції та їх аргументи

Функція в Python – це об'єкт, який приймає аргументи і повертає значення Зазвичай функція визначається за допомогою інструкції def.

Визначимо найпростішу функцію:

```
def add (x, y):
return x + y
```

Інструкція return каже, що потрібно повернути значення В нашому випадку функція повертає суму x і y.

Тепер ми її можемо викликати:

```
>>> add (1, 10)
11
>>> add ( 'abc', 'def')
'Abcdef'
```

Функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, і навіть функції!):

```
>>> def newfunc (n):
def myfunc (x): return x + n return myfunc
>>> new = newfunc (100)
>>> new (200)
300
>>> # new - це функція
```

Функція може і не закінчуватися інструкцією return, при цьому функція поверне значення None:

```
>>> def func (): pass
>>> print (func ())
None
>>>
```

Функція може приймати будь-яку кількість аргументів чи не приймати їх зовсім. Також поширені функції з довільним числом аргументів, функції з позиційними і іменованими аргументами, обов'язковими і необов'язковими.

```
>>> def func (a, b, c = 2): # c - необов'язковий аргумент return a
+ b + c
>>> func (1, 2) # a = 1, b = 2, c = 2 (за замовчуванням)
5
>>> func (1, 2, 3) # a = 1, b = 2, c = 3
6
>>> func (a = 1, b = 3) # a = 1, b = 3, c = 2
6
>>> func (a = 3, c = 6) # a = 3, c = 6, b не визначений Traceback
(most recent call last):
File "", line 1, in func (a = 3, c = 6)
```

TypeError: func () takes at least 2 arguments (2 given)

Функція також може приймати змінна кількість позиційних аргументів, тоді перед ім'ям ставиться *:

```
>>> def func (* args): return args
>>> func (1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func ()
()
>>> func (1) (1,)
```

Як видно з прикладу, args – це кортеж з усіх переданих аргументів функції, і зі змінною можна працювати так само, як і з кортежем.

Функція може приймати і довільне число іменованих аргументів, тоді перед ім'ям ставиться **:

Приклад аргументів функції:

```
>>> def func (** kwargs): return kwargs
>>> func (a = 1, b = 2, c = 3)
{ 'A': 1, 'c': 3, 'b': 2}
>>> func ()
{}
>>> func (a = 'python')
{ 'A': 'python'} >>>
```

В змінної kwargs у нас зберігається словник, з яким ми, знову-таки, можемо робити все, що нам заманеться.

Анонімні функції можуть містити лише один вислів, але виконуються вони швидше. Анонімні функції створюються за допомогою інструкції lambda. Крім цього, їх не обов'язково привласнювати змінній, як робили ми інструкцією def func ():

```
>>> func = lambda x, y: x + y
>>> func (1, 2) 3
>>> func ( 'a', 'b')
'Ab'
>>> (lambda x, y: x + y) (1, 2)
```

```
3
>>> (lambda x, y: x + y) ( 'a', 'b')
'Ab' >>>
```

lambda функції, на відміну від звичайної, не потрібно інструкція return, а в іншому, поводиться точно так же:

```
>>> func = lambda * args: args
>>> func (1, 2, 3, 4)
(1, 2, 3, 4) >>>
```

3.4.16. Файли. Робота з файлами

Перш, ніж працювати з файлом, його треба відкрити. З цим чудово впорається вбудована функція open:

```
f = open ('text.txt', 'r')
```

У функції open багато параметрів, нам поки важливі три аргументи: перший, це ім'я файлу. Шлях до файлу може бути відносним або абсолютним. Другий аргумент, це режим, в якому ми будемо відкривати файл.

Режим	Позначення
‘r’	Відкривання для читання (є значенням за замовчуванням).
‘w’	Відкривання для запису, зміст файлу видаляється, якщо файлу не існує, тоді створюється новий.
‘x’	Відкривання для запису, якщо файлу не існує, інакше – виключення.
‘a’	Відкривання для дозапису, інформація додається у кінець файлу.
‘b’	Відкривання для перегляду у двоїчному режимі.
‘t’	Відкривання для перегляду в текстовому режимі (є значенням за замовчуванням).
‘+’	Відкривання для читання й запису

Режими можуть бути об'єднані, тобто, наприклад, 'rb' – читання в двійковому режимі. За замовчуванням режим дорівнює 'rt'.

І останній аргумент, encoding, потрібен тільки в текстовому режимі читання файлу. Цей аргумент задає кодування.

Відкрили ми файл, а тепер ми хочемо прочитати з нього інформацію. Для цього є кілька способів, але великого інтересу заслуговують лише два з них.

Перший – метод read, читає весь файл цілком, якщо був викликаний без аргументів, і n символів, якщо був викликаний з аргументом (цілим числом n).

```
>>> f = open ( 'text.txt')
>>> f.read (1)
'H'
>>> f.read ()
'Ello world! \ NThe end. \ N \ n'
```

Ще один спосіб зробити це - прочитати файл через підрядник, скориставшись циклом for:

```
>>> f = open ( 'text.txt') >>> for line in f:
line
'Hello world! \ N'
'\ N'
'The end. \ N'
'\ N'
```

Тепер розглянемо запис в файл. Спробуємо записати в файл ось такий список:

```
>>> l = [str (i) + str (i-1) for i in range (20)]
>>> l
[ '0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110' ,
'1211', '1312', '1413
',
```

Відкриємо файл на запис:

```
>>> f = open ( 'text.txt', 'w')
```

Запис в файл здійснюється за допомогою методу write:

```
>>> for index in l:
f.write (index + '\ n')
4
```

```
3
3
3
3
```

Для тих, хто не зрозумів, що це за цифри, поясню: метод write повертає число записаних символів.

Після закінчення роботи з файлом його обов'язково потрібно закрити за допомогою методу close:

```
>>> f.close ()
```

Тепер спробуємо відтворити цей список з отриманого файлу. Відкриємо файл на читання і прочитаємо рядки.

```
>>> f = open ( 'text.txt', 'r')
>>> l = [line.strip () for line in f]
>>> l
[ '0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109',
'1110' , '1211', '1312', '1413 >>> f.close ()
',
```

Ми отримали той же список, що і був. У більш складних випадках (словниках, вкладених кортежів і т. д.), алгоритм запису придумати складніше. Але це і не потрібно. У Python вже давно придумали засоби такі як pickle або json, що дозволяють зберігати у файлі складні структури.

Конструкція with ... as використовується для оборачивания виконання блока інструкцій менеджером контексту. Інколи це більш зручна конструкція, ніж try...except...finally..

Синтаксис конструкции with ... as:

"with" expression ["as" target] ("," expression ["as" target])* ":" suite

Теперь по порядку того, что происходит при виконаннні даного блоку:

1. Виконується вираз в конструкції with ... as.

2. Завантажуется спеціальний метод __exit__ для подальшого використання.

3. Виконується метод __enter__. Якщо конструкція включає в себе слово as, то методом __enter__ значення, що повебртається, записується в змінну.

4. Виконується комплект.

5. Викликається метод __exit__, причому не має значення, чи виконана Suite чи сталося вимкнення. В цей метод передаються параметри вимкнення, якщо воно відбулося, або у всіх аргументах значення Non, якщо не було виключень.

Якщо в конструкції with - as було декілька виразів, то це еквівалентно декільком вкладеним конструкціям:

```
with A() as a, B() as b:
suite
```

Еквівалентно:

```
with A() as a:
with B() as b:
suite
```

Для чого застосовується конструкція with - as? Для гарантування того, що критичні функції виконуються в будь-якому випадку. Самий розповсюджений приклад використання цієї конструкції – відкриття файлів. зпзвичай відкриття файлів здійснюється за допомогою функції open, але конструкція with - as, як правило, є більш зручною і гарантує закриття файлу в будь-якому випадку.

Наприклад:

```
with open('newfile.txt', 'w', encoding='utf-8') as g: d = int(input())
print('1 / {} = {}'.format(d, 1 / d), file=g)
```

І ви можете бути впевнені, що файл буде закритий незалежно від того, що вводив користувач.

3.4.17. Бібліотека pandas

Це основна бібліотека для аналізу даних. Вона входить у так званий NumPy – метамову Pyton, пристосовану для аналізу даних.

3.4.17.1. Структури даних pandas

Щоб почати роботу з pandas, ви повинні освоїти дві основні структури даних: Series і DataFrame. Вони, звичайно, не є універсальним рішенням лю-бій завдання, але все ж утворюють солідну і просту для використання основу більшості додатків.

Series – одновимірний схожий на масив об'єкт, що містить масив даних (будь-якого типу, підтримуваного NumPy) і асоційований з ним масив міток, який називається індексом. Найпростіший об'єкт Series складається тільки з мас-сива даних:

```
In [4]: obj = Series ([4, 7, -5, 3])
In [5]: obj
Out [5]:
0 4
1 7
2 -5
3 3
```

У строковому поданні Series, який відображається в інтерактивному режимі, індекс знаходиться зліва, а значення праворуч. Оскільки ми не задали індекс для даних, то за замовчуванням створюється індекс, що складається з цілих чисел від 0 до N - 1 (де N - довжина масиву даних). Маючи об'єкт Series, отримати уявлення самого масиву і його індексу можна за допомогою атрибутів values і index соответ¬ственно:

```
In [6]: obj.values
OuU6]: array ([4, 7, -5, 3])
In [7]: obj.index
Out [7]: Int64Index ([0, 1, 2, 3])
```

Часто бажано створити об'єкт Series з індексом, що ідентифікує кожен елемент даних:

```
In [8]: obj2 = Series ([4, 7, -5, 3], index = [ 'd', 'b', 'a', 'з'])
In [9]: obj2 Out [9]: d 4 b 7
a -5 з 3
In [10]: obj2.index
Out [lO]: Index ([d, b, a, з], dtype = object)
```

На відміну від звичайного масиву NumPy, для вибірки одного або декількох елементів з об'єкта Series можна використовувати значення індексу:

```
In [11]: obj2 [ 'a']
Out [11]: -5
In [12]: obj 2 [ 'd •] = 6
In [13]: obj2 [[ 'c', 'a', 'd']]
Out [13]: з 3 a -5 d 6
```

Операції з масивом NumPy, наприклад фільтрація за допомогою булева масива, скалярне множення або застосування математичних функцій, зберігають зв'язок між індексом і значенням:

```
In [14]: obj2 Out [14]: d 6 b 7 a -5 c 3
In [15]: obj2 [obj2> Про] Out [15]: d 6 b 7 c 3
In [16]: obj2 * 2 Out [16]: d 12 b 14 a -10 c 6
In [17]: np.exp (obj2) Out [17]: d 403.428793 b 1096.633158 a
0.006738 c 20.085537
```

Об'єкт Series можна також уявляти собі як упорядкований словник фіксованої довжини, оскільки він відображає індекс на дані.

Об'єкт DataFrame представляє табличну структуру даних, що складається з впорядкованої колекції стовпців, причому типи значень (числовий, рядок-вий, логічний і т. Д.) В різних стовпчиках можуть

відрізнятися. В об'єкті DataFrame зберігаються два індексу: по рядках і але стовпчиках. Можна вважати, що це словник об'єктів Series. У порівнянні з іншими схожими на DataFrame структурами, які вам могли зустрічатися раніше (наприклад, data. frame в мові R), операції з рядками і стовпцями в DataFrame в першому наближенні симетричні. Усередині об'єкту дані зберігаються у вигляді одного або декількох двовимірних блоків, а не у вигляді списку, словника або ще який-небудь колекції одновимірних масивів.

Хоча в DataFrame дані зберігаються в двовимірному форматі, у вигляді таблиці, неважко уявити і дані більш високої розмірності, якщо скористатится ієрархічним індексуванням. Цю тему ми обговоримо в наступному розділі, вона лежить в основі багатьох просунутих механізмів обробки даних в pandas.

Є багато способів сконструювати об'єкт DataFrame, один з найбільш розповсюджекних – на основі словника списків однакової довжини або масивів NumPy:

```
data = { 'state': [ 'Ohio', 'Ohio 1,' Ohio ',' Nevada ',' Nevada '],
'Year': [2000, 2001, 2002 2001, 2002],
'Pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame (data)
```

Для отриманого DataFrame автоматично буде побудований індекс, як і в разі Series, і стовпці розташуються по порядку:

```
In [38]: Out [38]: frame
pop state year
0 1.5 Ohio 2000
1 1.7 Ohio 2001
2 3.6 Ohio 2002
3 2.4 Nevada 2001
4 2.9 Nevada 2002
```

Якщо задати послідовність стовпців, то стовпці DataFrame розташуються строго в зазначеному порядку:

```
In [39): DataFrame (data, columns = [ 'year', 'state', 'pop'])
Out [39]
year state pop
0 2000 Ohio 1.5
```

1 2001 Ohio 1.7
2 2002 Ohio 3.6
3 2001 Nevada 2.4
4 2002 Nevada 2.9

Як і в разі Series, якщо запросити стовпець, якого немає в data, то він буде заповнений значеннями NaN:

In [40]: frame2 = DataFrame (data, columns = ['year', 'state', 'pop', 'debt'], ...: index = ['one', 'two', 'three' , 'four', 'five'])
Out [41]:
year state pop debt
про 2000 Ohio 1.5 NaN
1 2001 Ohio 1.7 NaN
2 2002 Ohio 3.6 NaN
3 2001 Nevada 2.4 NaN
4 2002 Nevada 2.9 NaN
In [42]: frame2.columns
Out [42]: Index ([year, state, pop, debt], dtype = object)

Стовпець DataFrame можна витягти як об'єкт Series, скориставшись нотацією словників, або за допомогою атрибута:

In (43]: frame2 ['state'] In [44]: frame2.year
Out [43]: Out [44]:
one Ohio one 2 0 00
two Ohio two 2 001
three Ohio three 2002
four Nevada four 2001
five Nevada five 2 002
Name: state

Відзначимо, що повернутий об'єкт Series має той же індекс, що і DataFrame, а його атрибут name встановлено відповідним чином. Рядки також можна витягти з позиції або по імені, для чого є два методи, один з них – із зазначенням індексного поля:

In (45): frame2.ix ['three']
Out (45): year 2002
state Ohio pop 3.6
debt NaN
Name: three

Стовпці можна модифікувати шляхом привласнення. Наприклад, порожньому стовпчику 'debt 1 можна було б присвоїти скалярний значення або масив зна-чень:

```
In (46): frame2 [ 'debt'] = 16.5 In (47): frame2
Out [47]:
year state pop debt
one 2000 Ohio 1.5 16.5
two 2001 Ohio 1.7 16.5
three 2002 Ohio 3.6 16.5
four 2001 Nevada 2.4 16.5
five 2002 Nevada 2.9 16.5
In (48]: frame2 [ 'deЬt'] - np.arange (5.)
In (49]: frame2
Out [49]:
year state pop debt
one 2 000 Ohio 1.5 0
two 2 001 Ohio 1.7 1
three 2002 Ohio 3.6 2
four 2001 Nevada 2.4 3
five 2002 Nevada 2.9 4
```

3.4.17. 2. Редукція і обчислення описових статистик

Об'єкти pandas оснащені набором стандартних математичних і статистичних методів. Велика їх частина потрапляє в категорію редукцій, або зведених статистичних методів, які обчислюють єдине значення (наприклад, суму або середнє) для Series або об'єкт Series – для рядків або стовпців DataFrame. У порівнянні з еквівалентними методами масивів NumPy, всі вони ігнорують відсутні значення. Розглянемо невеликий об'єкт DataFrame:

```
In [198]: df = DataFrarne ([[l.4, пр.пап], [7.1, -4.5],
[Np.nan, пр.nan], [Про .75, -1.3]],
: Index = [ 'a', 'Ь', 'з', 'd'],
: Columnscc [ 'one', 'two'])
Out [199]:
one two
a 1.40 NaN
```

ь 7.10 -4.5
з NaN NaN
d 0.75 -1.3

Метод sum об'єкта DataFrame повертає Series, що містить суми по стовпцям:

In [200]: df.sum ()
Out [200]: one 9.2 5 two -5.80

Якщо передати параметр axis = l, то підсумовування буде проводитися по рядкам:

In [201]: df.sum (axis = l)
Out [201]:
a 1.40
Ь 2. 60
з NaN
d -0.55

Повний список зведених статистик:

argmin, argmax – Обчислює позицію в індексі (цілі числа), при якому досягається мінімальне або максимальне значення відповідно.

idxmin, idxmax – Обчислює значення індексу, при якому досягається мінімальне або максимальне значення відповідно.

quantile – Обчислює вибірковий квантиль в діапазоні від 0 до 1

sum – Сума значень.

mean – Середнє значення.

median – Медіана (50% -ий квантиль).

mad – Середнє абсолютне відхилення від середнього.

var – Вибіркова дисперсія.

std – Вибіркове стандартне відхилення.

skew – Асиметрія (третій момент).

kurt – Куртозіс (четвертий момент).

cumsum – Наростаюча сума.

cummin, cummax – Наростаючий мінімум або максимум відповідно.

cumprod – Наростання множення.

diff – Перша арифметична різниця (корисно для часових рядів).

pct_change – Обчислює процентну зміну.

Деякі зведені статистики, наприклад кореляція і ковариация, обчислюваляють по парам аргументів. Розглянемо об'єкти DataFrame, що містять ціни акцій і обсяги біржових угод:

```
import pandas.io.data as web all_data = {}
for ticker in [ 'AAPL', 'IBM', 'MSFT', 'GOOG']:
all_data [ticker] = web.get_data_yahoo (ticker, '1/1/2000', '1/1/2010')
price = DataFrame ({tic: data [ 'Adj Close']
for tic, data in all_data.iteritems ()})
volume = DataFrame ({tic: data [ 'Volume']
for tic, data in all_data.iteritems ()})
```

Тепер обчислимо процентні зміни цін:

```
In [209]: returns = price.pct_change ()
Out [210]: Date
2009-12-24
2009-12-28
2009-12-29
2009-12-30
2009-12-31
AAPL
0.034339 0.012294 -0. 011861 0.012147 -0.004300
GOOG
0.011117
0.007098
-0.005571
0.005376
-0.004416
IBM
0.004420
0.013282
-0.003474
0.005468
-0.012609
MSFT
0.002747
0.005479
0.006812
-0.013532
```

-0.015432

Метод corr об'єкта Series обчислює кореляцію перекриваються, відмінність-них від NA, вирівняних за індексом значень в двох об'єктах Series. Відповід-ного, метод cov обчислює ковариацию:

```
In [211]: returns.MSFT.corr (returns.IBM)
Out [211]: O.49609291822168838
In [212]: returns.MSFT.cov (returns.IBM)
Out [212]: O.00021600332437329015
```

З іншого боку, методи corr і cov об'єкта DataFrame повертають відпо-відно повну кореляційний або ковариационную матрицю у вигляді DataFrame:

```
In [213] I: returns. .corr ()
Out [213] I:
AAPL GOOG IBM MSFT
AAPL 1.000000 0.470660 0.410648 0.424550
GOOG 0.470660 1.000000 0.390692 0.443334
IBM 0.410648 0.390692 1.000000 0.496093
MSFT 0.424550 0.443334 0.496093 1.000000
In [214] I: returns. . cov ()
Out [214] 1:
AAPL GOOG IBM MSFT
AAPL 0.001028 0.000303 0.000252 0.000309
GOOG 0.000303 0.000580 0.000142 0.000205
IBM 0.000252 0.000142 0.000367 0.000216
MSFT 0.000309 0.000205 0.000216 0.000516
```

За допомогою методу corrwi th об'єкта DataFrame можна обчислити попарні кореляції між стовпцями або рядками DataFrame і іншим об'єктом Series або DataFrame. Якщо передати йому об'єкт Series, то буде повернуто Series, содер¬жащій значення кореляції, обчислені для кожного стовпця:

```
In [215]: returns.corrwith (returns.IBM)
Out (215]:
AAPL 0.410648 GOOG 0.3 9 06 92 IBM 1.000000 MSFT
0.496093
```

Якщо передати об'єкт DataFrame, то будуть обчислені кореляції стовпців з відповідними іменами. Нижче я обчислюю кореляції процентних зраді-ний з обсягом угод:

```
In [216]: returns.corrwith (volume)
Out [216]:
```

AAPL -0.057461 GOOG 0.062644 IBM -0.007900 MSFT -0.014175

Якщо передати axis = l, то будуть обчислені кореляції рядків. У всіх випадках перед початком обчислень дані вирівнюються по мітках.

3.4.17.3. XML і HTML: як із них вибирати дані

На Python написано багато бібліотек для читання і запису даних в всюди єси-щих форматах HTML і XML. Зокрема, бібліотека lxml (http://lxml.de) відомеа високою продуктивністю при розборі дуже великих файлів. Для lxml є кілька програмних інтерфейсів; спочатку продемонстрємпо інтерфейс lxml. html для роботи з HTML, а потім розберемо XML-документ за допомогою lxml.obj ectify.

Багато сайтів показують дані у вигляді HTM-таблиць, зручних для просмо¬тра в браузері, але не пропонують їх в таких машинозчитуваних форматах, raJSON або XML. Так, наприклад, йде справа з даними про біржових опціонах на сайті Yahoo! Finance. Опціон – це похідний фінансовий інструмент (дериватив), який дає право купувати (опціон на придбання, або колл-опціон) або продавати (опціон на продаж, або пут-опціон) акції компанії за деякою ціною (ціною виконання) в проміжку часу між поточним моментом і деяким фіксованим моментом в майбутньому (кінцевою датою). Колл- і пут-опціони торгуються з різними цінами виконання і кінцевими датами; ці дані можна знайти в таблицях на сайті Yahoo! Finance.

Для початку вирішіть, з якої URL-адреси ви хочете завантажувати дані, потім відкрийте його за допомогою засобів з бібліотеки urllib2 і розберіть потік, користуючись xml:

```
from lxml.html import parse
from urllib2 import urlopen. .
parsed = parse (urlopen ( 'http://finance.yahoo.com/q/op?s=AAPL+Options'))
```

```
doc = parsed.getroot ()
```

Маючи цей об'єкт, ми можемо вибрати все HTML-теги зазначеного типу, наприклад, теги table, всередині яких знаходяться дан, що нас.цікавлять. Для прикалду отримаємо список усіх активних посилань в документі, вони представляються в HTML тегом A. Викличемо метод findall кореневого елемента документа, передавши йому вираз XPath (це мова, на якому записуються «запити» до документу):

```
In [906]: links = doc.findall ( '.// a')
In [907]: links [l5: 20] Out [907]:
[<Element a at 0x6c488f0>,
<Element a at 0x6c48950>,
<Elernent a at 0x6c489Ь0>,
<Elernent a at 0x6c48a10>,
<Elernent a at 0x6c48a70>]
```

Але це об'єкти, що представляють HTML-елементи; щоб отримати URL і текст посилання, нам потрібно скористатися методом get елемента (для отримання URL) або методом text_content (для отримання тексту):

```
In [908]: lnk = links [28]
In [909]: lnk
0ut [909]: <Element a at 0x6c48dd0>
In [910]: lnk.get ( 'href')
0ut [910]: 'http://biz.yahoo.com/special.html'
In (911]: lnk.text_content ()
Out [911]: 'Special Editions'
```

Таким чином, отримання усіх активних посилань в документі зводиться до включення за списком:

```
In (912]: urls = [lnk.get ( 'href') for lnk in doc.findall ( '.// a')]
In (913]: urls [-10:]
Out [913]:
[ 'Http://info.yahoo.com/privacy/us/yahoo/finance/details.html',
'Http://info.yahoo.com/relevantads/',
'Http://docs.yahoo.com/info/terms/',
'Http://docs.yahoo.com/info/copyright/copyright.html',
'http://help.yahoo.com/l/us/yahoo/finance/forms_index.html', 'http:
// help .yahoo.eom / l / us / yahoo / finance / quotes /
fitadelay.html ','
http://help.yahoo.com/l/us/yahoo/finance/quotes/fitadelay.html ',
'Http://www.capitaliq.com',
```

'Http://-www.csidata.com',
'Http: //w^.morningstar.com/']

Що стосується відшукання потрібних таблиць в документі, то це робиться методом проб і помилок; на деяких сайтах рішення цього завдання спрощується, тому що таблиця має атрибут id. Я знайшов, які таблиці містять дані про колл- і пут-опціони:

tables = doc.findall ('.// table') calls = tables [9] puts = tables [13]
У кожній таблиці є рядок-заголовок, а за нею йдуть рядки з даними:
In [915]: rows = calls.findall ('.// tr')

Для всіх рядків, включаючи заголовок, ми хочемо отримати текст з кожного осередку; в разі заголовка осередками є елементи TR, а для рядків даних – елементи TD:

def _unpack (row, kind = 'td'):
elts = row.findall ('.//% s'% kind) return [val.text_content () for val in elts]

Таким чином, отримуємо:

In [917]: _unpack (rows [O], kind = 'th')
Out [917]: ['Strike', 'Syi ^ iol', 'Last', 'Chg', 'Bid', 'Ask', 'Vol', 'Openlnt'
In [918]: _unpack (rows [l], kind = 'td')
Out [918]:
['295.00',
'AAPL12 0818C002 95 000',
'310.40',
'0.00',
'289.80',
'290.80',
'1',
'169']

Тепер для перетворення даних в об'єкт DataFrame залишилося об'єднати всі описані кроки разом. Оскільки числові дані як і раніше записані у вигляді рядків, можливо, буде потрібно перетворити деякі, але не всі стовпці в формат з плаваючою точкою. Це можна зробити і вручну, але, на щастя, в бібліотеці pandas є клас TextParser, який використовується для здійснення read_ csv і іншими функціями розбору для автоматичного перетворення типів:

from pandas.io.parsers import TextParser

```
def parse_options_data (table): rows = table.findall ( '.// tr') header
= _unpack (rows [O], kind = 'th') data = [_unpack (r) for r in rows
[l:]] return TextParser (data, names = header) .get_chunk ()
```

Нарешті, викликаємо цю функцію розбору для табличних об'єктів lxml і отримуємо результат у вигляді DataFrame:

```
In [920]: call_data = parse_options_data (calls)
In [921]: put_data = parse_options_data (puts)
In [922]: call_data [: 10]
Out[922]:
Strike Symbol Last Chg Bid Ask Vol Open Int
o 295 AAPL120818C00295000 310.40 o.o 289.80 290.80 1 169
1 300 AAPL120818C00300000 277.10 1.7 284.80 285.60 2 478
2 305 AAPL120818C00305000 300.97 o.o 279.80 280.80 10 316
3 310 AAPL120818C00310000 267.05 o.o 274.80 275.65 6 239
4 315 AAPL120818C00315000 296.54 o.o 269.80 270.80 22 88
5 320 AAPL120818C00320000 291.63 o.o 264.80 265.80 96 173
6 325 AAPL120818C00325000 261.34 o.o 259.80 260.80 N/A 108
7 330 AAPL120818C00330000 230.25 o.o 254.80 255.80 N/A 21
8 335 AAPL120818C00335000 266.03 o.o 249.80 250.65 4 46
9 340 AAPL120818C00340000 272.58 o.o 244.80 245.80 4 30
```

XML (розширювана мова розмітки) - ще один популярний формат представ¬ленія структурованих даних, що підтримує ієрархічно вкладені дані, забезпечені метаданими.

Продемонструємо альтернативний інтерфейс, зручний для роботи з XML-даними, – lxml .objectify.

Управління міського транспорту Нью-Йорка (MTA) публікує тимчасові ряди з даними про роботу автобусів і електричок (http: j / www.mta.info / developers / download.html). Ми зараз розглянемо дані про якість обслуговування, зберігається у вигляді XML-файлів. Для кожної автобусної та залізничної компанії існує свій файл (наприклад, Performance_MNR. Xml для компанії MetroNorth Railroad), що містить дані за один місяць в вигляді послідовності таких XML-повідомлень

```
<INDICATOR>
<INDICATOR_SEQ> 3 73 889 </ INDICATOR_SEQ>
<PARENT_SEQ> </ PARENT_SEQ>
```

<AGENCY_N ^ E> Metro-North Railroad </ AGENCY_NAME>
<INDICATOR_NAME> Escalator Availability </ INDICATOR_NAME> <DESCRIPTION> Percent of the time that escalators are operational systemwide. The availability rate is based on physical observations performed the morning of regular business days only. This is a new indicator the agency began reporting in 2009. </ DESCRIPTION>
<PERIOD_YEAR> 20ll </ PERIOD_YEAR>
<PERIOD_MONTH> l2 </ PERIOD_MONTH>
<CATEGORY> Service Indicators </ CATEGORY>
<FREQUENCY> M </ FREQUENCY>
<DESIRED_CHANGE> U </ DESIRED_CHANGE>
<INDICATOR_UNIT>% </ INDICATOR_UNIT>
<DECIMAL_PLACES> l </ DECIMAL_PLACES>
<YTD_TARGET> 97.00 </ YTD_TARGET>
<YTD_ACTUAL> </ YTD_ACTUAL>
<MONTHLY_TARGET> 97.00 </ MONTHLY_TARGET>
<MONTHLY_ACTUAL> </ MONTHLY_ACTUAL>
</ INDICATOR>

Використовуючи lxml. obj ecti fy, ми розбираємо файл і отримуємо посилання на кореневий вузол XML-документа від методу getroot:

```
from lxml import objectify
path = 'Performance_MNR.xml' parsed = objectify.parse (open (path))
root = parsed.getroot ()
```

Властивість root. INDICATOR повертає генератор, послідовно віддає всі елементи <INDICATOR>. Для кожного запису ми заповнюємо словник імен тегів (наприклад, YTD_ACTUAL) значеннями даних (деякі теги пропускаються):

```
data = []
skip_fields = [ 'PARENT_SEQ', 'INDICATOR_SEQ',
'DESIRED_CHANGE', 'DECIHAL_PLACES']
for elt in root.INDICATOR:
el_data = {}
for child in elt.getchildren (): if child.tag in skip_fields: continue
el_data [child.tag] = child.pyval data.append (el_data)
```

Нарешті, перетворимо цей список словників в об'єкт DataFrame:

```
In [927]: perf = DataFrame (data)
In [928]: perf Out [928]:
Empty DataFrame
Columns: array ([], dtype = int64) Index: array ([], dtype = int64)
```

XML-документи можуть бути набагато складніше, ніж в цьому прикладі. Зокрема, в кожному елементі можуть бути метадані. Розглянемо тег гіперпосилання в форма¬те HTML, який є окремим випадком XML:

```
from StringlO import StringlO
tag = '<a href = "http: //^www.google.com"> Google </a>'
root = objectify.parse (StringIO (tag)). getroot ()
```

Тепер ми можемо звернутися до будь-якого атрибуту тега (наприклад, href) або до тексту посилання:

```
In [930]: root
0ut [930]: <Element a at 0x88bd4b0>
In [931]: root.get ( 'href')
Out [931]: 'http: //^www.google.com'
In [932]: root.text
Out [932]: 'Google'
```

У pandas є також підтримка для читання табличних даних в форматі Excel 2003 (і більш пізніх версією) за допомогою класу ExcelFiie. На внутрішньому рівні ExcelFile користується пакетами xlrd і openpyxl, тому їх потрібно попередньо встановити. Для роботи з ExcelFile створіть його екземпляр, передавши конструктору шлях до файлу з розширенням xls або xlsx:

```
xls_file = pd.ExcelFile ( 'data.xls')
```

Прочитати дані з робочого листа в об'єкт DataFrame дозволяє метод parse:

```
table = xls_file.parse ( 'Sheetl')
```

Взаємодія з HTML і Web AP /

Багато сайтів надають відкритий API для отримання даних в форматі JSON або якомусь іншому. Отримати доступ до таких API з Python можна різними способами; рекомендується простий пакет requests (http.j / docs.python-requests. org). Для пошуку за словами «python pandas» в Твіттері ми можемо відправити такий HTTP-запит GET:

```
In [944]: import requests
In [945]: url = 'http://search.twitter.com/search.json?q=python%20pandas'
In [946]: resp = requests.get (url)
In (947): resp
```

Out [947]: <Response [200)>

У об'єкта Response має атрибут text, в якому зберігається вміст відповіді на запит GET. Багато API в веб повертають JSON-рядок, яку слід завантажити в об'єкт Python:

```
In [948): import json
In [949]: data = json.loads (resp.text)
In [950]: data.keys ()
Out [950]:
[U'next_page ', u'completed_in', u'max_id_str ', u'since_id_str',
u'refresh_url ', u'results', u 'since_id', u'results_per_page ',
u'query', u'max_id ' , u'page ']
```

Поле відповіді results містить список твітів, кожен з яких представлений таким словником Python:

```
{U'created_at ': u'Mon, 25 Jun 2012 17:50:33 +0000', u1from_user
': u'wesmckinn', u'from_user_id ': 115494880, u'from_user_id_str':
u'l15494880 ', u'from_user_name ': u'Wes McKinney •, u'geo':
None,
u'id ': 217313849177686018,
u'id_str ': u'217313849177686018',
u'iso_language_code ': u'pt',
u'metadata ': {u'result_type': u'recent '},
u'source ': u' <a href="http://twitter.com/"> web </a> ',
u'text ': u'Lunchtime pandas-fu http://t.co/SI70xZZQ #pydata',
u'to_user ': None,
u'to_user_id ': 0,
u'to_user_id_str ': u'O',
u'to_user_name ': None}
```

Далі ми можемо побудувати список полів твіту, що нас цікавлять та передати його конструктору DataFramc:

```
In [951]: tweet_fields = [ 'created_at', 'from_user', 'id', 'text']
In [952]: tweets = DataFrame (data [ 'results'], columns =
tweet_fields)
In [953]: tweets Out [953]:
<Class 'pandas.core.frame.DataFrame'>
Int64Index: 15 entries, Про to 14 Data columns:
created_at 15 non-null values from_user 15 non-null values
id 15 non-null values
text 15 non-null values
dtypes: int64 (1), object (3)
```

Тепер в кожному рядку DataFrame знаходяться дані, витягнуті з одного твіту:

```
Out [121]: created_at frorn_user id
In [121]: tweets.ix [7]
Thu, 23 Jul 2012 9:54:00 +0000
deblike
227419585803059201
text pandas: powerful Python data analysis toolkit Name: 7
```

3.4.17.4. Побудова графіків та візуалізація

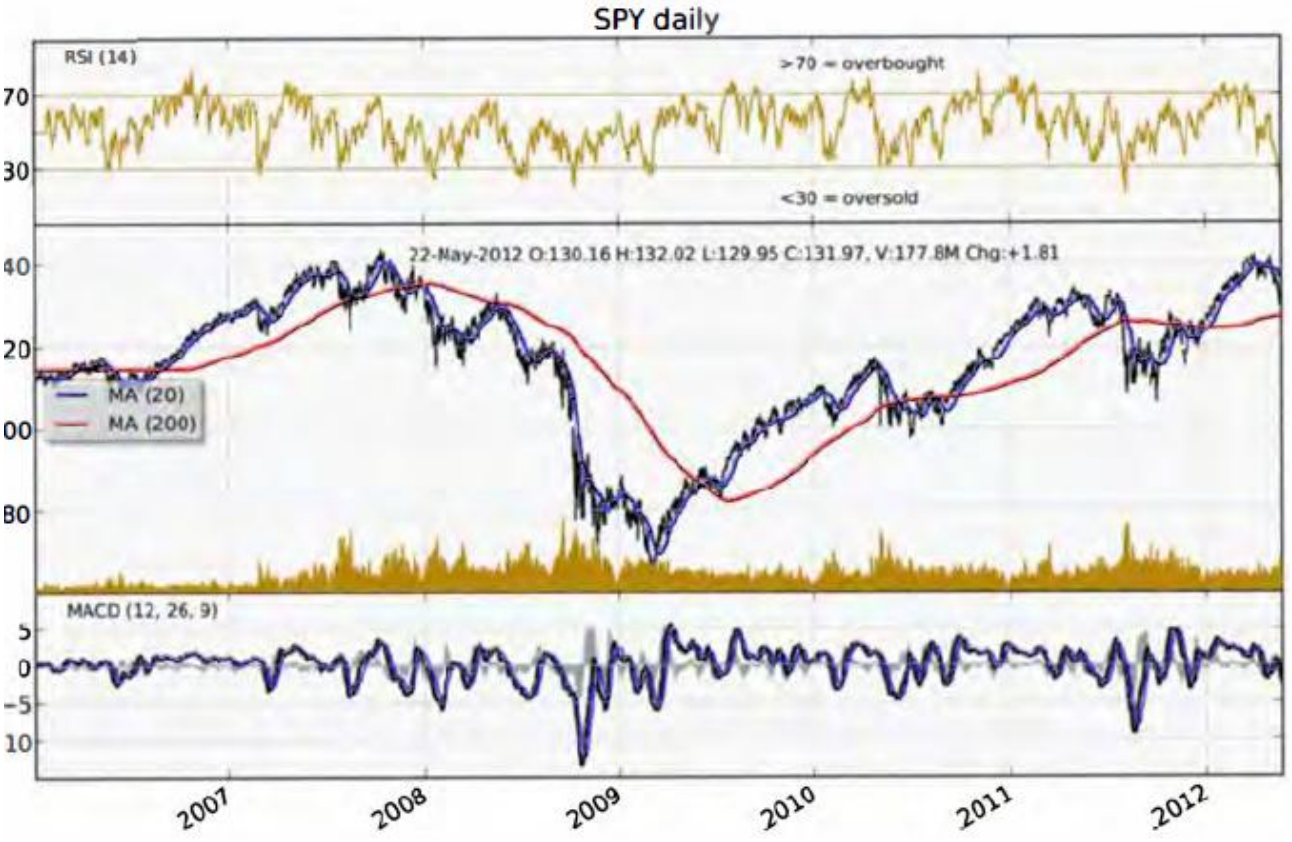
Побудова графіків, а також статична або інтерактивна візуалізація – одне з найважливіших завдань аналізу даних. Вони можуть бути частиною процесу дослідження, наприклад, застосовуватися для виявлення викидів, ухвали необхідних перетворень даних або пошуку ідей для побудови моделей. В інших випадках побудова інтерактивної візуалізації для веб-сайту, наприклад за допомогою бібліотеки d3.js (http: //d3js.or-g/), може бути кінцевою метою. Для Python є багато інструментів візуалізації.

Matplotlib – це пакет для побудови графіків (головним чином, двовимірних) поліграфічної якості. При використанні в поєднанні з якою-небудь бібліотекою ГІП (наприклад, всередині IPython), matplotlib набуває інтерактивні можливості: панорамування, масштабування і інші. Цей пакет підтримує різноманітні системи ГІП у всіх операційних системах, а також вміє експортувати графічні дані у всіх векторних і растрових форматах: PDF, SVG, JPG, PNG, BMP, GIF і т. Д.

Для matplotlib є цілий ряд додаткових бібліотек, наприклад mpiot3d для побудови тривимірних графіків і basemap для побудови карт і проекцій. Для опрацювання наведених в цьому пункті прикладів коду, не забудьте завантажити IPython в режимі pylab (ipython --pylab) або включити інтеграцію з циклом обробки подій ГІП за допомогою функції% gui.

Взаємодіяти з matplotlib можна декількома способами. Самий поширенням странения – запустити IPython в режимі pylab за допомогою команди ipython --pylab. В результаті IPython конфігується для підтримки обраної системи ГІП (Tk, wxPython, PyQt, платформний ГІП OS X, GTK). Для більшості користувачів мається на увазі за замовчуванням системи ГІП. У режимі pylab в IPython також імпортується багато модулів і функцій, щоб інтерфейс був більше схожий на MATLAB. Переконатися, що все працює, можна, побудувавши простийграфік:

plot (np.arange (lO))



Якщо все налаштовано правильно, то з'явиться нове вікно з лінійним графіком. Його можна закрити мишею або ввівши команду close (). Всі функції matplotlib API, зокрема plot і close, знаходяться в модулі matplotlib.pyplot, при імпорті якого зазвичай дотримуються наступної угоди:

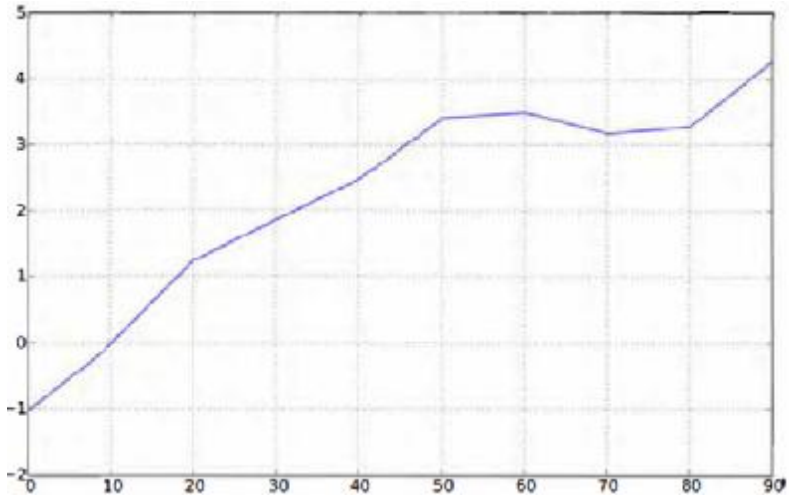
import matplotlib.pyplot as plt

Як ви могли переконатися, matplotlib – бібліотека досить низького рівня. Графік в ній складається з базових компонентів: спосіб відображення даних

(тип графіка: лінійний графік, стовбчаста діаграма, коробчаста діаграма, діаграма розсіювання, контурний графік і т. д.), Пояснювальий напис, назва, мітки рисок і інші анотації. Почасти так зроблено тому, що в багатьох випадках дані, необхідні для побудови повного графіка, розкидані по різних об'єктах. У бібліотеці pandas у нас вже є мітки рядків, мітки стовців і, можливо, інформація про угруповання. Це означає, що багато графіків, для побудови яких засобами matplotlib довелося б писати багато коду, в pandas можуть бути побудовані за допомогою одного-двох коротких команд. Тому в pandas є багато високорівневих методів побудови графіків для стандартних типів візуалізації, в яких використовується інформація про вну¬тренней організації об'єктів DataFrame.

У об'єктів Series і DataFrame є метод plot, який вміє будувати гра¬фікі різних типів. За замовчуванням він будує лінійні графіки:

```
In [55]: s = Series
(np.random.randn
(lO) .cumsum (),
index = np.arange
(O, 100, 10)) In
[56]: s.plot ()
```



Індекс об'єкту Series передається matplotlib для нанесення рисок на вісь X, але це можна відключити, задавши параметр use_index = False. Ризики і діапазон значень на осі X можна налаштовувати за допомогою параметрів xticks і xlim, а на осі Y – за допомогою параметрів yticks і y1im. Повний перелік параметрів методу plot наведено нижче:

label – Мітка для пояснювальної написи на графіку.

ax – Об'єкт подграфіка matplotlib, всередині якого будувати графік. Якщо параметр не заданий, то використовується активний подграфік.

Style – Рядок стилю, наприклад 'ko--', що передається у matplotlib.

alpha – Рівень непрозорості графіка (число від 0 до 1).

kind – Може приймати значення 'line', 'bar' barh '; «kde».

logy – Використовувати логарифмічний масштаб по осі Y.

use_index – Брати мітки рисок з індексу об'єкта.

rot – Кут повороту міток рисок (від 0 до 360).

xticks – Значення рисок на осі X.

yticks – Значення рисок на осі Y.

xlim – Межі по осі X (наприклад, [0, 10])

ylim – Межі по осі Y.

grid – Відображати координатну сітку (за замовчуванням включено).

3.5. R (Ар)

R – мова програмування і програмне середовище для статистичних обчислень, аналізу та представлення даних в графічному вигляді.

R розповсюджується безкоштовно за ліцензією GNU General Public License у вигляді вільнодоступого вихідного коду або відкомпільованих бінарних версій більшості операційних систем: Linux, FreeBSD, Microsoft Windows, Mac OS X, Solaris. R використовує текстовий інтерфейс, однак існують різні графічні інтерфейси.

R має значні можливості для здійснення статистичних аналізів, включаючи лінійну і нелінійну регресію, класичні статистичні тести, аналіз часових рядів (серій), кластерний аналіз і багато іншого. R легко розбудовується завдяки використанню додаткових функцій і пакетів доступних

на сайті Comprehensive R Archive Network (CRAN). Більша частина стандартних функцій R, написана мовою R, однак існує можливість підключати код написаний C, C++, або Фортраном. Також за допомогою програмного коду на C або Java можна безпосередньо маніпулювати R об'єктами.

3.5.1. Інсталяція R

R доступний на сайті Comprehensive R Archive Network (CRAN) або одному з його дзеркал за відповідними посиланнями http://cran.r-project.org/mirrors.html.

Середовище R доступне для різних дистрибутивів Linux і Unix подібних систем, а також операційних систем типу Windows і MacOS.

У випадку операційної системи Windows вибирається одне з дзеркал сайту CRAN, завантажується інсталяційний *.exe файл і встановлюється у вибрану директорію/папку операційної системи Windows.

В Windows запускаються на виконання файли R.exe (робота в терміналі Rterm) або Rgui.exe (робота з графічним інтерфейсом RGui). Існує ряд графічних редакторів та інтегрованих середовищ IDE, які дозволяють оптимізувати написання програм на R в залежності від інди¬відуальних потреб і уподобань (Див. перелік на сайті http://uk.wikipedia.0rg/wiki/R_(M0Ba_np0rpaMyBanHM).

Команди або функції R записані в текстовому файлі у вигляді скрипту (програми на R) можна завантажити і запустити на виконання використовуючи функцію source(). Наприклад маємо текстовий файл *Scrypt.R* який мість функцію print (). В результаті запуску скрипту *Scrypt.R* отримаємо

```
> source("Scrypt.R")
[1] "Просто чудовий день!"
```

Встановлючи параметр ec/io=TRUE в консолі виводимуться

команди/функції R перед результатом їхнього виконання

```
>       source("Scrypt.R",echo=TRUE)
>       print("Просто чудовий день!")
[1] "Просто чудовий день!"
```

Програми написані в R і збережені у вигляді скриптів (файли розширенням **.R)* можна також запускати у фоновому режимі. Наприклад результат виконання програми або скрипту *my script. R* скеровується в окремий файл *resultaty.out*

```
#       в UNIX системах
E CMD BATCH [options] my_script.R resultaty.out
#       в Windows
"C:\R-2.11.0\bin\R.exe" CMD BATCH
—vanilla —slave "c:\my projects\my_script.R
```

Пакети або бібліотеки в R представляють собою колекції функцій і даних призначені для вирішення певного типу задач. Середовище R інсталюється з набором пакетів, повний список, разом з коротким описом, яких можна продивитися за допомогою команди library ()

```
>       library()
Packages in library 1/usr/lib/E/library' :
base
boot
class
cluster
codetools
datasets
foreign
graphics
grDevices
grid
KernSmooth
lattice
MASS
The E Base Package
Bootstrap E (S-Plus) Functions (Canty)
Functions for Classification
Cluster Analysis Extended Eousseeuw et al.
```

Code Analysis Tools for E The E Datasets Package
Eead Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase, ...
The E Graphics Package
The E Graphics Devices and Support for Colours and Fonts
The Grid Graphics Package
Functions for kernel smoothing for Wand & Jones (1995)
Lattice Graphics
Main Package of Venables and Eipley's MASS

Частина зы встановлених пакетів завантажуються одночасно разом з початком роботи R. Функція search () виводить назви завантажених в середовище R пакетів.

```
>  searchQ
"package:graphics" "package:datasets" "package:base"
[1] ".GlobalEnv"        "package:stats'
[4] "package:grDevices" "package:utils'
[7] "package:methods" "Autoloads"
```

Можливості R значно розширюються за рахунок використання додаткових пакетів. Додаткові пакети знаходяться у вільному доступі на сайті CRAN або можуть бути написані безспосередньо самим користувачем. Існує кілька варіантів способів інсталяції нових пакетів:

В першу чергу слід завантажити необхідний пакет paketname з сайту CRAN. За допомогою наступної команди з консолі Unix певний пакет paketname, наприклад, інсталюється в папку /myfolder/R-paekages/

```
$ E CMD INSTALL paketname -1 /myfolder/R-packages/
```

Пакети з сайту CRAN можна заінсталювати безпосередньо за допомогою консолі R. Для цього слід скористатится наступною командою

```
>  install.packages("paketname")
```

або

```
>  install.packages("paketname", lib="/myfolder/R-packages/")
```

і вибрати з якого дзеркала сайту CRAN встановлюватимется пакет.

Функції і набори даних додаткових пакетів стають доступними до використання після завантаження (підключення) пакетів в середовище R. Щоб підключити вже заінстальований, додатковий пакет використовується функція

library ()

```
> library("mypackage")
# або вказуючи явно місце знаходження пакету в файловій системі
> library("mypackage", lib.loc=".../librariesFolder/")
```

де mypackage – назва пакету/бібліотеки, що підключається, lib.loc – місце знаходження пакету. Функція .libPathsQ виводить адреси місця знаходження пакетів в операційній системі. Наприклад

```
#       в Unix подібних системах
>       .libPathsQ #
[1]     "/usr/lib/R/library"
[2]     "/usr/share/R/library"
[3]     "/home/X/R/i386-redhat-linux-gnu-library/2.11"
#       в Windows
>       .libPathsQ
[1] "C:/R/R-211~l.1/library"
```

3.5.4. Об’єкти і типи даних в R

R представляє дані у вигляді структурованих об’єктів, таких як вектори, матриці, масиви, фактори, списки, датафрейми. Даний розділ демонструє як створюються об’єкти в середовищі R в залежності від типу даних.

Існують три типи векторів числовий, символьний і логічний. Вектори задаються наступним чином

```
>     vektor <- c(datal,data2,data3,...)
```

де конструкція c – абрівіатура від анг. collection. Приклади

```
>     a <- c(2,4,6,8,10) # числовий вектор
>     a
[1] 2 4 6 8 10
>     bl <- c("Kharkiv","Kyiv","Lviv") # символьний вектор
>     bl
[1] "Kharkiv" "Kyiv" "Lviv"
```

Числа можуть інтерпретуватися як символьні дані (наприклад, поштовий індекс):

```
>       b2 <- c("64000","01000","79000") # символьний вектор
>       b2
[1] "64000" "01000" "79000"
>       cl <- c(TRUE,FALSE,TRUE,TRUE) # логічний вектор
>       cl
[1] TRUE FALSE TRUE TRUE
>       c2 <- (a > 9)
>       c2
[1] FALSE FALSE FALSE FALSE TRUE
```

Вектор може задаватися у вигляді набору послідовних чисел за допомогою функції seq()

```
>       dl <- (1:15)
>       dl
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
>       d2 <- seq(10,0,by=-l)
>       d2
[1] 10 9876543210
>       seq(0,l,length.out=21)
[1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50
[12] 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00
```

або реплікацій, повторень чисел чи векторів за допомогою функції rep()

```
>       f <- rep(1,5)
>       f
[1] 1 1 1 1 1
>       g <- rep(a,3)
>       g
[1] 2 4 6 8 10 2 4 6 8 10 2 4 6 8 10
```

Показати елементи вектора

```
>     a[c(1,4)] # показати 1-й і 4-й елементи вектора a [1] 2 8
```

Фактори являють собою реалізацію символьного вектора і його номінальних значень, як результат класифікації/групування елементів символьного вектора. Фактори задаються за допомогою функції factor () наступним чином

```
>     f <- factor(x = character(),...)
```

де x – вектор символьних значень,... - додаткові аргументи. Нехай існує вектор, елементи якого містять варіанти відповідей на питання ”Так”, ” Ні” або ” Незнаю”:

```
> answers <-
c("Так","Так","Ні","Незнаю","Так","Так","Незнаю",
"Ні","Ні","Ні" )
> answers
[1] "Так" "Так" "Ні" "Незнаю" "Так" "Так"
[7] "Незнаю" "Ні" "Ні" "Ні"
```

Фактори показують інформацію про категорії або рівні (Levels) за якими групуються дані.

```
> factor1 <- factor(answers)
> factorl
[1] Так Так Ні Незнаю Так Так Незнаю
[8] Ні Ні Ні Levels: Незнаю Ні Так
```

Фактор являється більш ефективним об'єктом зберігання символьних даних, що повторюються, особливо коли існує великий об'єм символьних даних. Рівні фактора кодуються середовищем R як цілі числа, відповідно фактор можна представити в закодованому вигляді з використанням функції as . integer ()

```
> as.integer(factorl)
[1] 3321331222
```

Загальну кількісну інформацію по категоріям, в даному випадку, варіантам відповідей "Так", "Ні". ''Незнаю", можна отримати за допомогою функції table ()

```
> table(factorl) factorl
Незнаю Ні Так
2 4 4
```

Фактор також можна згенерувати за допомогою функці gl()

```
> gl(l,4)
[1] 1 1 1 1 Levels: 1
> gl(2,4)
[1] 11112222 Levels: 1 2
> gK3,l)
[1] 1 2 3 Levels: 12 3
```

Змінити назви рівнів можна використовуючи аргумент label

```
> gl(3,1,20,label = c("Low","Middle","Top"))
[1] Low Middle Top Low Middle Top Low
[8] Middle Top Low Middle Top Low Middle [15] Top
Low Middle Top Low Middle Levels: Low Middle Top
```

Часові ряди або ж часові серії, являють собою об’єкт, який дозволяє зберігати змінні в часі дані. Часові серії створюються за допомогою функції ts() і складаються з даних (у вигляді векторів, матриці чисел) і дат, що розміщенні між собою з певним заданим часовим інтервалом.

В загальному випадку синтаксис функції ts() має вигляд

```
>     ts(data, start, frequency,...)
```

де data – дані часової серії, start – час першої обсервації, frequency – число обсервацій за цикл (одиницю) часу. Приклад часової серії випадкових чисел з 2010 по 2012 рік.

```
>     timeseries <- ts(data=rnorm(34), frequency = 12,
>     start = c(2010))
>     plot(timeseries)
```

Матриця в R представляє собою вектор у вигляді 2-х вимірної таблиці, розміром *n* x *m*, всі елементи якої належать до числового типу даних.

Загальний формат запису матриці з використанням функції matrix () має вигляд

```
>     matrixname <- matrix(vector,nrow,mcol)
```

де vector – набір даних у вигляді вектора, nrow – кількість рядків, ,mcol – кількість стовбців. Приклад матриці розміром 3x5

```
>   matrixl <- matrix(l:15,nrow=3,ncol=5)
>   matrixl
    [Д]   C, 2]  [,3]   [ >4]  C, 5]
[1,] 1      4      7      10     13
[2,] 2      5      8      11     14
[3,] 3      6      9      12     15
```

Функція dim (matrix) дозволяє показати розмір матриці

```
>       dim(matrixl)
[1] 3 5
```

Матрицю можна згенерувати з векторів розміщуючи їх рядками або стовпчиками за допомогою функцій rbindQ та cbindQ відповідно.

```
>       rbind(c(l,3,5),c(6,4,2),c(0,0,0),c(7,8,9)) [,1] C,2] [,3]
[1,]   1      3      5
[2,]   6      4      2
[3,]   0      0      0
```

```
[4,]    7       8       9
> cbind(c(l  ,3,5)  ,c(6    ,4,2),c(0,0,0),c(7,8,9))
        [,1] [  ,2] [   ,3]     [ >4]
[1,]    1       6       0       7
[2,]    3       4       0       8
[3,]    5       2       0       9
```

Знаходження елементу, рядка, стовбчика матриці за їхніми індексами ви-глядає наступним чином.

Нехай маємо матрицю

```
>   matrix2<- matrix(c(l,3,5,6,4,2,0,0,0,7,8,9),2,6)
>   matrix2
[,1] C, 2] [,3] [,4] [, 5] [,6]
[1,] 1 5 4 0 0 8 [2,] 3 6 2 0 7 9
```

тоді

```
>   matrix2[1,] # перший рядок матриці matrix2 [1] 1 5 4 0 0
8
>   matrix2[,6] # шостий стовбчик матриці matrix2 [1] 8 9
>   matrix2[2,3] # елемент 2-го ряду і 3 стовпчика матриці
matrix2 [1] 2
```

На відміну від матриці, масив являє собою вектор значень vector у вигляді таблиці, в якій може бути довільна кількість вимірів к. Синтаксис запису масиву в R задається наступним чином

```
>     arrayname <- array(vector,k)
```

Приклад 3-х вимірного масиву який складається з 4-х таблиць (матриць):

```
>   array(1:16,c(2,2,4))
, , 1
[,1] C, 2]
[1,]1       3
[2,]2       4
, , 2
[,   1]     [,2]
[1,]5       7
[2,]
, , 3       6       8
[,   1]     [,2]
[1,]9       11
[2,]10      12
, , 4
[,   1]     [,2]
```

```
[1,]13  15
[2,]14  16
```

Незалежно від розташування елементів матриці або масиву, всі елементи доступні, як елементи одного вектора.

Об’єкт formula відіграє важливу роль в статистичних розрахунках з використанням середовища R, окреслюючи модель згідно якої аналізуватимуться дані. Наведені нижче приклади відображають запис формули відповідо до аналітичного вигляду статистичної моделі

Для лінійної залежності з двома незалежними змінними x_1 і x_2 виду

$$y = b_0 + b_1 x_1 + b_2 x_2 + \epsilon$$

формула моделі відповідно має вигляд

y ~ x1+x2

Лінійна залежність включає в себе точку перетину з віссю y. Наступна формула дозволя виключити точку перетину з віссю ординат

y ~ 0+x
y ~ -1 + x
y ~ x - 1

Математичні оператори $\times, -, \hat{}$, i : у випадку використання їх у формулах лінійної регресії приймають зовсім інший зміст.

- Оператор :

Формула y ~ x1:x2

відповідає лінійній залежності виду

$$y = b_0 + b_{12} x_1 x_2 + \epsilon$$

- Оператор x

Формула y ~ x1*x2

еквівалентна запису y ~ xl+x2+xl:x2 і відповідає лінійній залежності виду

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_{12} x_1 x_2 + \epsilon$$

- Оператор ^

Формула y ~ (x1+x2)^2

еквівалентна запису y ~ xl+x2+xl:x2

і відповідає залежності виду

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_{12} x_1 x_2 + \epsilon$$

- Оператор -

Використовується для усунення компонент з формули. Наприклад формула

y ~ (xl+x2+x3)^2 - x1:x3

еквівалентна запису

y ~ xi + x2 + x3 + x1:x2 + x2:x3

і в аналітичній формі відповідає рівнянню

$$y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_{12} x_1 x_2 + b_{23} x_2 x_3 + \epsilon$$

- Оператор I(...)

дозволяє записувати компоненти формули з операторами $\times, -, \hat{}, \mathrm{i}$: як ps звичайними арифметичними операторами. Наприклад формула

y ~ x1+I(x2+x3)

в аналітичній формі відповідає залежності

$$y = b_0 + b_1 x_1 + b_2 (x_2 + x_3) + \epsilon$$

Датафрейми є одними з основних і фундаментальних об'єктів середовища R, який можна охарактеризувати, як матрицю з різними типами даних. В стовпчиках такої матриці можуть зберігатися числові, символьні, логічні вектори, фактори, матриці чисел, інші блоки даних. Символьні вектори автоматично конвертуються у фактори. Датафрейми створюються з використанням функції data.frameQ. Загальний формат запису датафрейму має вигляд

> dataframename<-data.frame(varl,var2,var3...varN)

де varl,var2,var3...varN – змінні різних типів.

Приклад датафрейму:

```
> datal <- c(101,102,103,104)
> data2 <- c("Lviv", "Kyiv", "Kharkiv", NA)
> data3 <- c("79000","01000","64000","00000")
> data4 <- c(TRUE,FALSE,TRUE,FALSE)
> alldata <- data.frame(datal,data2,data3,data4)
> names(alldata) <- c("ID","City","PostID","Check")
> alldata
ID City PostID Check
1 101 Lviv 79000 TRUE
2 102 Kyiv 01000 FALSE
3 103 Kharkiv 64000 TRUE
4 104 <NA> 00000 FALSE
```

Кожний стовпчик або змінна датафрейму має унікальне імя. Вміст змінної датафрейму можна отримати за допомогою команди, яка складається з імені датафрейму, знаку $ та імені змінної dataframe$t>aria&/e

```
> alldata$City
[1] Lviv Kyiv Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv
> alldata$PostID
[1] 79000 01000 64000 00000 Levels: 00000 01000 64000 79000
```

Використання функції attach () дозволяє звертатися до змінних датафрейму безпосередньо за назвою/іменем змінної.

```
> attach(alldata)
> City
[1] Lviv Kyiv Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv
> PostID
[1] 79000 90000 64000 00000 Levels: 00000 01000 64000 79000
```

Щоб повернутися до попереднього формату написання змінних використовується функція detach ()

```
> detach(alldata)
> City
Error: object 'City' not found
> alldata$City
[1] Lviv Kyiv Kharkiv <NA>
Levels: Kharkiv Kyiv Lviv
```

Аналогічним чином функції attach () і detach () використовуються для роботи зі змінними списку (list).

Доступ до елементів датафрейму здійснюється вказуючи номер/назву рядка і/або стовпчика

```
>       alldata[4,"ID"]
[1] 104
>       alldata[2:3,]
ID City PostID Check
2       102 Kyiv 90000 FALSE
3       103 Kharkiv 64000 TRUE
>       alldata[,"City"]
[1] Lviv Kyiv Kharkiv <NA> Levels: Kharkiv Kyiv Lviv
>       alldata[["City"]][1]
[1] Lviv
Levels: Kharkiv Kyiv Lviv
```

Список являє собою об’єкт, що за своєю структурою подібний до вектора. Однак на відміну від вектора, де всі елементи належать до одного типу даних, список може містити різні об’єкти з різними типами даних, вклю¬чно з іншими списками і блоками даних (датафреймами). Відповідно спи¬ски використовуються в ситуаціях коли дані близькі за своїм контентом, але неоднорідні за своєю структурою. Списки генеруються за допомогою функції list ()

```
>       xi <- 1:5
>       x2                                                  <-
c(’Ponedilok’,’Vivtorok’,’Chetver’,’Sereda’,’Piatnycia
>       x3 <- c(T,T,F,F,T)
>       y <- list(daynumber=xl, day=x2, order=x3)
>       y
$daynumber
[1] 1 2 3 4 5
$day
[1] "Ponedilok" "Vivtorok" "Chetver" "Sereda" "Piatnycia
$order
[1] TRUE TRUE FALSE FALSE TRUE
```

Функція names () дозволяє отримати імена компонент списку

```
>       names(y)
[1] "daynumber" "day"    "order"
```

Доступ до компонентів списку відбувається: за індексом компоненти або за іменем компоненти. Наприклад:

```
> slobozan <- list( Kharkivskaobl = list(mista =
c('Kharkiv' = 1440676, 'Izum' = 53223, 'Krasnograd' =
27600 , 'Bohoduxiv'
= 17653, 'Zmijiv' = 16976, 'Liubotyn' = 25700,'Balakliia'
= 32117,'Lozova'= 71500), naselenia = 2760948, oblcentr
= 'Kharkiv'), Sumskaobl = list(mista = c('Sumy' =
273984, 'Okhtyrka' = 49431, 'Trostianets' = 23370,
'Konotop' = 93671, 'Shostka' = 80389 ), naselenia =
1164619, oblcentr = 'Sumy'))
> slobozan[[1]] $mista
Kharkiv 1440676 Balakliia 32117
Izum Krasnograd Bohoduxiv
53223 27600 17653
Lozova 71500
Zmijiv Liubotyn
16976 25700
$naselenia [1] 2760948
$oblcentr [1] "Kharkiv"
> slobozan$Kharkivskaobl $mista
Kharkiv Izum Krasnograd Bohoduxiv Zmij iv Liubotyn
1440676 532232760017653 1697625700
Balakliia Lozova 32117 71500
$naselenia [1] 2760948
$oblcentr [1] "Kharkiv"
> slobozan$Sumskaobl$oblcentr [1] "Sumy"
```

Результати більшості статистичних аналізів представляються у вигляді списків.

3.5.5. Експорт/Імпорт даних в R

Записати дані з середовища R в файл можна кількома способами, в залежності від необхідної вихідної структури/формату файлу даних. Основною фунціею є write.tableQ, яка дозволяє зберегти матрицю чи¬сел або датафрейм у вигляді таблиці даних. Загальний синтаксис функції

```
> write.table(x, file = ...)
```

де x – об'єкт, що записується (матриця або датафрейм), file – назва

файлу, в який записуються дані, ... - додаткові аргументи. Детальніше про додаткові аргументи можна знайти за командою ?write. table або help (write, table).

В результаті використання функції write. table () створюється файл з вказаними іменами рядків та стовбчиків (при умові, що імена існували), в якому дані розділені між собою пробілами.

```
> xval<-mtcars[1:10,]
> write.table(xval,file="datal.txt")
```

Файл datal.txt містить десять рядків з набору даних mtcars (див. ?mtcars)

```
"mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" .
"Mazda RX4" 21 6 160 110 3.9 2.62 16.46 0 .
"Mazda RX4 Wag" 21 6 160 110 3.9 2.875 17.02 0 .
"Datsun 710" 22.8 4 108 93 3.85 2.32 18.61 1 .
"Hornet 4 Drive" 21.4 6 258 110 3.08 3.215 19.44 1 .
"Hornet Sportabout" 18.7 8 360 175 3.15 3.44 17.02 0 .
"Valiant" 18.1 6 225 105 2.76 3.46 20.22 1 .
"Duster 360" 14.3 8 360 245 3.21 3.57 15.84 0 .
"Merc 240D" 24.4 4 146.7 62 3.69 3.19 20 1 .
"Merc 230" 22.8 4 140.8 95 3.92 3.15 22.9 1 .
"Merc 280" 19.2 6 167.6 123 3.92 3.44 18.3 1 .
```

Функції write. csv () і write. csv2 () є різновидами функції write. table () з означеними аргументами, використання яких дозволяє зберігати дані у форматі CSV розділених , і ; в першому і другому випадку, відповідно.

```
> write.csv(xval,file="data2.csv")
```

Файл data2.csv має вигляд

```
"","mpg","cyl","disp","hp","drat","wt","qsec","vs" .
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0 .
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,17.02,0 .
"Datsun 710",22.8,4,108,93,3.85,2.32,18.61,1 .
"Hornet 4 Drive",21.4,6,258,110,3.08,3.215,19.44,1 .
"Hornet Sportabout",18.7,8,360,175,3.15,3.44,17.02,0 .
"Valiant",18.1,6,225,105,2.76,3.46,20.22,1 .
"Duster 360",14.3,8,360,245,3.21,3.57,15.84,0 .
"Merc 240D",24.4,4,146.7,62,3.69,3.19,20,1 .
"Merc 230",22.8,4,140.8,95,3.92,3.15,22.9,1 .
"Merc 280",19.2,6,167.6,123,3.92,3.44,18.3,1 .
```

Формат CSV є поширеним форматом збереження даних і

використовується для експорту/імпорту даних між різними програмними середовищами.

За допомогою пакету xlsReadWrite дані з середовища R можна зберегти у форматі MS Office Excel (файл з розширенням *.xls). Перед викликом функції write. xls (), призначеної для збереження даних у форматі Excel в робоче середовище R завантажується пакет xlsReadWrite.

```
> library(xlsReadWrite)
> write.xls(datal, ".../rtoexceldata.xls")
```

Дані з робочого середовища R можна зберегти за допомогою функці sinkQ і функції cat(). Функція sink(file) направляє стандартний вихід всіх команд з екрану терміналу середовища R в файл file.

```
> x <- c(1,3,5,7,9,15,7)
> sink("testdatafile.txt") # Скерування стандартного виходу в файл testdatafile.txt
> print(x)
> cat("The mean of x is",round(mean(x),3))
> sinkQ # Завершення запису в файл і перенаправлення знову в стандартний вихід середовища E
```

На відміну від попередньої функції sinkQ, cat () направляє в файл дані, що явно задані в межах самої функції.

```
> cat("2 3 5 7", "11 13 17 19", file="exportcat.dat", sep="\n")
```

Необхідним навиком роботи в R є введення та імпорт даних. Великі об’єми даних зазвичай імпортуються з файлів, а окремі, дані вводятся безпосередньо з клавіатури. Щоб полегшити роботу з файлами, R містить кілька функцій, які дозволяють безпосередньо взаємодіяти з операційною системою. Наприклад при написанні команди або скрипту зчитування даних з файлу необхідно знати точне ім’я файлу даних. Аби проглянути список об’єктів в папці/каталозі існує фунція list.filesQ

```
> list.filesQ
[1] "data.csv" "myfile.txt" "Rlib" "testscript.R"
```

Кількість об’єктів в даній папці

```
> length(list.files())
[1] 4
```

Функція getwdQ дозволяє отримати повний адрес робочого каталогу. Натомість фунція setwd(". ./Folder") змінює адрес робочого каталогу. У випадку коли файл знаходиться поза робочим каталогом необхідно вказати повний адрес перед самим файлом.

Завантажити дані з текстового файлу (ASCII формат файлів) в R можна за допомогою наступних функцій:

read. table () – імпортує дані у вигляді датафрейму з форматованого текстового файлу;

read, csv() – імпортує датафрейми з текстового файлу, в якому дані відокремлюються символом коми;

read. delimQ – зчитує дані з текстового файлу, в якому дані відокремлені символами табуляції;

read.fwf () – імпортує дані з файлу формат фіксованої ширини;

scanQ – надає можливості низько-рівневого зчитування даних.

Базовою є функція scanQ, яка часто використовується для зчит¬ування файлів великих розмірів. Функцій read.tableQ, read.csvQ, read.fwf () є похідними від базової.

Найбільш зручний спосіб імпортувати текстові дані є використання функції read.tableQ. Синтаксис функції має вигляд

```
> read.table(file,. . . )
```

де file – назва файлу і повний шлях до нього (якщо файл знаходиться за межами робочої директорії), ... – набір аргументів. Деталі ?read.table або help(read.table)

Нехай маємо текстовий файл group.txt, що містить назви змінних і дані

```
"Names" "Age" "Weight,kg" "Height,cm" "Sex" "ID"
"Sofia" 2 12 55 "W" "10001"
"Petro" 40 101 173 "M" "10002"
"Vitalij" 37 90 175 "M" "10003"
"Inna" 18 52 180 "W" "10004"
"Anna" 20 55 170 "W" "10005"
```

Тоді зчитування даних, включаючи іменна змінних, з файлу може

виглядати так

```
> groupdata <- read. tableCgroup.txt" ,header=TRUE) groupdata
   Names      Age Weight.kg Height.cm      Sex    ID
1  Sofia 2    12     55     W      10001
2  Petro 40   101    173    M      10002
3 Vitalij 37  90     175    M      10003
4  Inna  18   52     180    w      10004
5  Anna 20    55     170    w      10005
```

Опції header ставиться значення TRUE для зчитування першого рядка текстового файлу, що містить імена змінних, у якості назв стовпчиків.

Функції read. csv() і read.delimO є більш компактним записом функції read.table() у випадку завантаження даних відокремлених комами і символами табуляції, відповідно. Існують також варіації у вигляді функцій read.csv2() і read.delim2() призначені для зчитування даних з файлів, в яких десяткові числа відокремленні від цілих комами.

Наприклад за допомогою функції read.csv() використовуючи файл data2.csv дані імпортуються в середовище R, наприклад, як датафрейм з іменем datacsv. За допомогою функції print () вміст датафрейму виводиться на екран

```
>     datacsv<-read.csv('data2.csv')
>     print(datacsv)
```

Формат файлів з фіксованою шириною зустрічається доволі рідко, оскільки більшість даних у текстових файлах відокремленні або комою або символом табуляції. Тим не менш такі файли зустрічаються і під час імпорту даних з використанням функції read.fwf() вказується параметер width – тобто вектор, який містить числові значеня кількості символів (ширини) для кожної змінної, що імпортується. Функція read. fwf () створює і записує дані в тимчасовий файл, в якому дані відокремлюються символами табуляції, а безпосередній імпорт даних в середовище R здійснюється функцією read.tableQ.

```
>     dat.ff <- tempfileQ
>     cat(file=dat.ff,"12345678","abcdefgh",sep="\n")
>     read.fwf(dat.ff,width=c(2,4,1,1))
>     ¥1 ¥2 ¥3 ¥4
>     1 12 3456 7 8
>     2 ab cdef g h
```

```
>       unlink(dat.ff) # видаляє файл
```

Функція unlinkQ видаляє вказаний файл або директорію/папку.

Функція scanQ використовується в ситуаціях в яких попередні функції є менш ефективними або неспроможні коректно імпортувати дані. scanQ зчитує дані у вигляді вектора або списку. Синтаксис функції має вигляд

```
>     scan(file = "",...)
```

де file – назва файлу і повний шлях до нього (якщо файл знаходиться за межами робочої директорії), ... – набір аргументів.

```
content <- scan('group.txt',what="character",sep='') Read 36
items > content
[1]     "Names"     "Age"Weight,kg"  "Height,cm""Sex"
  "ID"
[7]     "Sofia"       "2"    "12"   СЛ
сл      "W"   "10001
[13]    "Petro"       "40"   "101" "173"         "10002
[19]    "Vitalij"     "37"   "90"   "175"         "10003
[25]    "Inna"        "18"   "52"   "180" "W"   "10004
[31]    "Anna"        "20"   СЛ
сл      "170" "W"   "10005
```

Аргументи what і sep задають тип даних та символ сепарації даних відповідно. Деталі див ?scan або help (scan).

У випадку коли джерело даних, тобто аргумент file не вказано, введення даних здійснюється інтерактивно (див. стр. 27)

Найкращий спосіб прочитати *.xls файли – зберегти дані в середовищі у вигляді форматованого *.csv файлу та імпортувати його як вищезгаданий csv файл. В Windows системах для доступу до даних файлів Excel, можна скористатися пакетом R0DBC. Перший рядок повинен містити змінні/імена стовбчиків.

```
#     перший рядок містить імена зміних
#     дані зчитуються з worksheet mysheet
>     library(RODBC)
>     channel <- odbcConnectExcel("c:/exelfile.xls")
>     mydata <- sqlFetch(channel, "mysheet")
>     odbcClose(channel)
```

Щоб імпортувати дані з програми SPSS необхідно перед цим в R

завантажити пакет Hmisc

```
#   Збереження даних SPSS в формат який розпізнається R
>   get file=Ic:\dataSPSS.sav1.
>   export outfile='c:\dataSPSS.por'.
#   Завантаження даних в R
>   library(Hmisc)
>   mydata <- spss.get("c:/dataSPSS.por",
use.value.labels=TRUE)
#   остання опція конвертує назви в фактори
```

Блоки даних можна створити інтерактивно вводячи дані безпосередньо з клавіатури. В першому випадку дані вводяться безпосередньо в середовище R за допомогою функції scanQ. Введення чисел

```
>   numbervector<-scan()
>
1: 0
2: 10
3: 20
4: -30
5: 40
6: 50
7
Read 6 items
> numbervector
[1] 0 10 20 -30 40 50
```

Введення даних символьного типу

```
charvector <- scan(what = sep = "\n")
1: Перший рядок 2: Другий 3: Третій
4: Напевно вистачить?
5:
Read 4 items charvector
[1] "Перший рядок"   "Другий"   "Третій"
[1]   "Напевно вистачить?"
```

Пустий рядок (тобто два рази Enter) означає закінчення режиму введення даних.

В другому випадку дані в R заносяться через редактор за допомогою функції edit () або f ix(). Змінні, в які вносяться дані, мусять бути попередньо задекларовані.

```
> country<-c("EU","USA","Japan")
```

```
> currencyquant<-c(100,100,1000)
> currency<-c("EURO","USD","JPY")
> rate<-c(1070.08,793.77,95.12)
> nationaK-repO'UAH" ,3)
> mydata <- data.frame(country,currency,currencyquant,
> rate,national)
> temp <- edit(mydata)
> mydata <-temp # перезапис/оновлення даних об'єкту mydata
```

3.5.6. Функції і конструкції в R

Переважна більшість операцій і задач в R здійснюється за допомогою функцій. Функції являють собою програмний код з певного набору змінних, констант, операторів, інших функцій, що призначені для виконання певної конкретної задачі або операцій і поверненням результату виконання функції. За своїм призначенням функції можна поділити на кілька окремих груп арифметичні, символьні, статистичні та інші, а також вбудовані і власні, тобто ті, що написані безпосередньо самими користувачами.

До вбудованих належать функції, які є складовою частиною програмного середовища R.

Арифметичні функції

```
abs(x) # модуль величини x
ceiling(x) # округлення до цілого в більшу сторону,
#       ceiling(9.435)
#       10
floor(x) # округлення до цілого в меншу сторону,
#       floor(2.975) буде 2
round(x, digits=n) # округлення до вказаного числа digits
#       після коми(крапки),
#       round(5.475, digits=2) буде 5.48 signif(x, digits=n) #
округлення до вказаного числа digits
#       з урахуванням чисел перед комою
#       signif(3.475, digits=2) буде 3.5 trunc(x) # відкиненя
значень після коми (крапки)
#       trunc(4.99) буде 4
```

```
cos(x), sin(x), tan(x), acos(x), cosh(x), acosh(x)
exp(x) #      e~x
log(x) #      логарифм натуральний x
loglO(x) #    логарифм десятковий x
sqrt(x) #     корінь квадратний x
```

Функції для роботи з символьними типами даних

```
grep(pattern, x , ignore.case=FALSE, fixed=FALSE)
#       Пошук значень pattern серед даних в x і повернення
#       значення індексу елементу, що співпав grep("A",
c("x","y","A","z"), fixed=TRUE)
[1] 3
substr(x, start=nl, stop=n2)
#       Вибір або заміна символів вектора символьного типу
substr(x, 2, 4)
[1] "yxx"
substr(x, 2, 4) <- "01100" print(x)
[1] "zOllwt"
paste(..., sep="")
#       Об'єднання символів або рядків
#       після використання символа відокремлення, що
задається
#       аргументом sep=""
paste("x",1:3,sep="")
[1] "xi" "x2" "x3" paste("x",1:3,sep="val")
[1] "xvall" "xval2" "xval3"
strsplit(x, split) # Розділяє елементи вектора
#       x за вказаним split критерієм
strsplit("abc", "")
[[1]]
[1] "a" "b" "c"
strsplit("abcabcab", c("c","a"))
[[1]]
[1] "ab" "ab" "ab"
split(1:10, 1:2)
$'1'
[1] 1 3 5 7 9
$'2'
[1] 2 4 6 8 10 sub(pattern, replacement, x, ignore.case =FALSE,
fixed=FALSE)
#       Заходить pattern в об'єкті x
#       і замінює на значення задене в replacement
sub("\\s","!","Привіт Всім")
[1] "Привіт!Всім"
```

```
toupper(x) # заміна на ВЕЛИКІ ЛІТЕРИ toupper("великі")
[1] "ВЕЛИКІ"
tolower("МАЛІ ЛІТЕРИ") # заміна на малі [1] "малі літери"
```

Для розширення функціоналу програмного середовища в R існує можливість написання власних нових функцій. Загальний синтаксис власної функції має вигляд

```
functionname <- function(argl, arg2,...) {
body # певний функціональний код у вигляді набору команд,
#  зміних, констант, операторів, інших вже існуючих функцій.
return(object)
}
```

де functionname – ім'я функці, argl, arg2,... – вхідні аргументи функції , body – тіло функції, код який виконує функція, object – результат, що повертається в результаті виконання функції. Функції також можуть бути і без вхідних аргументів. Приклад функції, результатом виконання якої є одночасне представлення середнього і значення середьоквадратичного відхилення

```
>  functionAverageDev <- function(x){ aver <- mean(x)
stdev <- sd(x) c(MEAN=aver, SD=stdev)
}
>  functionAverageDev(l:100)
MEAN  SD
50.50000 29.01149
```

Функція functionAverageDevQ викликає дві стандартні функції середовища R : mean() і sd(). Функцію можна записати у скриптовий файл і викликати її для виконання з файлу. Наприклад файл avsqroot.R містить код функції середнього і квадрату середнього значення

```
>  functionAvSqroot <- function(x){
aver <- mean(x) sq <- sqrt(aver) c(MEAN=aver,
SquareRoot=sq)
}
```

Перед викликом функції з скриптового файлу необхідно вказати ім'я файла, де знаходиться функція, за допомогою команди source().

```
>  functionAvSqroot(1:100)
Error: could not find function "functionAvSqroot"
```

```
>       source("avsqroot.R")
>       functionAvSqroot(1:100)
MEAN SquareRoot 50.500000 7.106335
```

Використання скриптового файлу для запису функції є зручним в ситуаціях коли тіло функції містить код значного об'єму.

Синтакс заголовку функції вказує чи аргумент функції є обов'язковим чи може вказуватися опціонально. У випадку якщо обов'язковий аргумент не вказаний, то при виконані функції виникне помилка.

```
>       power <- function(x,n=3){ x~n
}
>       power()
Error in x~n : 1 x' is missing
>       power(2)
[1] 8
```

аргумент *n* – не є обов'язковим, однак може приймати інше значення.

```
>   power(2,6)
[1] 64
```

Аргументами функцій можуть бути також об'єкти будь-якого типу, навіть функції.

```
>   exampl <- function(x, function2)
{
x <- runif(x) function2(x)
}
>   exampl(5,log)
[1] -0.5747896 -0.6354184 -0.8888178 -0.2795405 -0.7150940
```

Останній вираз в тілі функції є результатом виконання функції. В даному випадку ним може бути тільки одне значення об'єкту.

```
>   myf <- function(x,y){ zl <- sin(x)
z2 <- cos(y) zl+z2
}
```

Значення кількох об'єктів можна отримати використовуючи список.

```
>       myf <- function(x,y){ zl <- sin(x)
z2 <- cos(y) list(zl,z2)
}
```

Щоб вийти з функції раніше останьої лінії коду використовується функція return(). Будь-який код після return() ігнорується.

```
>       myf <- function(x,y){ zl <- sin(x)
```

```
z2 <- cos(y) if(zl < 0){
return( list(zl,z2)) else{
return( zl+z2)
}
}
```

Змінні в серединні функції є локальними змінними. Локальні змінні не взаємодіють/перекриваються з об'єктами поза межами функції (глобальни¬ми змінними) і існують тільки в межах самої функції. Якщо змінна знах¬одиться в тілі функції то ця змінна повертається як результат функції

```
>       y <- 0
>       functionY <- function(){
y <- ю
}
>       print (functionYQ )
[1] 10
>       print(y)
[1] o
```

Аргумент . . . використовується для передачі аргументів з однієї функц¬ії в іншу. В якості . . . можуть використовуватися аргументи у вигляді змінних або інших функцій. Наприклад

```
>       product <- function(x,...) { arguments <- list(...)
for (y in arguments) x <- x*y x
}
>       product(10,10,10)
1000
>       plotfunc <- function(upper = pi, ...)
{
x <- seq(0,upper,1 = 100) plot(x,..., type = "1", col="blue")
}
>       plotfunc(tan(x))
```

3.5.7. Статистика

Узагальнююча (описова) статистика даних в R здійснюється за допомогою базових статистичних функцій , представлених нижче.

Базові статистичні функції:

mean(x) – середнє арифметичне значень об’єкту *x*.

sd(x) – середньо-квадратичне відхилення *x*.

var(x) – дисперсія *x*.

median(x) – медіанна *x*.

quantile(x, probs) – квантиль змінної *x*, де *x* – числовий вектор,

probs – числовий вектор ймовірностей.

sum(x) – сума по x min(x) # мінімальне значення *x*.

max(x) – максимальне значення *x*.

range(x) – мінімальне і максимальне з діапазону значень *x*.

Приклади використання базових статистичних функцій:

```
> x <- c(—5:10)
> mean(x)
[1] 2.5
> sd(x)
[1] 4.760952
> var(x)
[1] 22.66667
> median(x)
[1] 2.5
> quantile(x, c(.3,.75)) # 30-Й and 75-й процентилі x 30% 75%
-0.50 6.25
> sum(x)
[1] 40
> min(x)
[1] -5
> max(x)
[1] 10
> range(x)
[1] -5 10
```

Базові статистичні функції використовуються як окремо так і в комбінації з apply-функціями, що дозволяє застосовувати їх до списків або датафреймів. Нехай маємо курси валют: американського долара - USD, еВро - EURO, швейцарського франка - CHF, за період з 06.09.2010 по 10.09.2010 у вигляді датафрейму. Представимо середнє значення курсів валют протягом вибраного тижня з використанням функції sapply ()

```
> EURO <- c(1014.9384, 1018.1017, 1007.821, 1004.1042,1005.5276)
> USD <- c(790.82,790.82,790.82,790.82,790.82)
> CHF <- c(778.148,778.9607,781.075,782.1953,781.9641)
> kursy <- c(EURO,USD,CHF)
> sapply(kursy,mean)
EURO   USD  CHF
1010.0986 790.8200 780.4686
```

Крім того в R існують вбудовані узагальнюючі, так звані, generic функції summary(), fivenumQ призначені для розрахунку і одночасного представлення значень основних статистичних показників. Результатом використання функції summary() є значення максимального і мінімального, середнього статистичного, медіани, 1-го та 3-го квартилів

```
> summary(EURO)
Min. 1st Qu. Median Mean 3rd Qu. Max.
1004 1006 1008 1010 1015 1018
```

Функція fivenumQ є альтернативною generic функцією, повертає значення мінімального, 3-х найважливіших квантилів (1-го, медіани та 3-го квартилів) і максимальне значення.

```
> fivenum(EURO)
[1] 1004.104 1005.528 1007.821 1014.938 1018.102
```

В середовищі R реалізовано набір функцій густини, функції розподілу, квантилів більшості розподілів ймовірностей. Назви майже всіх функцій розподілу ймовірностей наслідують наступну конвенцію запису: складаються

з першої літери відповідної функції

Функція густини ймовірностей d Функції розподілу ймовірностей p Квантильної функції q Генератор випадкових чисел r

і скороченої назви розподілу ймовірностей.

Кілька прикладів поширених на практиці розподілів Функції нормального розподілу

dnorm(x) # Нормальний розподіл або розподіл Гауса

pnorm(q) # (комулятивна) функція розподілу ймовірностей

qnorm(p) # квантиль нормального розподілу

rnorm(n, m=0,sd=l) # генератор п випадкових величин з

нормального розподілу

де аргументи m - середнє арифметичне,

sd - середньо-квадратичне відхилення.

Приклади:

```
>   x <- seq(-10,10,Ьy=.1)
>   yl <- dnorm(x)
>   plot(x,yl)
>   y2 <- pnorm(x)
>   plot(x,y2)
>   y3 <- qnorm(x)
>   plot(x, y3)
>   z <- rnorm(lOOO) # генерація 1000 випадкових чисел
#   з нормального розподілу (ш = 0, sd= 1)
>   hist(z,breaks=50)
```

Функції рівномірного розподілу:

dunif(x, min=0, max=l) – Рівномірний розподіл

punif(q, min=0, max=l) – Функція розподілу

qunif(p, min=0, max=l) – квантиль ріномірного розподілу

runif(n, min=0, max=l) – Генератор випадкових чисел

Приклади:

```
>  x <- seq(-10,10,by=0.01)
>  yl <- dunif(x)
>  plot(x, yl)
>  y2 <- punif(x)
>  plot(x, y2)
>  x <- seq(-l,1,by=0.01)
>  y3 <- qnorm(x)
>  plot(x, y3)
```

```
> z <- runif(lOO)
> hist(z)
```

Функції Log-нормального закону розподілу:

dinorm(x, meanlog = 0, sdlog = 1) – Log-нормальний розподіл.

plnorm(q, meanlog = 0, sdlog1) – функція розподілу.

qlnorm(p, meanlog = 0, sdlog1) – квантиль Log-нормального розподілу.

rlnorm(n, meanlog = 0, sdlog 1) – генератор випадкових чисел.

Приклади:

```
> x <- seq(0,10, by=0.01)
> ylc-dlnorm(x)
> plot(x,yl)
> y2<-plnorm(x)
> plot(x,y2)
> y3<-qlnorm(x)
> plot(x,y3)
> z <- rlnorm(lOO)
> hist(z)
 Histogram of z
8 10
10  15
```

Функції експоненційного (експоненціального) розподілу:

dexp(x, rate =1) – експоненційний (експоненціальний) розподіл.

pexp(q, rate =1) – функція розподілу.

qexp(p, rate =1) – квантиль експоненційного розподілу.

rexp(n, rate =1) – генератор випадкових чисел, розподілених за нормальним законом.

Приклади:

```
> x <- seq(0,10, by=0.01)
> yl <- dexp(x)
> plot(x, yl)
> y2 <- pexp(x)
> plot(x, y2)
> z <- гехр(IOO)
> hist(z)
Histogram of z
```

Інформація про доступні для використання в середовищі R розподіли ймовірностей (що входять до базового пакету stats), реалізованих у відповідності до вищезгаданої конвенції запису функцій розподілу ймовірностей, представленна у наступномуц списку:

Тип розподілу (варіант англ.) Назва в E
Бета (beta)beta
Біноміальний (binomial) binom
Коші-Лоренца (Cauchy) cauchy
$\chi~{2}$ (chi-squared) chisq
Експоненційний (exponential) exp
Фішера (F) f
Гамма (gamma) gamma
Геометричний (geometric) geom
Гіпергеометричний (hypergeometric) hyper
Log-нормальний (log-normal) lnorm
Логістичний (logistic) logis
Нормальний (normal) norm
Пуассона (Poisson) pois
T-розподіл Стьюдента (T) t
Рівномірний (uniform) unif
Вейбула (Weibull) weibull

В R існує функція sample () за допомогою якої, можна генерувати випадкові дані, у вигляді вектора даних, на основі значень іншого вектора. Приклад:

```
>     sample(1:6,15,replace=T)
[1] 364155632111361
```

симулює і відповідно генерує значення 6-граного кубика, в даному випадку після 15-ти ’’віртуальних кидків”. По дефолту функція sample() генерує значення, що неповторюються двічі. Аргумент replace=T вказує, що значення можуть повторюватися.

3.5.8. Регресійний аналіз

Регресія або регресійний аналіз сукупність статистичних методів, що

застосовуються для знаходження, аналізу, моделювання і прогнозування залежностей між змінними. В середовищі R існує широкий набір функцій (lm(),aov(),glm()) для проведення різних видів регресійного аналізу. Кожна з функцій містить ключовий параметр formula - модель, згідно якої аналізуються дані.

Базовою є лінійна регресія, яка моделює лінійну залежність між змінною y і незалежними змінними $x_1,...,x_n$. В загальному випадку (лінійної множинної регресії), залежність між змінними має вигляд

$$y_i = b_0 + b_{i,1}x_{i,1} + b_{i,2}x_{i,2} + ... + b_{i,j}x_{i,j} + \epsilon_i$$

де y_i – змінна, значення якої спостерігається для і-вої обсервації, b_{ij} – коефіцієнти (параметри) регресії, x_{ij} незалежні змінні, предиктори, $\epsilon_1, ..., \epsilon_n$ – залишкові компоненти, незалежні однаково розподілені випадкові величини, інтерпретуються як шум або помилки.

Лінійний регресійний аналіз в R реалізується з використанням функції lm() . Синтаксис функції lm () має вигляд

lm(formula, data, ...)

де formula – символьний опис моделі, data – об’єкт даних (датафрейм), ... – додаткові аргументи.

Параметр formula відіграє важливу роль в R окреслюючи модель, тобто зв’язок між змінними, згідно якої аналізуються flanidata. Достовірність результатів регресійного аналізу залежить від вибраної моделі.

В найпростішому випадку лінійної залежності

$$y_i = b_0 + b_1x_i + \epsilon_i$$

формула моделі має вигляд $y \sim x$.

Наступний приклад демонструє використання базових функцій для проведення лінійного регресійного аналізу статистичних даних longley.

```
>       dani.lm
 <- lm(GNP ~ Population,data = longley)
>       dani.1m
```

```
Call:
lm(formula = GNP ~ Population, data = longley)
Coefficients:
(Intercept) Population -1275.21         14.16
```

Результат використання функції 1m зберігається в даному випадку в об’єкті dani.lm, який являє собою об’єкт класу ’lm’. Щоб вивести повну інформацію про об’єкт dani.lm можна скористатися функцією print.default() .

```
>   print.default(dani.lm)
$coefficients (Intercept) Population -1275.21004 14.16157
$residuals
1947          1948       1949       1950          1951
-14.399443 -3.763893 -21.294247 -11.120025 17.026813
1952         1953
18.127734 10.683026
. . .
1962 554.894 130.081
attr(,"class")
[1] "1m"
```

Тобто об’єкт класу 1m, в даному випадку dani.lm, насправді містить значно більше інформації.

Використовуючи спеціальні, вже згадуванні generic функції, резуль¬тат виконання яких залежить від типу об’єкту даних або класу об’єкту, можна отримати узагальнюючу статистичну інформацію результатів аналізу, спрогнозувати значення, побудувати діагностичні графіки і тд. Приклади деяких generic функцій, основною з яких є функція summary () представлено нижче:

summary(object) – узагальнена статистична інформація обєкту.

coef(object) – коефіцієнти регресії.

resid(object) – залишки.

fitted(object)– допасовані/встановленні значення таблиця

anova(object) – таблиця дисперсійного аналізу.

predict(obj ect) – прогнозовані значення.

plot(object) – створення діагностичних графіків.

aov(object) – AN0VA аналіз.

glm(object)– узагальнений лінійний аналіз.

Функція summary () в випадку застосування її до обєктів класу lm повертає формулу моделі, залишки (residuals), коефіцієнти регресії, середньоквадр- атичне відхилення оцінки регресії, коефіцієнт детермінації R^2, статистику дисперсійного аналізу (F-statistic)

```
>     summary(dani.1m)
Call:
lm(formula = GNP ~ Population, data = longley)
Residuals:
Min IQ Median   3Q     Max
-21.2942 -11.3519 -0.6804 11.2152 18.1277
Coefficients:
Estimate    Std. Error t value Pr(>|t|)
(Intercept) -1275.2100  59.8256 -21.32 4.53e-12 ***
Population 14.1616       0.5086 27.84 1.17e-13 ***
Signif. codes: 0 '***> 0.001 '**’ 0.01 '*’ 0.05 0.1 ‘ 1 1
Residual standard error: 13.7 on 14 degrees of freedom
Multiple R-squared: 0.9823,Adjusted R-squared: 0.981
F-statistic: 775.2 on 1 and 14 DF, p-value: 1.168e-13
```

Один із способів перевірки достовірності вибраного моделю, заданого параметром formula, є представлення результатів аналізу в графічному вигляді. Застосування generic функції plotQ до обектів класу ’lm’

```
>     par(mfrow=c(2,2))
>     plot(dani.lm)
```

дозволяє отримати діагностичні графіки, показані на наступній сторінці.

В R існують функції, що дозволяють оновити або змінити параметри лінійного аналізу. Функція update () використовується для оновлення або зміни моделі лінійного аналізу. Результатом застосування функції update() є об’єкт класу ’lm’. Конструкція ,+Armed.Forces у вигляді параметра

```
>     dani.Im2year <- update(dani.lm, ~.+Year)
```

дозволяє оновити модель лінійного аналізу, з урахуванням доданої змінної Year. Результатом виконання в даному випадку буде об’єкт dani. Im2year

```
>     dani.Im2year Call:
lm(formula = GNP ~ Population + Year, data = longley)
```

Coefficients:

(Intercept)	Population	Year
-34306.363	2.175	17.620

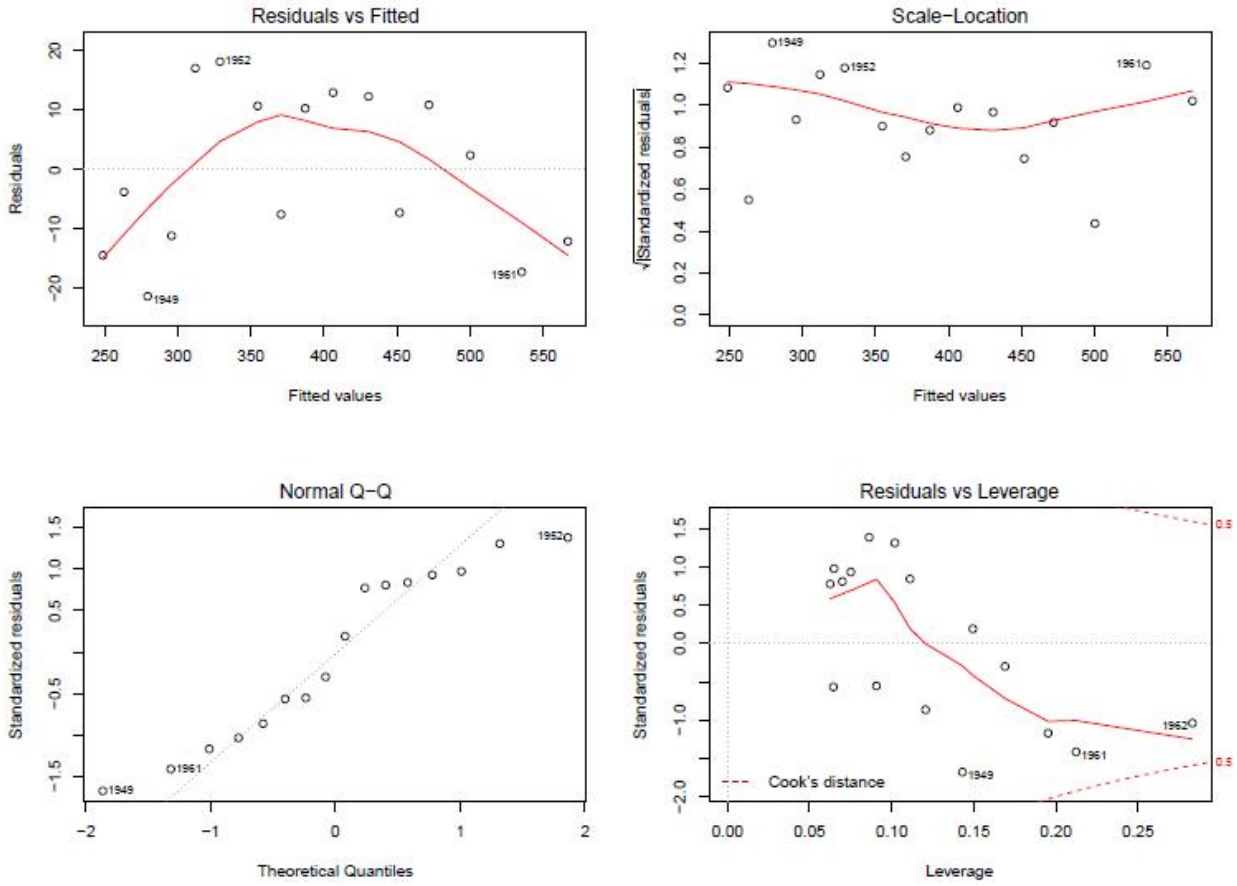


Рис. 3. Результат роботи функції plot() для регресійного аналізу

Функція addl () включає додатковий аргумент (змінну) до аналізованої моделі і дозволяє спостерігати за змінами значень суми квадратів і залишкових сум.

```
> dani.lml <- addl(dani.lm, ~.+Year)
> dani.1ml
Single term additions Model:
GNP ~ Population
     Df Sum of Sq    RSS    AIC
<none>           2629.0  85.628
Year 1 1272.8 1356.2  77.037
```

Функція droplQ дозволяє спостерігати за змінами значень суми квадр¬атів і залишкових сум усуваючи змінні, окрім необхідних, з моделі.

```
> dropl(dani.lm2year,~ Population)
Single term deletions
Model:
GNP ~ Population + Year
           Df Sum of Sq     RSS    AIC
<none>                    1356.2 77.037
Population 1 41.39    1397.5 75.518
```

Функція nls() дозволяє провести нелінійний регресійний аналіз і знайти параметри моделі нелінійної регресії виду

$$y_i = f(x_i, \beta) + \epsilon_i$$

де y_i – змінна, значення якої спостерігається для i-ї обсервації, f – певна нелінійна функція, x_i – незалежні змінні, предиктори, $\beta = (\beta_1, ..., \beta_p)$ – коефіцієнти (параметри) регресії, що розраховуються/допасовуються на основі даних (y_i, x_i), $\epsilon_1, ..., \epsilon_n$ – залишкові компоненти, незалежні однаково розподілені випадкові величини, інтерпретуються як шум або помилки. Синтаксис функції nls() має вигляд

nls(formula, data, start,...)

де formula – формула нелінійної моделі, data – дані у вигляді датафрейму, списку, start – вектор початкових (приблизних) значеннь коефіцієнтів регресії, ... – додаткові аргументи.

На відміну від лінійної регресії, formula моделі нелінійної регресії має звичайну математичну нотацію. Наприклад, для нелінійної моделі

$$y_i = \beta_1(1 - exp^{-b2*x_i}) + \epsilon_i$$

формула моделі матиме вигляд

y ~ b1*(1-exp(-b2*x))

Нехай маємо певну зашумлену нелінійну залежість

```
>   x <- runif(100,0,15)
>   y <- 2*x/(5+x)
>   y <- y +rnorm(100,0,0.15)
>   xy.dani <- data.frame (x=x,y=y)
>   plot(xy.dani)
```

Допасування даних, представлених в графічному вигляді і знаходження

коефіцієнтів нелінійної регресії, можна реалізувати з використанням наступної моделі

$$y_i = \frac{\beta_1 x_1}{\beta_2 x_2} + \beta_3 + \epsilon_i$$

і, наприклад, наступних початкових значень коефіцієнтів регресії 1.5, та 3.

```
> fitnl<-nls(
y ~ betal*x /(beta2 + x),
start = list( betal = 1.5, beta2 = 3), # декларуються початкові
data=xy.da)                    # (приблизні) значення
```

Результатом виконання функції nls() є об'єкт класу 'nls'

```
Nonlinear regression model
model: y ~ betal * x/(beta2 + x)
data: xy.dani
betal    beta2
2.014    4.995
residual sum-of-squares: 2.092

Number of iterations to convergence: 3
Achieved convergence tolerance: 8.634e-07
```

Проглянути результат нелінійного регресійного аналізу в узагальненому структурованому вигляді можна використовуючи generic функцію summary ()

```
>     Formula: y ~ betal * x/(beta2 + x)
Parameters:
Estimate          Std. Error  t          value Pr(>|t|)
betal 2.01433 0.08791 22.914     < 2e-16 ***
beta2 4.99545 0.60934 8.198      9.57e-13 ***
----
Signif. codes: 0 '***> 0.001 '**' 0.01 '*' 0.05 0.1 ' 1 1
Residual standard error: 0.1461 on 98 degrees of freedom

Number of iterations to convergence: 3
Achieved convergence tolerance: 8.634e-07
```

Використовуючи generic функцію predict () на основі результатів проведеного аналізу можна спрогнозувати/обрахувати інші значення і властиві їм похибки.

```
>   x <- seq(0,15,Ьy=0.1)
>   prog.dani <- data.frame(x=x)
>   y.zprog <- predict(fitnl, newdata = prog.dani)
>   prog.dani$yzprog <- y.zprog
>   lines(prog.dani$x,prog.dani$yzprog,cor="blue")
```

3.5.9. Графіки і графічні параметри

Однією з сильних сторін R є широкі можливості представлення даних в графічному вигляді. Середовище R містить два базові пакети graphics та lattice для представлення даних в графічному вигляді. Пакет graphi¬cs містить інтуїтивно зрозумілий набір команд, який дозволяє будувати графіки різних типів, а також редагувати атрибути графіків, в той час як пакет lattice містить альтернативний набір функцій для побудови складних графіків. Даний підрозділ представляє огляд графічних функцій, а також приклади їх використання, які демонструють основні можли¬вості пакету graphics.

Графічні об'єкти в R створюється за допомогою функції plot (). Функція plot () є основною графічною функцією з широким набором аргументів, що дозволяє використовувати її для побудови різних типів графіків. Функція plot () також є generic функцією, тобто результат її використання (у вигляді відповідного графіку або набору графіків) залежить від об'єкту даних до яких застосовується функція.

Загальний вигляд команди

plot(x,y,...)

де *x, y* – змінні (координати змінних) на графіку, ... – додаткові параметри.

Кілька основних параметрів:

- *type* – точковий (*type = 'p'*) встановлюється за замовчуванням, лінійний (*type = 'l'*), точки з'єднані лініями (*type = 'b'*), ступінчастий (*type = 's'*) та інших. Використання функції plot() з параметром (*type = 'n'*). Створює графік з усіма його атрибутами (осями, назвами і т.ін.) не відображаючи при цьому

дані у вигляді точок, ліні і т.п..

- *xlim, ylim* – становлює діапазон значень *x* та *y* осей, відповідно.
- *col* – встановлює колір графіку.
- *log* – дозволяє встановити логаріфмічну шкалу осей (log=’y’, log=’x’, log=’xy’)

Більше інформації можна отримати командою > ?plot

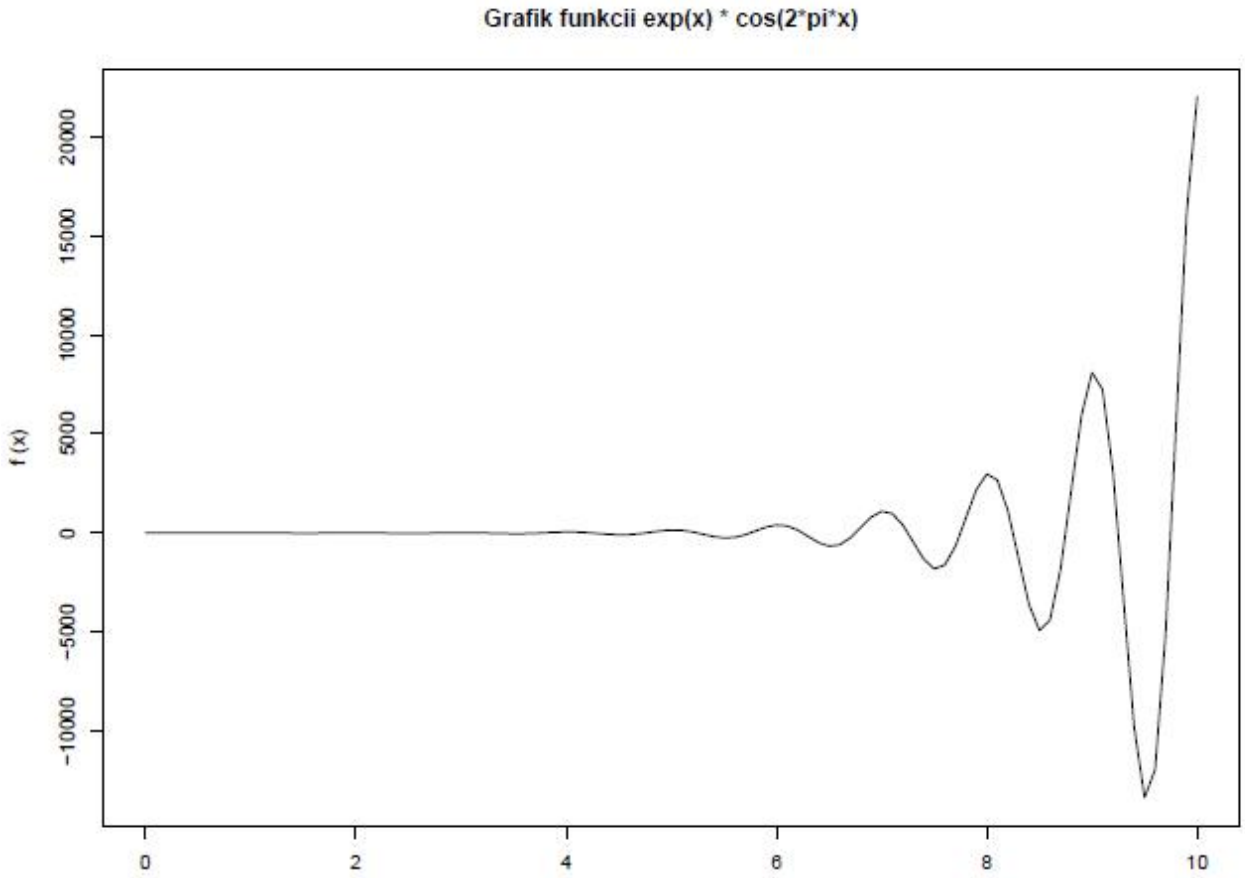
Приклад побудови графіка за допомогою функції plot()

> plot (10~c(1:9) ,ylim=c(10~3,10~9) ,type="b" , colored1 ,log="y")

Функція plot () використовується для побудови графіків на основі даних. Для побудови графіку певної математичної функції або виразу використовується графічна функція curve().

> f <- function(x) exp(x) cos(2*pi*x)

> curve(f, 0, 10, main=1 Grafik funkcii exp(x) * cos(2*pi*x)1)



Лінійні графіки створюються використовуючи базову функцію plotQ з означеним параметром type=’*l*’ або з використанням функції lines (), яка

застосовується до накладання ліній до вже існуючого графіку. Загальний синтаксис функції lines () має вигляд

lines(x,y,...)

де *x,y* – змінні (координати змінних) на графіку,.... – графічні параметри.

Побудова лінійного графіку з використанням функції lines ()

```
> plot(c(2,0),c(-1,2))
> lines(c(0,0,2,0) , c(2,1,-1,1), col="red")
```

Гістограма створюється за допомогою функції hist(),

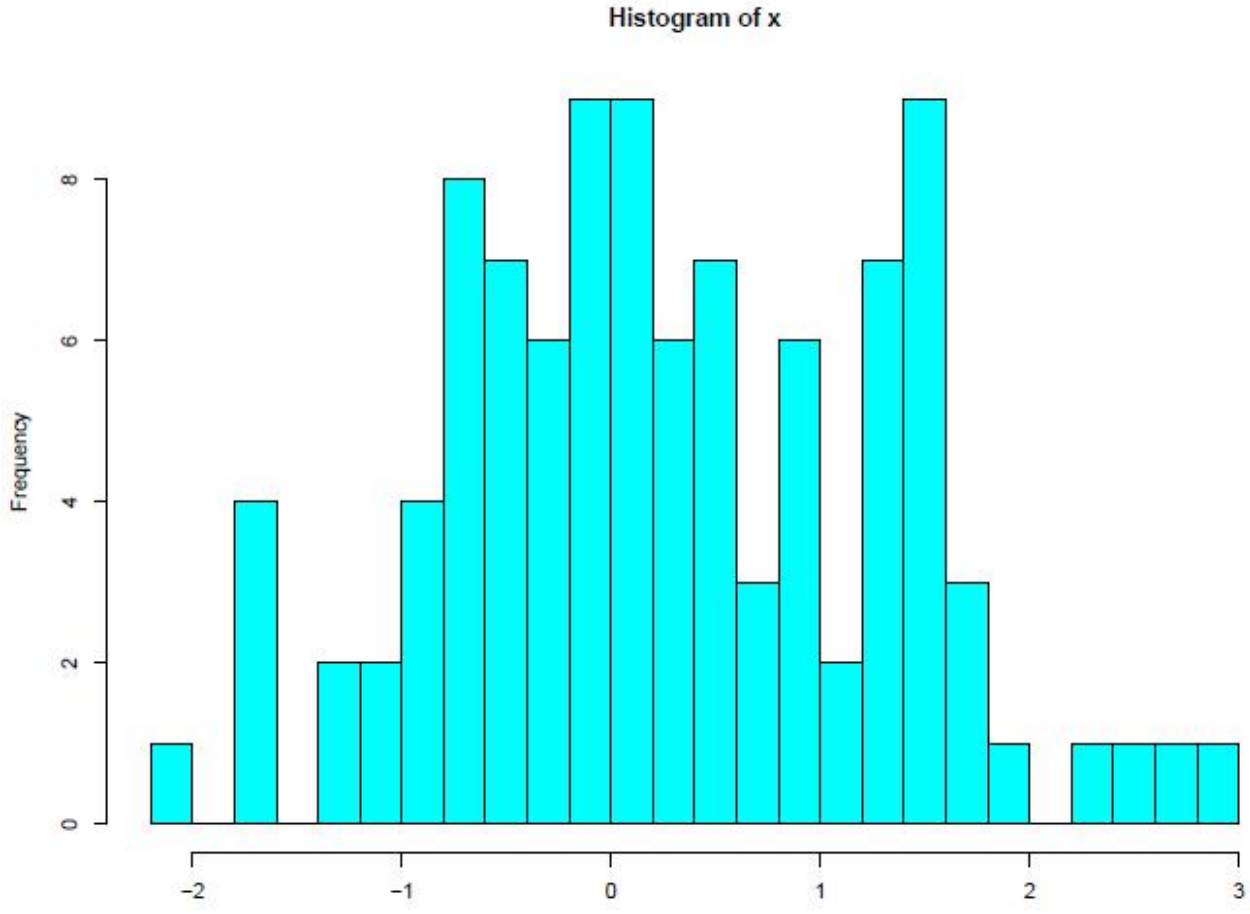
hist(x,...)

де *x* – вектор числових значень, ... – додаткові параметри.

```
> x <- rnorm(lOO)
> hist(x)
```

Кольорова гістограма з заданою кількістю інтервалів отримується окреслюючи додаткові параметри

hist(x, breaks=20, col="cyan")



Для визначення форми розподілу змінної часто використовуються

графіки густини розподілу. В R графік густини розподілу створюється за допомогою команди

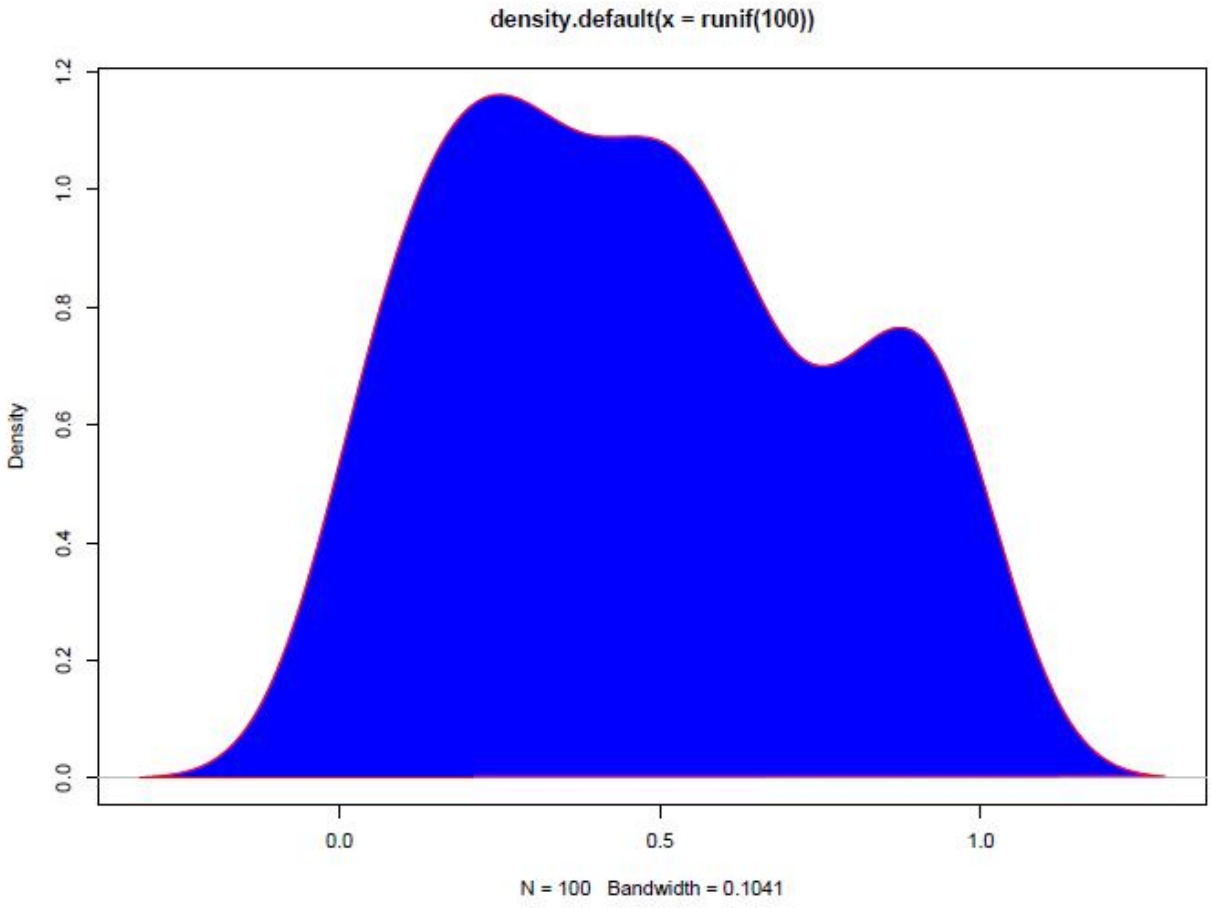
plot(density(x))

де *x* – вектор числових значень. Приклад графіка густини розподілу випадкових величин

```
>       dl <- density(rnorm(100))
>       plot(dl)
```

Приклад графік густини розподілу із кольоровим заповненням

```
> d2 <- density(runif(100))
> plot(d2)
> polygon(d2, col="black", border="red")
```



Q-Q графіки дозволяють порівняти розподіл значень певного набору (емпіричних) даних з вибраним теоретичним розподілом. Якщо обидва розподіли подібні між собою, тоді точки на Q-Q графіку вистроюються у вигляді прямої лінії, якщо розподіли не схожі то точки на графіку

нагадуватимуть криву.

Для побудови Q-Q графіків в R існує функція qqplotQ, qqnormQ і qqline (). Функція qqplot () дозволяє отримати Q-Q графік з двох набор¬ів даних. Синтаксис функції

qqpiot(x, y, …)

Функція qqnormQ створює Q-Q графік, де в якості теоретичного розподілу заданий нормальний розподіл.

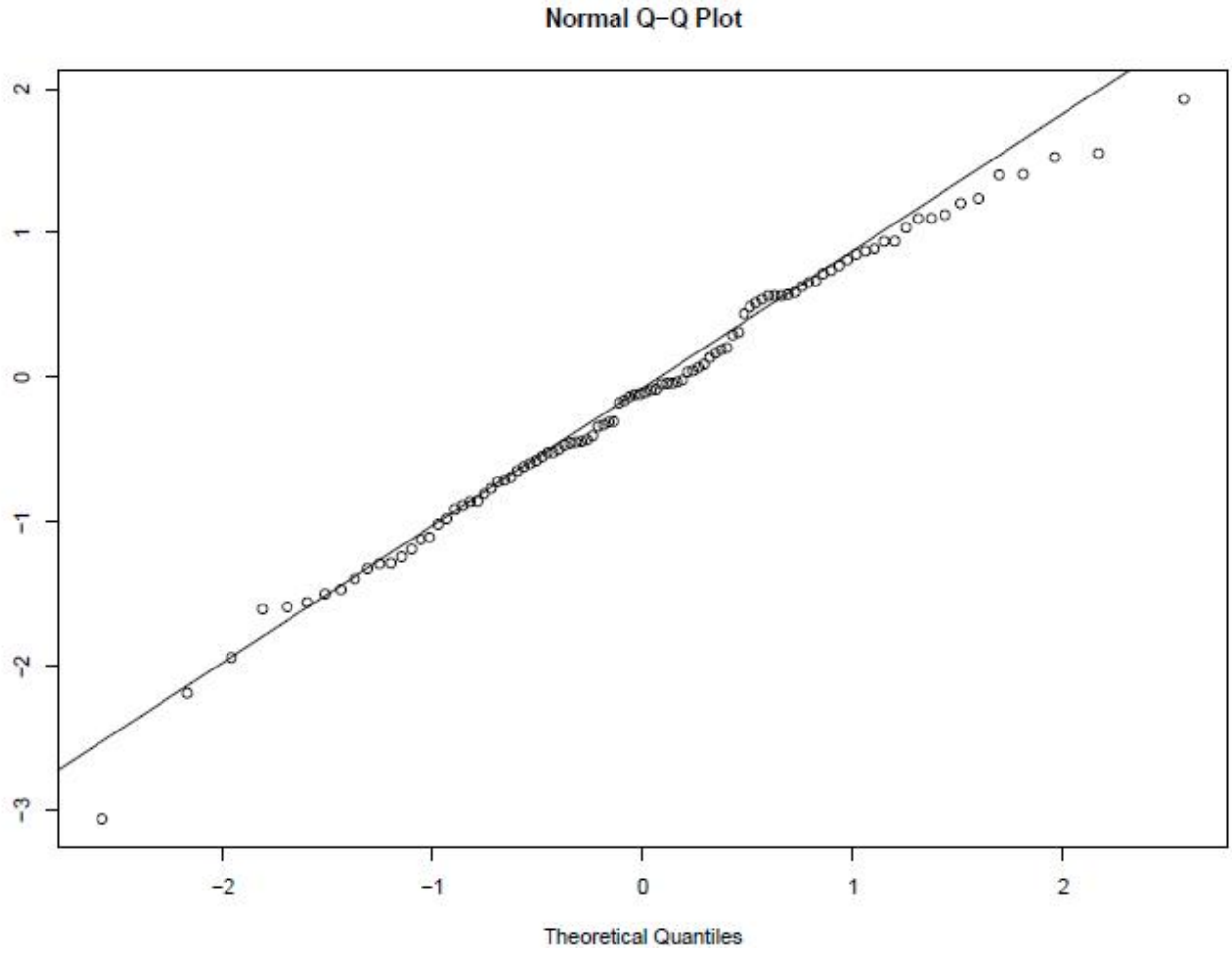
qqnorm(y, ...)

Функція qqline() додає лінію яка проходить через перший (25) і четвертий (75) квартилі (перцентилі) на Q-Q графік

qqline(y, datax = FALSE, ...)

Приклад Q-Q графіку

```
> x<- rnorm(lOO)
> qqnorm(x)
> qqline(x)
```



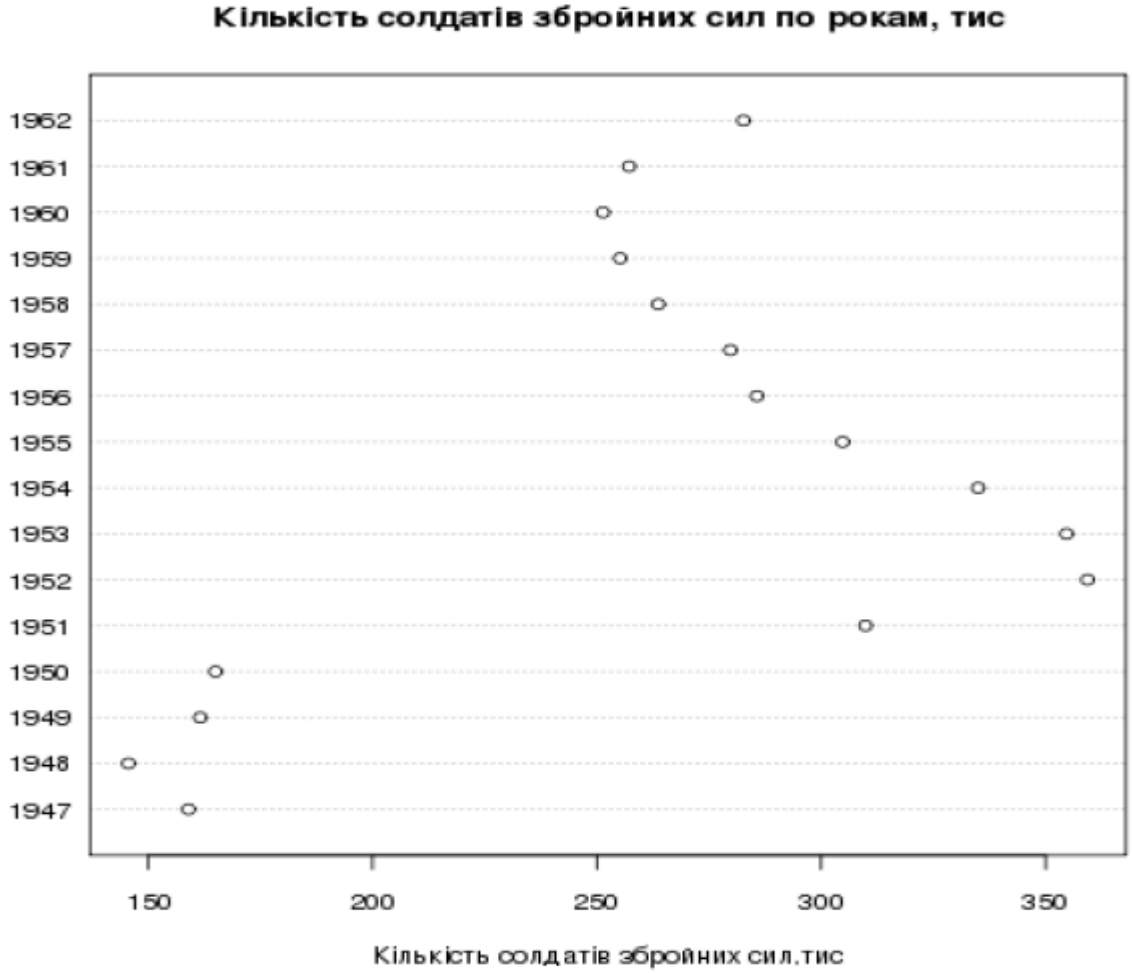
Точкові графіки створюються за допомогою графічної функції dot chart () синтаксис якої має вигляд

dotchart(x, labels, ...)

де *x* – вектор числових значень, labels - вектор міток/назв кожної точки, ... - додаткові параметри. Наприклад, параметр groups дозволяє встановити критерій за яким можна згрупувати елементи вектора *x*. Більше інформації див. help(dotchart). Приклад побудови точкового графіку на основі даних longley, що входить до середовища R

```
>     dotchart(Armed.Forces,label=Year,cex=.9,
main="Кількість солдатів збройних сил по рокам, тис",
x1ab="Кількість солдатів збройних сил,тис")
```

КІЛЬКІСТЬ солдатів збройних сил по рокам, тис



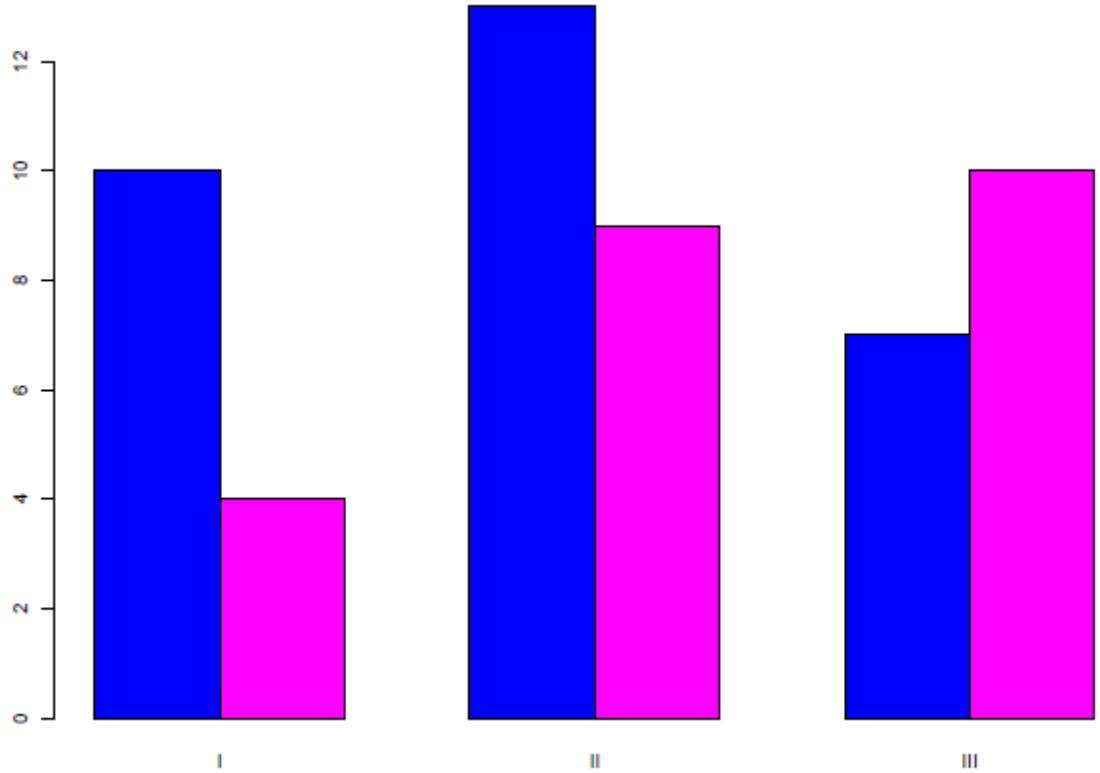
Стовпчикові діаграми будуються за допомогою функції barplotQ

barplot(height,...),

де height – вектор або матриця чисел, ... – додаткові параметри. У випадку вектора, діаграма складається з послідовності прямокутних стовбчиків, причому значення елементів вектора визначають висоту стовпчиків.

У випадку матриці разом з параметром 6eside=TEUE суміжні рівні/колонки матриці відображаються у вигляді окремо згрупованих стовпчиків діагр¬ами. Використання beside=FELSE дозволяє представлення різних рівнів у вигляді підстовпичків складених в один окремий стовпчик.

```
>  a <- c(10, 13, 7)
>  b <- c(4, 9, 10)
>  barplot(rbind(a, b), beside=TRUE, names=c("I","II","III"),
>  + col=c("blue","magenta"))
```



Кругові діаграми в R створюються за допомогою функції pie (x, labels)

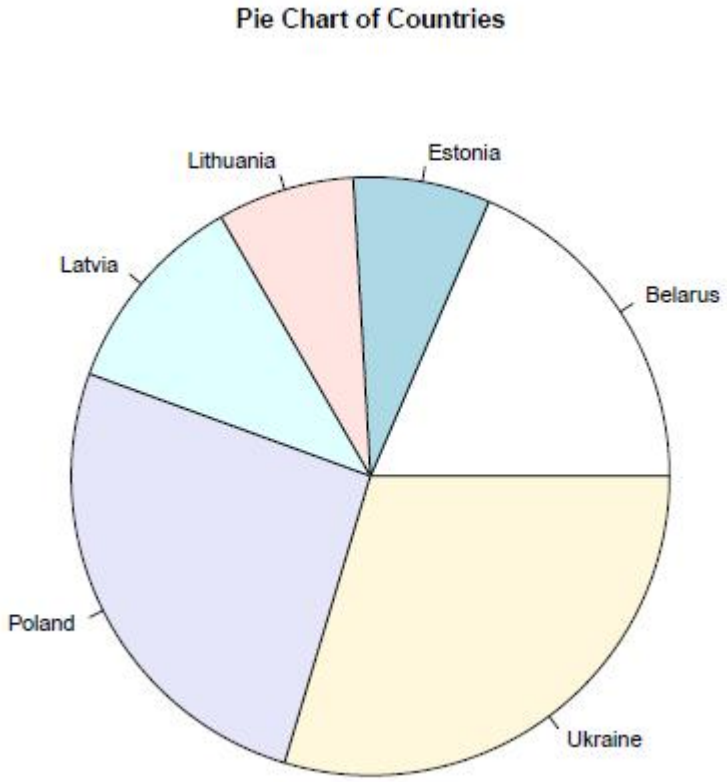
, де x - вектор числових значень (причому x(i)i=0), labels - вектор символьного типу, елементами якого є імена секторів діаграми.

Кілька прикладів побудови кругових діаграм за допомогою функції pie()

Проста кругова діаграма

```
>   slices <- c(10,4,4,6,14,16)
>   Ibis <- c("Belarus","Estonia","Lithuania","Latvia","Poland",
>   "Ukraine")
>   pie(slices, Ibis, main="Pie Chart of Countries")
```

I ithnania Estonia



Кругові діаграми з вказаними процентами

```
>   slices <- c(10, 4,4,6, 14, 16)
>   Ibis <-
c("Belarus","Estonia","Lithuania","Latvia","Poland","Ukraine")
>   pet <- round(slices/sum(slices)*100)
>   Ibis <- paste(Ibis, pet)
>   Ibis <- paste(Ibissep="") # додає % до значень
>   pie(slices,labels = Ibis, col=rainbow(length(Ibis)),
main="Pie Chart of Countries")
```

Існує можливість побудови 3D кругові діаграми підключивши до середовища R пакет plotrix і викликавши функцію pie3D().

Для побудови трьохвимірпих графіків неперервних даних в базовому пакеті graphics передбачені функції image(), contour () і perspQ:

- Функція image () генерує графік у вигляді прямокутників, кольори

яких представляють значення змінної z.

- Функція contour () дозволяє отримати контурний графік, значення змінної z відображають контурні лінії.
- Функція persp () будує трьохвимірні графіки.

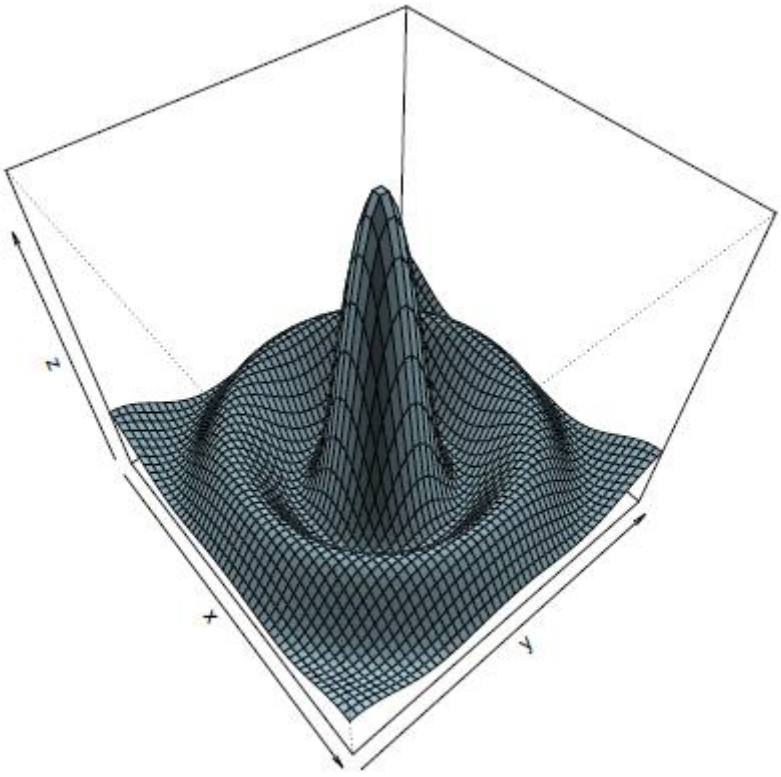
Функція persp() дозволяє представляти 3D графіки з різної перспективи.

Синтаксис функції perspQ має вигляд

persp(x,y,z,...,theta, phi,...)

де *x,y,z* - дані, theta, phi - кути, які задають перспективу (азимутальний і вертикальний напрямки,відповідно). Приклад використання функції persp()

```
x <- seq(-10, 10, length.out = 50) y <- x
rotsinc <- function(x,y) {
sine <- function(x) { y <- sin(x)/x ; y }
10 * sinc( sqrt(x~2+y~2) )
}
z <- outer(x, y, rotsinc)
persp(x, y, z,theta=20, phi=50, shade = 0.3,col = "lightblue")
```



Для запису графіків у файл використовуються наступні функції в залежності від необхідного формату графічного файлу:

1. растровий графічний формат

bmp("myplot.bmp") – файл формату BMP

jpeg("myplot.jpg") – файл формату JPG

png("myplot.png") – файл формату PNG t

iff("myplot.png") – файл формату TIFF

2. векторний графічний формат

postscript("myplot.ps") – файл postscript-ового формату

pdf("myplot.pdf") – файл формату PDF

svg("myplot.svg") – для Unix/Linux

CairoSVG("myplot.svg") – для Windows

win.metaf ile("myplot.wmf")

Приклад запису графіка в файл формату PostScript

```
> postscript("Testplot.ps")
> x<-seq(0,10*pi,by=pi/100)
> plot(x,sin(x))
> dev.off()
```

Функція dev.off () закриває режим запису графіки в файл і зберігає файл.

За допомогою параметрів width і height можна встановити розмір рисунку, що зберігається в файл. Додаткова інформація про параметри див. ?postsript, ?bmp, ?svg.

3.6. Індивідуальне завдання №3

Тема роботи: Вирішення економічних задач, застосовуючи можливості програмних комплексів Maxima та MatLab, а також мов програмування, таких як «C++», «Python» та «R».

Мета роботи: Набуття знань щодо використання програмних комплексів Maxima та MatLab, а також мов програмування, таких як «C++», «Python» та «R» при вирішенні економічних задач.

3.6.1. Maxima

Застосовуючи комплекс Maxima, знайти рішення наступних задач, обираючи їх за номером списку групи.

Завдання А. Обчислити площі фігур, обмежених графіками функцій.

1. $y = (x-2)^3,$ $y = 4x - 8.$

2. $y = x\sqrt{9-x^2},\quad y = 0,$ $(0 \le x \le 3).$

3. $y = 4 - x^2,$ $y = x^2 - 2x.$

4. $y = \sin x \cos^2 x,\quad y = 0,$ $(0 \le x \le \pi/2).$

5. $y = \sqrt{4-x^2},\quad y = 0,$ $x = 0,\quad x = 1.$

6. $y = x^2\sqrt{4-x^2},\quad y = 0,$ $(0 \le x \le 2).$

7. $y = \cos x \sin^2 x,\quad y = 0,$ $(0 \le x \le \pi/2).$

8. $y = \sqrt{e^x - 1},\quad y = 0,$ $x = \ln 2.$

9. $y = \dfrac{1}{x\sqrt{1+\ln x}},\quad y = 0,$ $x = 1,\quad x = e^3.$

10. $y = \arccos x,\quad y = 0,$ $x = 0.$

11. $y=(x+1)^2,$
$y^2=x+1.$

12. $y=2x-x^2+3,$
$y=x^2-4x+3.$

13. $y=x\sqrt{36-x^2},\quad y=0,$
$(0\le x\le 6).$

14. $x=\arccos y,\quad x=0,$
$y=0.$

15. $y=\operatorname{arctg} x,\quad y=0,$
$x=\sqrt{3}.$

16. $y=x^2\sqrt{8-x^2},\quad y=0,$
$\left(0\le x\le 2\sqrt{2}\right).$

17. $x=\sqrt{e^y-1},\quad x=0,$
$y=\ln 2.$

18. $y=x\sqrt{4-x^2},\quad y=0,$
$(0\le x\le 2).$

19. $y=\dfrac{x}{1+\sqrt{x}},\quad y=0,$
$x=1.$

20. $y=\dfrac{1}{1+\cos x},\quad y=0,$
$x=\pi/2,\quad x=-\pi/2.$

21. $x=(y-2)^3,$
$x=4y-8.$

22. $y=\cos^5 x\sin 2x,\quad y=0,$
$(0\le x\le \pi/2).$

23. $y=\dfrac{x}{(x^2+1)^2},\quad y=0,$
$x=1.$

24. $x=4-y^2,$
$x=y^2-2y.$

25. $x=\dfrac{1}{y\sqrt{1+\ln y}},\quad x=0,$
$y=1,\quad y=e^3.$

26. $y=\dfrac{e^{1/x}}{x^2},\quad y=0,$
$x=2,\quad x=1.$

27. $y=x^2\sqrt{16-x^2},\quad y=0,$
$(0\le x\le 4).$

28. $x=\sqrt{4-y^2},\quad x=0,$
$y=0,\quad y=1.$

29. $y=(x-1)^2,$
$y^2=x-1.$

30. $y=x^2\cos x,\quad y=0,$
$(0\le x\le \pi/2).$

31. $\begin{cases} x = 4 - (y-1)^2, \\ x = y^2 - 4y + 3. \end{cases}$

Завдання Б. Знайти загальний інтеграл диференціального рівняння зі змінними, що розділяються. (Відповідь представити у вигляді $\psi(x, y) = C$.)

1. $4xdx - 3ydy = 3x^2ydy - 2xy^2dx.$

2. $x\sqrt{1+y^2} + yy'\sqrt{1+x^2} = 0.$

3. $\sqrt{4+y^2}dx - ydy = x^2ydy.$

4. $\sqrt{3+y^2}dx - ydy = x^2ydy.$

5. $6xdx - 6ydy = 2x^2ydy - 3xy^2dx.$

6. $x\sqrt{3+y^2}dx + y\sqrt{2+x^2}dy = 0.$

7. $\left(e^{2x} + 5\right)dy + y\,e^{2x}\,dx = 0.$

8. $y'y\sqrt{\dfrac{1-x^2}{1-y^2}} + 1 = 0.$

9. $6xdx - 6ydy = 3x^2ydy - 2xy^2dx.$

10. $x\sqrt{5+y^2}dx + y\sqrt{4+x^2}dy = 0.$

11. $y\left(4 + e^x\right)dy - e^x\,dx = 0.$

12. $\sqrt{4-x^2}\,y' + xy^2 + x = 0.$

13. $2xdx - 2ydy = x^2ydy - 2xy^2dx.$

14. $x\sqrt{4+y^2}dx + y\sqrt{1+x^2}dy = 0.$

15. $\left(e^x + 8\right)dy - y\,e^x\,dx = 0.$

16. $\sqrt{5+y^2} + y'y\sqrt{1-x^2} = 0.$

17. $6xdx - ydy = yx^2dy - 3xy^2dx.$

18. $y\ln y + xy' = 0.$

19. $\left(1 + e^x\right)y' = y\,e^x.$

20. $\sqrt{1-x^2}\,y' + xy^2 + x = 0.$

21. $6xdx - 2ydy = 2yx^2dy - 3xy^2dx.$

22. $y\left(1 + \ln y\right) + xy' = 0.$

23. $\left(3 + e^x\right)yy' = e^x.$

24. $\sqrt{3+y^2} + \sqrt{1-x^2}\,yy' = 0.$

25. $xdx - ydy = yx^2dy - xy^2dx.$

26. $\sqrt{5+y^2}dx + 4\left(x^2y + y\right)dy = 0.$

27. $\left(1 + e^x\right)yy' = e^x.$

28. $3\left(x^2y + y\right)dy + \sqrt{2+y^2}dx = 0.$

29. $2xdx - ydy = yx^2dy - xy^2dx.$

30. $2x + 2xy^2 + \sqrt{2-x^2}\,y' = 0.$

31. $20xdx-3ydy=3x^2ydy-5xy^2dx.$

3.6.2. MatLab

Завдання. А. Виконати наступні дії з матрицями, обираючи їх за списком номеру в групі.

1. **2(A+B)(2B–A),** де $A=\begin{bmatrix}2&3&-1\\4&5&2\\-1&0&7\end{bmatrix}$, $B=\begin{bmatrix}-1&0&5\\0&1&3\\2&-2&4\end{bmatrix}$.
2. **3A–(A+B)B**, де $A=\begin{bmatrix}4&5&-2\\3&-1&0\\4&2&7\end{bmatrix}$, $B=\begin{bmatrix}2&1&-1\\0&1&3\\5&7&3\end{bmatrix}$.
3. **2(A–B)(A²+B)**, де $A=\begin{bmatrix}5&1&7\\-10&-2&1\\0&1&2\end{bmatrix}$, $B=\begin{bmatrix}2&4&1\\3&1&0\\7&2&1\end{bmatrix}$.
4. **(A²–B²)(A+B)**, де $A=\begin{bmatrix}7&2&0\\-7&-2&1\\1&1&1\end{bmatrix}$, $B=\begin{bmatrix}0&2&3\\1&0&-2\\3&1&1\end{bmatrix}$.
5. **(A–B²)(2A+B)**, де $A=\begin{bmatrix}5&2&0\\10&4&1\\7&3&2\end{bmatrix}$, $B=\begin{bmatrix}3&6&-1\\-1&-2&0\\2&1&3\end{bmatrix}$.
6. **(A–B)A+2B**, де $A=\begin{bmatrix}5&-1&3\\0&2&-1\\-2&-1&0\end{bmatrix}$, $B=\begin{bmatrix}3&7&-2\\1&1&-2\\0&1&3\end{bmatrix}$.
7. **2(A–0,5B)+AB**, де $A=\begin{bmatrix}5&3&-1\\2&0&4\\3&5&-1\end{bmatrix}$, $B=\begin{bmatrix}2&4&16\\-4&-2&0\\6&8&2\end{bmatrix}$.
8. **(A–B)A+3B**, де $A=\begin{bmatrix}3&2&-5\\4&2&0\\1&1&2\end{bmatrix}$, $B=\begin{bmatrix}-1&2&4\\0&3&2\\-1&-3&4\end{bmatrix}$.
9. **2A–(A²+B)B**, де $A=\begin{bmatrix}1&4&2\\2&1&-2\\0&1&-1\end{bmatrix}$, $B=\begin{bmatrix}4&6&-2\\4&10&1\\2&4&-5\end{bmatrix}$.

10. **$3(A^2-B^2)-2AB^T$**, де $A=\begin{bmatrix}4&2&1\\3&-2&0\\0&-1&2\end{bmatrix}$, $B=\begin{bmatrix}2&0&2\\5&-7&-2\\1&0&-1\end{bmatrix}$.

11. **$(2A-B)(3A+B)-2AB$**, де $A=\begin{bmatrix}1&0&3\\-2&0&1\\-1&3&1\end{bmatrix}$, $B=\begin{bmatrix}7&5&2\\0&1&2\\-3&-1&-1\end{bmatrix}$.

12. **$A(A-B^2)-2(B^T+A)B$**, де $A=\begin{bmatrix}2&3&1\\-1&2&4\\5&3&0\end{bmatrix}$, $B=\begin{bmatrix}2&7&13\\-1&0&5\\5&13&21\end{bmatrix}$.

13. **$(A+B)A-B(2A+3B)$**, де $A=\begin{bmatrix}1&-2&3\\2&3&5\\1&4&-1\end{bmatrix}$, $B=\begin{bmatrix}4&11&3\\1&6&1\\2&2&16\end{bmatrix}$.

14. **$A(2A-B)-B(A-B)$**, де $A=\begin{bmatrix}2&3&1\\4&-1&0\\0&1&2\end{bmatrix}$, $B=\begin{bmatrix}9&8&7\\2&7&3\\4&3&5\end{bmatrix}$.

15. **$3(A+B)(AB^T-2A)$**, де $A=\begin{bmatrix}2&1&3\\1&-2&0\\4&-3&0\end{bmatrix}$, $B=\begin{bmatrix}1&2&3\\-3&0&1\\2&1&2\end{bmatrix}$.

16. **$2AB-(A+B)(A-B)$**, де $A=\begin{bmatrix}4&-2&0\\1&1&2\\3&-2&0\end{bmatrix}$, $B=\begin{bmatrix}0&-2&4\\1&2&3\\3&4&-5\end{bmatrix}$.

17. **$2A+3B(AB-2A)$**, де $A=\begin{bmatrix}1&-1&0\\2&0&-1\\7&3&2\end{bmatrix}$, $B=\begin{bmatrix}5&3&1\\-1&2&0\\-3&0&0\end{bmatrix}$.

18. **$(A-B)(A+B)-2A^TB$**, де $A=\begin{bmatrix}3&2&-4\\-1&0&2\\-2&1&-1\end{bmatrix}$, $B=\begin{bmatrix}0&1&-2\\-1&1&2\\3&-1&0\end{bmatrix}$.

19. **$2A-AB(B-A)+B$**, де $A=\begin{bmatrix}3&2&-1\\0&-1&2\\5&7&1\end{bmatrix}$, $B=\begin{bmatrix}0&3&-1\\2&-1&2\\-3&1&4\end{bmatrix}$.

20. **$A^2-(A+B)(A-3B)$**, де $A=\begin{bmatrix}4&5&6\\-1&0&3\\-1&2&-1\end{bmatrix}$, $B=\begin{bmatrix}0&-1&1\\1&0&-2\\3&1&2\end{bmatrix}$.

21. **$B(A+2B)-3AB$**, де $A=\begin{bmatrix}7&-3&0\\1&-1&0\\2&0&3\end{bmatrix}$, $B=\begin{bmatrix}-4&2&1\\1&0&1\\3&2&1\end{bmatrix}$.

22. **3(A+B)–(A–B)A**, де $A=\begin{bmatrix}1 & 2 & 3\\ 0 & -2 & 3\\ 1 & 1 & 1\end{bmatrix}$, $B=\begin{bmatrix}4 & 2 & 1\\ -1 & 2 & 0\\ 2 & 3 & -1\end{bmatrix}$.

23. **A(A–B)+2B(A+B)**, де $A=\begin{bmatrix}1 & -2 & -2\\ 1 & 1 & -2\\ 1 & -1 & -1\end{bmatrix}$, $B=\begin{bmatrix}0 & 3 & 5\\ 4 & 1 & 0\\ 1 & 1 & 2\end{bmatrix}$.

24. **(2A+B)B–3B**, де $A=\begin{bmatrix}1 & -1 & 2\\ 3 & 0 & -2\\ 2 & -1 & 1\end{bmatrix}$, $B=\begin{bmatrix}-1 & 0 & 2\\ 2 & 1 & 1\\ -2 & 0 & 1\end{bmatrix}$.

25. **AB–2(A^T+B)A**, де $A=\begin{bmatrix}2 & 1 & -1\\ 1 & 0 & 1\\ 3 & 1 & -2\end{bmatrix}$, $B=\begin{bmatrix}2 & -1 & 0\\ 0 & 2 & 1\\ 1 & 3 & -1\end{bmatrix}$.

26. **(A+2B)(3A–B)**, де $A=\begin{bmatrix}1 & 2 & 3\\ 4 & -2 & 1\\ 0 & 1 & -1\end{bmatrix}$, $B=\begin{bmatrix}2 & 3 & -1\\ -2 & 0 & -1\\ 1 & 0 & 1\end{bmatrix}$.

27. **2A^TB–A(B^T–A)**, де $A=\begin{bmatrix}1 & 2 & -1\\ 2 & 3 & 0\\ 0 & 2 & -1\end{bmatrix}$, $B=\begin{bmatrix}1 & 2 & -1\\ 2 & -1 & 0\\ 1 & 2 & 1\end{bmatrix}$.

28. **3(A+B)(2B–A)**, де $A=\begin{bmatrix}1 & 2 & 3\\ -1 & 0 & 2\\ 1 & 2 & 1\end{bmatrix}$, $B=\begin{bmatrix}1 & 0 & 2\\ 2 & 3 & 1\\ 3 & 1 & 0\end{bmatrix}$.

29. **2A(A+B)–3AB**, де $A=\begin{bmatrix}2 & 3 & 4\\ 1 & -2 & 0\\ 0 & 1 & 2\end{bmatrix}$, $B=\begin{bmatrix}2 & 0 & -2\\ 1 & 1 & 0\\ 1 & -1 & 1\end{bmatrix}$.

30. **3AB+(A–B)(A+2B)**, де $A=\begin{bmatrix}2 & 5 & -1\\ 0 & 2 & 1\\ 1 & 0 & 1\end{bmatrix}$, $B=\begin{bmatrix}1 & -2 & 0\\ 1 & 0 & 2\\ 0 & 0 & 3\end{bmatrix}$.

31. **2(A–B)+(A+B)A**, де $A=\begin{bmatrix}1 & 2 & 3\\ -2 & 0 & 5\\ 1 & 2 & -1\end{bmatrix}$, $B=\begin{bmatrix}2 & -2 & 4\\ 0 & 5 & -3\\ 1 & 1 & -1\end{bmatrix}$.

Завдання Б. Варіант обирається згідно останньої цифри номеру залікової книжки.

Написати файл-функцію, яка:

0. Буде ділити кожен стовпець матриці на величину середньоквадратичного відхилення цього стовпця.
1. З кожного рядка матриці віднімати середнє цього рядка.
2. Множити елементи матриці на суму елементів її головної діагоналі.
3. Знаходити визначник зворотної матриці.
4. Розділяти матрицю на дві частини для побудови графіку, з яких перша частина буде аргументам, а друга – функцією.
5. З наданої прямокутної матриці будувати квадратну за значенням меншого розміру по вертикалі чи по горизонталі.
6. Вставляти у задану матрицю одиниці в головну діагональ.
7. Вставляти у задану матрицю нулі в допоміжну діагональ.
8. Попарно складати числа по рядках у заданій матриці, утворюючі нову.
9. Транспонувати задану матрицю, розфарбувавши її стовпці у різні кольори.

3.6.3. C#

Завдання А. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Дано довжину ребра куба. Знайти його об'єм та площу всієї поверхні.

1. Дано два дійсні числа. Знайти їх середнє арифметичне і середнє геометричне значення.

2. Дано катети прямокутного трикутника. Знайти його гіпотенузу і площу.

3. Визначити час падіння каменю на поверхню. Землі з висоти *h.*

4. Трикутник задано координатами вершин. Знайти: 1) периметр трикутника; 2) площу трикутника.

5. Визначити, яку роботу необхідно виконати, щоб підняти тіло масою *m* на висоту *h* від поверхні Землі.

6. Визначити значення опору *R* в колі електричного струму за заданими значеннями сили струму *I* і напруги *U (R = U/I)*

7. Визначити, яку платню одержить на фірмі сумісник за виконану роботу, якщо йому нараховано S гривень, а податок становить 20%

8. Підприємство поклало в банк на депозитний рахунок суму в *S* тисяч гривень під 40 % річних Яку суму зніме підприємство в кінці року?

9. Визначити опір електричного кола, якщо в ньому резистори R_1, R_2, R_3, R_4 з'єднані:

1) послідовно; 2) паралельно

Завдання В. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Описати клас tovar (товар) з полями price (ціна), weight (вага), date (дата випуску), kind (їжа, канцтовари, будівельний, запчастини тощо). Описати метод-конструктор для введення даних властивостей з клавіатури.

1. Описати клас computers (комп'ютери) з полями price (ціна), processor (тип процесора), hddSize (об'єм жорсткого диска), ramSize (об'єм оперативної пам'яті), frequency (тактова частота). Описати метод-конструктор для введення даних властивостей з клавіатури.

2. Описати клас student (студент) з полями firstName (ім'я), lastName (прізвище), faculty (факультет), yearIn (рік вступу), curs (курс). Описати метод-конструктор для введення даних властивостей з клавіатури.

3. Описати клас company (фірма) з полями name (назва), year (рік заснування), rem (короткий опис), budget (уставний капітал). Описати метод-конструктор для введення даних властивостей з клавіатури.

4. Описати клас book (книга) з полями price (ціна), size (кількість сторінок), date (дата видання), kind (жанр), index (шифр, наприклад, К32-4 тощо) Описати метод-конструктор для введення даних властивостей з клавіатури

5. Описати клас employer (робітник) з полями firstName (ім'я), lastName (прізвище), nameTceh (назва цеху де працює), birthDay (дата народження), stag (стаж). Описати метод-для виводу значень усіх полів на консоль.

6. Описати клас car (легковий автомобіль) з полями price (ціна), power (потужність), date (дата випуску), kind (Mersedes, ВАЗ, Lanos, Opel тощо), maxHuman (пасажиромісткість). Описати метод-для виводу значень усіх полів на консоль.

7. Описати клас weapon (зброя) з полями kind (рушниця, пістолет, автомат, кулемет тощо), weight (маса), date (дата випуску), speed (початкова швидкість польоту кулі), mark (марка, наприклад, Макаров, ТТ, АКМ-47, Максим). Описати метод-для виводу значень усіх полів на консоль.

8. Описати клас plane (літак) з полями name (марка, наприклад, АН-24, МИГ-21, Ту-144), length (довжина), width (ширина), power (потужність), date (дата випуску), kind (гвинтовий, турбореактивний), enginCount (кількість двигунів), maxSpeed (максимальна швидкість), top (максимальна висота польоту). Описати метод-для виводу значень усіх полів на консоль.

9. Описати клас truck (вантажний автомобіль) з полями price (ціна), power (потужність), date (дата випуску), kind (Mersedes, КрАЗ, Comatsu тощо), maxCargo (вантажопідйомність).

Завдання С. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Обчисліть значення виразу: x>y&&x= =2, де x=2, y=4.

1. Обчисліть значення виразу: x= =y&&x!=2, де x=12, y=5.

2. Обчисліть значення виразу: k>v||k<=2, де k=2, v=4.

3. Записати умову, за якою лише одне з A, B та C буде більше за 50.

4. Записати умову, за якою лише одне з A, B та C буде меншим за 0.

5. Записати умову яка буде істинною, коли хоча б одне з X, Y чи Z буде кратним 5.

6. Записати умову, за якою число S буде кратним 5 або 3.

7. На шаховому полі з координатами (a,b) знаходиться кінь. Записати умову, коли він погрожуватиме фігурі на полі з координатою (c,d).

8. На шаховому полі з координатами (a,b) знаходиться кінь. Записати умову, коли він не погрожуватиме фігурі на полі з координатою (c,d).

9. Записати умову, за якою число M не буде кратним 4 і буде кратним 3.

Завдання D. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Дано два числа Якщо квадратний корінь другого числа більший за перше число, то збільшити його на п’ять.

1. Дано три числа Вивести на екран ті з них, які є парними.

2. Дано чотири числа. Визначити скільки з них є від’ємними.

3. Дано чотири дійсні числа. Знайти суму тих з них, які є більші за 5.

4. Визначити мінімальне та максимальне з трьох уведених з клавіатури чисел.

5. Дано коефіцієнти a, b та c квадратного рівняння. Визначити чи має дане рівняння дійсні корені.

6. Дано точку з координатами (x,y) в декартовій системі координат. Знаючи, що ні x ні y не дорівнюють нуль, визначити в якій чверті знаходиться точка.

7. Скласти програму для знаходження двох найбільших з трьох уведених з клавіатури чисел.

8. Скласти програму, яка в залежності від номера уведеного для в тижні визначить його назву (понеділок, вівторок, …, неділя) .

9. З початку 2010 року пройшло n місяців та 2 дні Визначити назву поточного місяця (січень, лютий тощо).

Завдання D. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Надрукувати в стовпчик всі цілі числа від 15 до 45.

1. Надрукувати в стовпчик всі числа від 35 до 5.

2. Надрукувати таблицю переводу дюймів у сантиметри від 1 до 20. 1 дюйм=25,4 мм.

3. Надрукувати таблицю Піфагора від 1 до 15.

001 002 003 004 005 006 007 008 009 010 011 012 013 014 015

002 004 006 008 010 012 014 016 018 020 022 024 026 028 030

003 006 009 012 015 018 021 024 027 030 033 036 039 042 045

004 008 012 016 020 024 028 032 036 040 044 048 052 056 060

005 010 015 020 025 030 035 040 045 050 055 060 065 070 075

006 012 018 024 030 036 042 048 054 060 066 072 078 084 090

007 014 021 028 035 042 049 056 063 070 077 084 091 098 105

008 016 024 032 040 048 056 064 072 080 088 096 104 112 120

009 018 027 036 045 054 063 072 081 090 099 108 117 126 135

010 020 030 040 050 060 070 080 090 100 110 120 130 140 150

011 022 033 044 055 066 077 088 099 110 121 132 143 154 165

012 024 036 048 060 072 084 096 108 120 132 144 156 168 180

013 026 039 052 065 078 091 104 117 130 143 156 169 182 195

014 028 042 056 070 084 098 112 126 140 154 168 182 196 210

015 030 045 060 075 090 105 120 135 150 165 180 195 210 225

4. Надрукувати таблицю множення на 7.

7x1=7

7x2=14

…

7x10=70

5. Надрукувати в стовпчик значення: *sin* 0*, sin* 1*, sin* 2*, …, sin* 90 з точністю до десятитисячних Кут вимірювати в градусах

6. Надрукувати в стовпчик значення коренів квадратних з ряду.

$$0{,}1,\ 0{,}2\ ,\ \ldots,\ 1\ .$$

7. Не використовуючи функцію піднесення до степені знайти значення x^y, де x – дійсне, а y– натуральне .

8. Вирахувати суму: $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$.

9. Надрукувати в стовпчик наступні числа:

1,1

1,2

1,3

…

1,8

Завдання Е. Варіант обирається згідно останньої цифри номеру залікової книжки:

0. Дано одновимірний масив. Скласти програму для визначення квадратного кореня з кожного елемента масиву.

1. Дано одновимірний масив. Збільште кожен його елемент удвічі.

2. У заданому двовимірному масиві відсортуйте елементи кожного стовпця за зростанням.

3. У заданому двовимірному масиві відсортуйте елементи кожного рядка за зростанням.

4. Дано двовимірний масив. Вивести на консоль значення правого нижнього елемента.

5. Дано квадратний двовимірний масив. Вивести на консоль сему елементів головної діагоналі масиву.

6. Дано квадратний двовимірний масив. Вивести на консоль сему елементів побічної діагоналі масиву.

7. Заповнити масив розмірністю [6,5] добутком номера стовпця на подвоєний квадратний корінь номера рядка.

8. З двовимірного масиву надрукувати на екрані в стовпчик найбільший елемент кожного рядка.

9. Дано двовимірний масив. Вивести на екран суму всіх елементів третього рядка.

3.6.4. Python

Із сайту Державного управління статистики http://www.ukrstat.gov.ua/ за допомогою команд Pyton:

1. Вибрати статистичну інформацію згідно номеру студента за списком групи за якнайбільший період. Конкретну таблицю в групі даних обирати довільно.

2. Зберегти цю інформацію на диску у файлі.

3. Розрахувати середнє, дисперсію та стандарт для кожної колонки таблиці.

4. Розрахувати кореляційну матрицю зв'язку усіх колонок таблиці.

5. Представити кореляційну матрицю у вигляді графіку з точками.

№ за списком групи	Статистична інформація
1.	Ринок праці
2.	Освіта
3.	Охорона здоров'я
4.	Доходи та умови життя
5.	Соціальний захист
6.	Населені пункти та житло
7.	Правосуддя та злочинність
8.	Культура
9.	Макроекономічна статистика
10.	Національні рахунки
11.	Економічна діяльність
12.	Діяльність підприємств
13.	Внутрішня торгівля
14.	Капітальні інвестиції
15.	Основні засоби
16.	Сільське, лісове та рибне господарство
17.	Енергетика
18.	Промисловість
19.	Будівництво
20.	Транспорт
21.	Туризм

№ за списком групи	Статистична інформація
22.	Реєстр статистичних одиниць
23.	Навколишнє середовище
24.	Державні фінанси, податки та публічний сектор
25.	Зовнішньоекономічна діяльність
26.	Ціни
27.	Наука, технології та інновації
28.	Інформаційне суспільство

3.6.5. R (Ap)

Скориставшись даними з завдання у п. 3.6.4 та вибравши з них таку пару колонок, що має найбільший коефіцієнт кореляції, побудувати нелінійну залежність виду $y = \frac{Ae^x}{(B - x^{-3})} + CSinx$.

Підбір чисельних значень коефіцієнтів *A, B, C* здійснювати, використовуючи функцію nls() мови R (Ap).

Контрольні запитання

1. Для чого призначена програма Maxima?
2. Як отримати перетворення формули в загальному вигляді у програмі Maxima?
3. Поясніть принцип отримання тривимірних графіків у програмі Maxima?
4. Чи потрібно знати математику, щоб брати інтеграли за допомогою програми Maxima? Дати розгорнуту відповідь.
5. Як інсталювати програму MatLab?
6. Чи обмежуються можливості MatLab тільки операціями з матрицями?
7. Як виглядає програма на MatLab?

8. Порівняйте зручність створення графіків на MatLab з Maxima.
9. Для чого призначений пакет Visual Studio Express?
10. Які типи даних містить пакет Visual Studio Express?
11. Поясніть принцип доступу до масивів у пакеті Visual Studio Express.
12. Що таке цикли в програмі? Поясніть на прикладі команд пакету Visual Studio Express.
13. Чим тримісний вираз if / else у пакеті Pyton відрізняється від звичайної конструкції if / else?
14. Як працювати з бібліотеками у пакеті Pyton?
15. Чим множини відрізняються від словників у пакеті Pyton?
16. Яка бібліотека дозволяє робити статистичні обчислення у пакеті Pyton?
17. Як вибирати дані зі сторінок в Інтернеті, написаних у кодах XML і HTMLза допомогою команд пакету Pyton?
18. Чим статистична обробка у пакеті Pyton відрізняється від такої у пакеті R?
19. Поясніть принципи отримання результатів регресійного аналізу у пакеті R.
20. Порівняйте зручність створення зручність створення графіків у R з Maxima.
21. Як записувати результати розрахунків на диск у пакеті R?

В розділі розглянуто можливість застосування програмних пакетів Maxima, MatLab, Visual Studio Express, Python та R для збору даних та розрахунків статистичних характеристик економічної інформації, доступ до якої забезпечує Інтернет, та забезпечення діяльності в галузі електронної комерції.