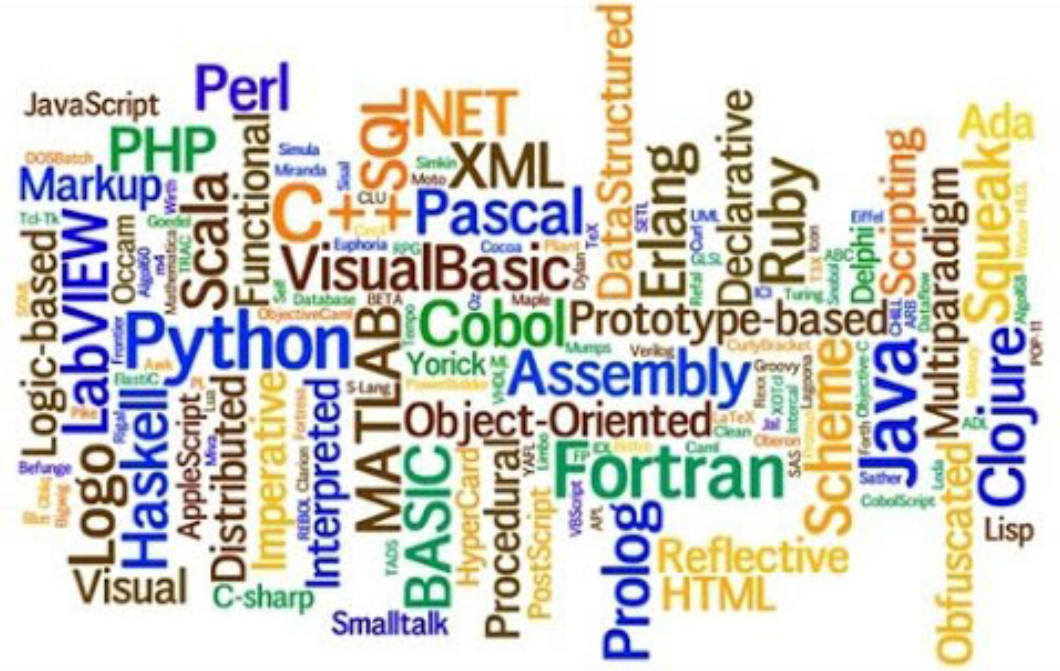
Лекція 3

ПОЧАТКОВІ ПОНЯТТЯ ПРО МОВУ ПРОГРАМУВАННЯ PYTHON

Типи даних і змінні у мові Python

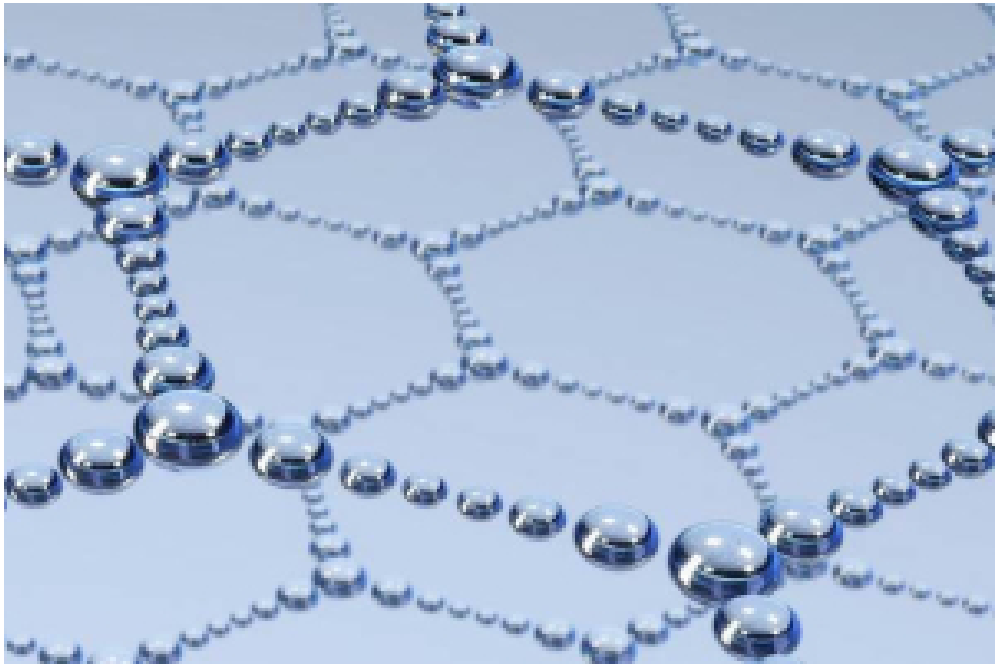
Основні характеристики мови Python



1. Курс присвячено одній з популярних мов програмування, що бурхливо розвиваються – Python. Мова Python дозволяє швидко створювати програми, допомагає в інтеграції програмного забезпечення для розв'язування виробничих завдань.

2. Python має багату стандартну бібліотеку й велику кількість модулів розширення практично для всіх потреб в галузі інформаційних технологій.

3. Завдяки зрозумілому синтаксису вивчення мови не становить великої проблеми. Тому вона має широке розповсюдження.

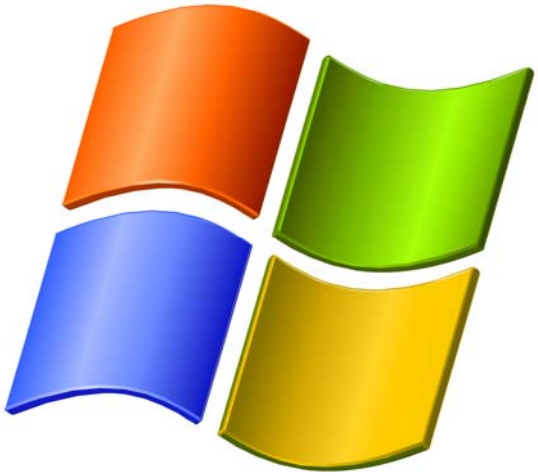


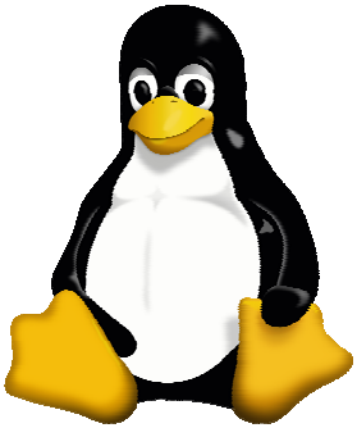
4. Написані на ній програми структуровані за формою, і в них легко простежити логіку роботи.

5. На прикладі мови Python легко усвідомити такі важливі поняття як:

- об'єктно-орієнтоване програмування (ООП),
- функціональне програмування,
- подійно-керовані програми (Gui-додатки),
- формати представлення даних (Unicode, XML і т. п.).

6. Можливість діалогового режиму роботи інтерпретатора Python дозволяє істотно скоротити час вивчення мови.





Windows Linux Android ios

7. Python вільно доступний і швидко розвивається для багатьох платформ, а написані на ньому програми зазвичай переносяться між платформами без змін.

Історія створення мови Python

Створення Python було почато Гвідо ван Россумом (Guido van Rossum) в 1991 році.

Він вибрав назву Python на честь комедійних серій BBC "Літаючий цирк Монти-Пайтона", а зовсім не за назвою змії.

З того часу Python розвивався за підтримки тих організацій, у яких Гвідо працював.

Особливо активно мова удосконалюється тоді, коли над нею працює не тільки команда творців, а й співтовариство програмістів усього світу. Однак останнє слово про напрямок розвитку мови залишається за Гвідо ван Россумом.

Як вимовляється назва мови

Python краще вимовляти "пайтон", оскільки це англійське звучання назви мови поширено в усьому світі, хоча деякі говорять "пітон"

Складові мови Python

Python – це універсальна мова програмування. Він має свої переваги й недоліки, а також сфери застосування.

1. Універсальні бібліотеки.

2. Бібліотеки по різних предметних областях:
   - засоби обробки текстів і технології Інтернет,
   - обробка зображень,
   - інструменти для створення додатків,
   - механізми доступу до баз даних,
   - пакети для наукових обчислень,
   - бібліотеки побудови графічного інтерфейсу і т. п.

3. Засоби інтеграції з мовами C, C++ (і Java)

Підтримує 3 парадигми програмування

Мова Python підтримує три основні **парадигми** програмування:

1. імперативне програмування
   (процедурний, структурний, модульний підходи),
2. об'єктно-орієнтоване програмування,
3. функціональне програмування.

Python – це технологія для створення програмних продуктів. Вона доступна майже на всіх сучасних платформах (як 32-бітних, так і на 64-бітні)

Багато хто думають, що немає кращого, ніж C/C++, Java, Visual Basic, C#. Однак це не так. Завдяки даному курсу в Python обов'язково з'являться нові прихильники, для яких він стане незамінним інструментом.

Технологія освоєння мови Python

Додаткова робота. Для вивчення деталей опису ви повинні використовувати інструкцію користувача, інші допоміжні методичні матеріали, книги.
Тим більше, що деякі версії мови можуть мати відмінності.

Ми будемо вивчати технологію програмування мовою Python, використовуючи велику кількість прикладів і практичних завдань.

Мета такого підходу – забезпечити відсутність збоїв у ланцюжку **семантика -> синтаксис -> прагматика.**

семантика -> синтаксис -> прагматика

Семантика – це той «зміст», який програміст вкладає у свою програму.

Синтаксис – це засіб, за допомогою якого програміст викладає свій задум у вигляді, зрозумілому інтерпретатору.

Прагматика – це ті дії, які виконує інтерпретатор з реалізації завдання, представленого в синтаксично правильному вигляді.

В цьому є основна перевага Python. Синтаксис мови Python має потужні засоби, які допомагають наблизити розуміння проблеми програмістом до її "розуміння" інтерпретатором. Саме тому Python має найнижчий поріг «входження».

Дзен Пайтона

Дзен або «споглядання», одна з найважливіших шкіл китайського і всього східно-азіатського буддизму

Якщо інтерпретатору Пайтона дати команду

import this (імпортувати "сам об'єкт"),

то виведеться так званий "Дзен Пайтона", що ілюструє ідеологію й особливості даної мови.

Глибоке розуміння цього дзену приходить до тих, хто зможе освоїти мову Python повною мірою й набуде досвіду практичного програмування.

1. Beautiful is better than ugly. Гарне краще, ніж потворне.
2. Explicit is better than implicit. Явне краще, ніж неявне.
3. Simple is better than complex. Просте краще, ніж складне.
4. Complex is better than complicated. Складне краще, ніж заплутане.
5. Flat is better than nested. Плоске краще, ніж вкладене.
6. Sparse is better than dense. Розріджене краще, ніж щільне.
7. Readability counts. Легкість для читання (читабельність) має значення.
8. Special cases aren't special enough to break the rules. Особливі випадки не настільки істотні, щоб порушувати правила.
9. Although practicality beats purity. Хоча практичність важливіша за бездоганність.
10. Errors should never pass silently. Помилки ніколи не повинні замовчуватися.
11. Unless explicitly silenced. За винятком замовчування, яке задано явно.
12. In the face of ambiguity, refuse the temptation to guess. У випадку неоднозначності опирайтеся спокусі вгадати.

13. There should be one — and preferably only one — obvious way to do it. Повинен існувати один — і, бажано, тільки один — очевидний спосіб зробити це.
14. Although that way may not be obvious at first unless you're Dutch. Хоча він може бути з першого погляду не очевидний, якщо ти не голландець.
15. Now is better than never. «Зараз» краще, ніж «ніколи».
16. Although never is often better than *right* now. Хоча, «ніколи» інколи (і часто!) краще, ніж «прямо зараз».
17. If the implementation is hard to explain, it's a bad idea. Якщо реалізацію складно пояснити — це погана ідея.
18. If the implementation is easy to explain, it may be a good idea. Якщо реалізацію легко пояснити — це може бути гарна ідея.
19. Namespaces are one honking great idea — let's do more of those! Простори імен — прекрасна ідея, давайте робити їх більше!

Що таке змінна?

Вікіпедія

Змінна величина в математиці — символ, що позначає будь-яке число в алгебраїчному виразі.

$$a^2 + b^2 = c^2 \begin{cases} 3^2 + 4^2 = 5^2 \\ 5^2 + 12^2 = 13^2 \\ 7^2 + 24^2 = 25^2 \end{cases}$$

Змінна (програмування) — поіменована, або адресована у інший спосіб, область пам'яті, адресу якої можна використовувати для здійснення доступу до даних і змінювати значення в ході виконання програми.

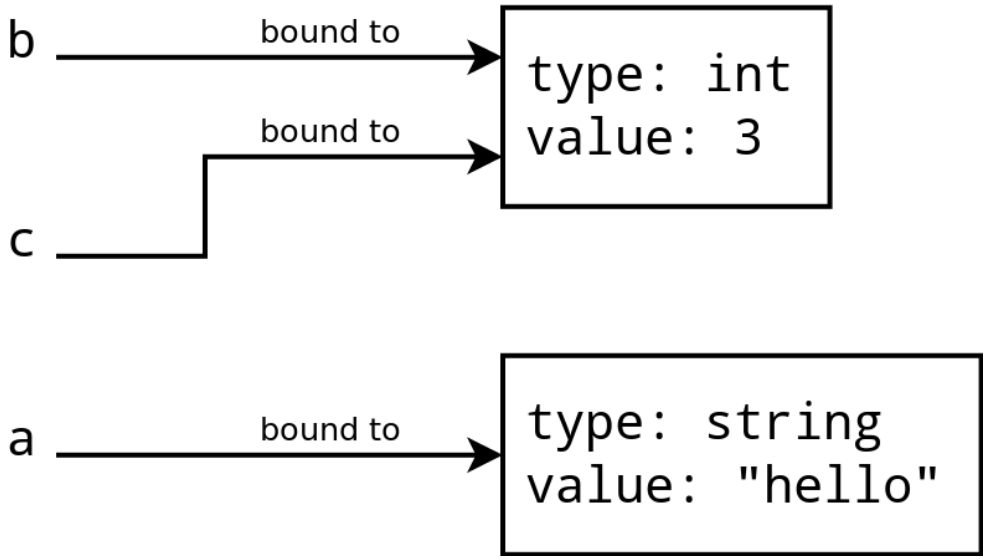
Представлення даних в Python

Executed Code:

Variable Assignment

```
a = 3
b = a
c = a
a = "hello"
```

Variables **Values in Memory**



Усі дані в мові Python представлені об'єктами. Кожний об'єкт має тип даних і значення. Для доступу до об'єкта призначені змінні. При ініціалізації в змінній зберігається посилання на об'єкт (адреса об'єкта в пам'яті комп'ютера). Завдяки цьому посиланню можна надалі змінювати об'єкт із програми.

Як вибирати ім'я для змінної

Строго дотримуватись:

1. Кожна змінна повинна мати унікальне ім'я, що складається з латинських букв, цифр і знаків підкреслення

Приклади правильного задавання змінних:

A, a, bin, bin124, rim_12, ALPHA_2016

2. Ім'я змінної не може починатися із цифри.

$\backslash$1A, $\backslash$5myData

3. Слід уникати символу підкреслення на початку імені

$\neq$ velocity, $\neq$ pressure123

оскільки ідентифікатори з таким символом мають спеціальне призначення.

4. Як ім’я змінної не можна використовувати ключові слова. Одержати список усіх ключових слів дозволяє код:

Список усіх ключових слів

```
>>> import keyword
>>> keyword.kwlist
 ['False', 'None', 'True', 'and', 'as',
'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

5. Слід уникати збігів із вбудованими ідентифікаторами. Одержання списку вбудованих ідентифікаторів

```
>>> import builtins
>>> dir(builtins)
```

['Arithmeticerror', 'Assertionerror', 'Attributeerror', 'Baseexception', 'Blockingioerror',.....

'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

Рекомендовано дотримуватись:

1. Не використовувати кирилицю при назві змінних.

~~Скорость~~ , ~~п_125~~

2. Назва змінної повинна обов'язково нести змістовне навантаження.

My_age = 16, my_height = 175

В імені змінної важливо враховувати регістр букв: x і X — різні змінні:

Приклад.

```
>>> x = 10; X = 20
>>> x, X
```

```
(10, 20)
```

Типи даних

bool - логічний тип даних. **(Числа)**

Може містити значення `True` або `False`, які поводяться як числа 1 і 0 відповідно.

```
>>> type(True), type(False)
(<class 'bool'>, <class 'bool'>)

>>> int(True), int(False)
(1,0)
```

Логічні змінні також можуть використовуватися в логічних виразах (наприклад, в умовних операторах). Python очікує, що логічний вираз після виконання в результаті дає логічне значення. Це можна пояснити на прикладах.

Приклад 1.

```
size=int(input("write the number"))
if size >= 0:
  print('number positive')
else:
  print('number negative')
```

У режимі інтерпретатора ми можемо побудувати логічний вираз і одержимо в результаті False або True

Приклад 2

```
>>> size = 1
>>> size < 0
False
>>> size = 0
>>> size < 0
False
>>> size = -1
>>> size < 0
True
```

Через деякі старі проблеми, що залишилися від Python 2, логічні значення можна розглядати як числа. True 1; False 0.

Приклад 3

```
>>> True + True
2
>>> True - False
1
>>> True * False
0
```

Цілочисельний тип int (Числа)

Це тип для цілих чисел. Розмір числа обмежений лише обсягом оперативної пам'яті:

Приклад 4.

```
>>>type(2147483647),
type(99999999999999999999999)
(<class 'int'>, <class 'int'>)
```

Як правило, тип `int` представляє числа в діапазоні

−2 147 483 648 (-231) до 2 147 483 647 (231−1)

Для задавання значення змінної типу `int` можна використовувати системи числення з основами 2, 8, 10, 16

Приклад 5. Введення літералів у різних системах числення

```
# Десяткове число
>>> 12344556778778
12344556778778
# Двійкове число (починається з префікса 0b)
>>> 0b00110101101010101010101010101
28136789
# Вісімкове число (починається з префікса 0o)
>>> 0o1237645
343973
# Шістнадцяткове число (починається з префікса 0x)
>>> 0xdec1F
912415
```

Тип з плаваючою точкою float (Числа)

Представляє числа з плаваючою точкою подвійної точності

```
>>> type(5.1), type(8.5e-3)
(<class 'float'>, <class 'float'>)
```

Числа записуються з десятковою точкою або в експонентній формі запису.

0.0, 4., 5.7, -2.5, -2e9, 8.9e-4.

2.1e-1=0.21, 2.1e1=21

Приклад 6.

```
>>> float(5.4)
5.4
>>> float(8.9e-4)
0.00089
>>> float(8.9e-12)
8.9e-12
```

Тип комплексні числа `complex` (Числа)

Комплексні числа задають у форматі:

<Дійсна частина> + <Уявна частина>j

Для позначення уявної частини можна використовувати як букву J, так і букву j.

```
>>> type(2+2j)
<class 'complex'>
```

Приклад 7

```
>>> 2+5J
(2+5j)
>>> 23j
23j
>>> 11+3j, 23+12j
((11+3j), (23+12j))
```

Тип NoneType (Спеціальні типи)

Це об'єкт зі значенням `None`.

Він позначає відсутність будь-якого значення

```
>>> type(None)
<class 'NoneType'>
```

У логічному контексті значення `None` інтерпретується як `False`:

```
>>> bool(None)
False
```

Приклад 8

Можна створити змінну типу `NoneType`:

```
my_variable = None
```

Перевірити:

```
if my_variable is None:
    print('my_variable is None')
else:
    print('my_variable is not None')
```

Тип Unicode-Рядки `str` (Послідовності)

Рядки використовують для запису текстової інформації. Тип `str` відноситься до групи «Послідовності».

```
>>> type ("Рядок")
<class 'str'>
```

Приклад 9.

```
>>> S = "Spam"
>>> len(S)     #довжина
4
>>> S[0]       #перший елемент в S
"S"
>>> S[1]       #другий елемент зліва
"p"
```

Індекс	0	1	2	3
Рядок	S	p	a	m

Типи bytes і bytearray (Послідовності)

- `bytes` — незмінювана послідовність байтів:

```
>>> type(bytes("Рядок", "utf-8"))
<class 'bytes'>
```

- `bytearray` — змінювана послідовність байтів:

```
>>> type(bytearray("Рядок", "utf-8"))
<class 'bytearray'>
```

Порівняємо близькі типи

1. тип str – об'єкт, що є послідовністю символів (тобто складається з рядків довжини 1)
2. типи bytes і bytearray – об'єкти, що складаються з послідовності цілих чисел (від 0 до 255), що є байтовими ASCII кодами

Тип list- список (Послідовності)

Тип даних `list` аналогічний масивам в інших мовах програмування: `L=[1, 2, 3]`

```
>>> type(=[1, 2, 3])
<class 'list'>
```

Варіанти задавання списку за допомогою літералів і спискових включень. **Приклад10.**

```
>>> list1=[1,2,3,4]
>>> print(list1)
[1, 2, 3, 4]
>>> list2 = [x**2 for x in range(10)]
>>> print(list2)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> list3 = list("abcde")
>>> print(list3)
['a', 'b', 'c', 'd', 'e']
```

Тип tuple — кортеж (Послідовності)

```
>>> type ( (1, 2, 3) )
<class 'tuple'>
```

Константна послідовність (різнорідних) об'єктів. Зазвичай записують в круглих дужках.

Приклад 11.

```
>>> for s in "one","two": print(s)
one                          #цикл за значеннями кортежу
two
>>>p = (1.2, 3.4, 0.9)#точка в тривимірному просторі
>>> print(p)
(1.2, 3.4, 0.9)
>>> p1 = 1, 3, 9             #без дужок
>>> print(p1)
(1, 3, 9)
>>> p2 = 1, 2, 3, 4,  #кома ігнорується
>>> print(p2)
(1, 2, 3, 4)
```

Тип range — діапазони (Послідовності)

```
>>> type(range(1, 10))
<class 'range'>
```

Приклад 12

```
>>>#one parameter
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

```
>>> # Two parameters
>>> for i in range(3, 6):
...     print(i)
...
3
4
5
```

```
>>> # Three parameters
>>> for i in range(4, 10, 2):
...     print(i)
...
4
6
8
```

```
>>> # Going backwards
>>> for i in range(0,-10,-2):
... print(i)
...
0
-2
-4
-6
-8
```

Тип dict — словники

Тип даних `dict` аналогічний асоціативним масивам в інших мовах програмування: `Slov={"x":5, "y":20}`

```
>>> type( {"x": 5, "y": 20} )
<class 'dict'>
```

Словник (Хеш масив) – це структура даних для зберігання пар ключ-значення, де значення однозначно визначається ключем. Як ключі можуть використовуватись змінні типу числа, рядка, кортежу й т. п. Порядок пар ключ-значення довільний. **Приклад 13**

```
dan={'Name':'Ivan','Age':17,'Course':1}
print(dan)
print('Name: ',dan['Name'])
------
{'Course': 1, 'Age': 17, 'Name': 'Ivan'}
Name:  Ivan
```

Тип set і frozenset — множини (колекції унікальних об'єктів):

```
>>>type( {"a", "b", "c"})
<class 'set'>
```
– змінювані множини

```
>>> type(frozenset(["a", "b", "c"]))
<class 'frozenset'>
```
- незмінювані множини

Об'єкт `set` є змінюваним неупорядкованим набором незмінних значень.
Загальні області застосування включають тестування членства, видалення дублікатів з послідовності і обчислення математичних операцій, таких як перетин, об'єднання, різниця й симетрична різниця.

Об'єкт `frozenset` не можна змінити під час виконання програми

Приклад 14

```
Group={"Galenko","Petrenko","Sydorko"}
Total=len(Group)
print(Group)
print("Total: ",Total)
if " Petrenko " in Group:
      print("Petrenko is a student")
Group.add("Trump")
print(Group)
------
{'Galenko', 'Sydorko', 'Petrenko'}
Total:  3
Petrenko is a student
{'Galenko', 'Sydorko', 'Trump', 'Petrenko'}
```

Тип function — функції:

```
>>> def func(): pass
>>> type(func)
<class 'function'>
```

Тип module — модулі:

```
>>> import sys
>>> type(sys)
<class 'module'>
```

Тип type — тип:

Не дивуйтеся! Усі дані в мові Python є об'єктами, навіть самі типи даних!

```
>>> class C: pass
>>> type(C)
<class 'type'>
>>> type(type('мм'))
<class 'type'>
```

Змінювані й незмінювані типи

Змінювані типи: списки, словники й тип `bytearray`.

Приклад15. Зміна елемента списку (тип list)

```
>>> arr = [1,2,3]
>>> arr[0]=0 #змінюємо перший елемент списку
>>> arr
[0, 2, 3]
```

Незмінювані типи: числа, рядки, кортежі, діапазони й тип `bytes`. Для додавання двох рядків використовуємо операцію *конкатенації*, а посилання на новий об'єкт присвоюємо змінній. **Приклад 16**

```
>>> str1 = "авто"
>>> str2 = "транспорт"
>>> str3 = str1 + str2 # Конкатенація
>>> print(str3)
автотранспорт
```

Послідовності й відображення

Послідовності: рядки, списки, кортежі, діапазони, типи `bytes` і `bytearray`.

Відображення: словники.

Послідовності й відображення підтримують механізм ітераторів, що дозволяє зробити обхід усіх елементів за допомогою функції `next()`.

Приклад 17. Вивести елементи списку можна так:

```
>>> arr = [1, 2, 3]
>>> i = iter(arr)
>>> next(i)
1
>>> next(i)
2
>>> next(i)
3
```

Механізм ітераторів для словника (тип dict)

На кожній ітерації повертається ключ:

Приклад 18.

```
d = {"x": 1, "y": 2}
i = iter(d)
c = next(i)
print(c,':',d[c])
c = next(i)
print(c,':',d[c])
-----
x : 1
y : 2
```

Цей механізм майже не використовують у такому вигляді. Частіше його використовують опосередковано в операторі циклу.

Приклад 19. Вивід елемента списку:

```
>>> for i in [1, 2]: print(i)
1
2
```

Приклад 20. Перебрати слово по буквах:

```
>>> for i in "Рядок": print(i)
-----
Рядок
```

Приклад 21. Перебір елементів словника:

```
>>> d = {"x": 1, "y": 2}
>>> for key in d:
    print(d[key])

1
2
```

Присвоювання значення змінним

У мові Python використовується динамічна типізація. Значення змінній присвоюється за допомогою оператора = у такий спосіб:

```
>>> x = 7          #Тип int
>>> y = 7.8        #Тип float
>>> s1 ='Рядок1'  # s1 присвоєне значення Рядок
>>> s2 ='Рядок2'  #s2 присвоєне значення Рядок
>>> b = True       # b присвоєне логіч. значення True
```

В одному рядку можна присвоїти значення відразу декільком змінним:

```
>>> x = y = 10
>>> x, y
(10, 10)
```

Особливості групового присвоєння

1. Після присвоювання значення в змінній зберігається посилання на об'єкт, а не сам об'єкт. Це обов'язково слід враховувати при *`груповому присвоюванні`*.
2. Групове присвоювання можна використовувати для чисел, рядків і кортежів, але для змінюваних об'єктів цього робити не можна.

Приклад 22. Чому не можна застосовувати групове присвоєння для змінюваних об'єктів

```
>>> x = y = [1, 2]   #Наче створили два об'єкти
>>> x, y
([1, 2], [1, 2])
```

У цьому прикладі ми створили список із двох елементів і присвоїли значення змінним `x` и `y`.

Тепер спробуємо змінити значення в змінній `y`:

```
>>> y[1] = 100 # Змінюємо другий елемент
>>> x, y
([1, 100], [1, 100])
```

Як видно з прикладу, зміна значення в змінній `y` привела також до зміни значення в змінній `x`.

Таким чином, обидві змінні посилаються на той самий об'єкт, а не на два різні об'єкти.

Щоб одержати два об'єкти, необхідно робити роздільне присвоювання:

```
>>> x = [1, 2]
>>> y = [1, 2]
>>> y[1] = 100    # Змінюємо другий елемент
>>> x, y
([1, 2], [1, 100])
```

Перевірка на подвійне посилання

Перевірити, чи посилаються дві змінні на той самий об'єкт, дозволяє оператор `is`. Якщо змінні посилаються на той самий об'єкт, то оператор `is` повертає значення True:

Приклад 23.

```
>>> x = y = [1,2]  #один об'єкт
>>> x is y
True
>>> x = [1,2]     #різні об'єкти
>>> y = [1,2]     #різні об'єкти
>>> x is y
False
```

Об'єднання посилань при кешируванні

З метою підвищення ефективності коду інтерпретатор виконує кеширування малих цілих чисел і невеликих рядків.

Це означає, що якщо ста змінним присвоєне число 2, то в цих змінних буде збережено посилання на той самий об'єкт.

Приклад 24.

```
>>> x = 2; y = 2; z = 2
>>> x is y, y is z
(True, True)
```

Перевірка кількості посилань

Подивитися кількість посилань на об'єкт дозволяє метод `getrefcount ()` з модуля `sys`:

```
>>> import sys  # Підключаємо модуль sys
>>> sys.getrefcount(2)
304
```

Коли число посилань на об'єкт дорівнює нулю, об'єкт автоматично видаляється з оперативної пам'яті. Виключенням є об'єкти, які підлягають кешируванню.

Позиційне присвоювання

Крім групового присвоювання, мова Python підтримує *позиційне присвоювання*. У цьому випадку змінні вказуються через кому ліворуч від оператора =, а значення — через кому праворуч.

Приклад 25. Позиційне присвоювання:

```
>>> x, y, z = 1, 2, 3
>>> x, y, z
(1, 2, 3)
```

За допомогою позиційного присвоювання можна поміняти значення змінних місцями.

Приклад 26.

```
>>> x, y = 1, 2; x, y
(1, 2)
>>> x, y = y, x; x, y
(2, 1)
```

По обидві сторони оператора = можуть бути зазначені послідовності.

Згадаємо, що послідовностями є рядки, списки, кортежі, діапазони, типи `bytes` і `bytearray`.

Приклад 27

```
>>> x, y, z = ”123”  # Рядок
>>> x, y, z
('1', '2', '3')
>>> x, y, z = [1, 2, 3] # Список
>>> x, y, z
[1, 2, 3]
>>> x, y, z = (1, 2, 3) # Кортеж
>>> x, y, z
(1, 2, 3)
>>> [x, y, z] = (1, 2, 3) #Список ліворуч, кортеж праворуч
>>> x, y, z
(1, 2, 3)
```

Питання відповідності елементів при обміні

Зверніть увагу на те, що кількість елементів праворуч і ліворуч від оператора = повинна збігатися, інакше буде виведене повідомлення про помилку:

```
>>> x, y, z = (1, 2, 3, 4)
Traceback (most recent call last):
File "<pyshell#130>", line 1, in <module>
x, y, z = (1, 2, 3, 4)
Valueerror: too many values to unpack
(expected 3)
```

Python 3 при невідповідності кількості елементів праворуч і ліворуч від оператора = дозволяє зберегти в змінній список, що складається із зайвих елементів. Для цього перед іменем змінної вказується зірочка (*).

Приклад 28

```
>>>x,y,*z=(1,2,3,4)
>>>x,y,z
(1,2,[3, 4])
>>>x,*y,z = ( 1,2,3,4)
>>>x,y,z
(1,[2,3],4)
>>>*x,y,z =(1,2,3,4)
>>>x,y,z
([1,2],3,4)
>>>x,y,*z =(1,2,3)
>>>x,y,z
(1,2,[3])
>>>x,y*z=1,2)
>>>x,y,z
(1,2,[])
```

Як видно із прикладу, змінна, перед якою зазначена зірочка, завжди містить список. Якщо цій змінній не вистачило значень, то їй присвоюється порожній список.

Слід пам'ятати, що зірочку можна вказати тільки перед однією змінною.

В протилежному випадку виникне неоднозначність і інтерпретатор виведе повідомлення про помилку:

```
>>> *x, y, *z = (1, 2, 3, 4)
Syntaxerror: two starred expressions in assignment
```

Перевірка типу даних

Python у будь-який момент часу змінює тип змінної відповідно до даних, що зберігаються в ній.

Приклад 29.

```
>>> a = 'Рядок'    #тип str
>>> a = 7          #тепер змінна має тип int
```

Визначити, на який тип даних посилається змінна, дозволяє функція `type (<змінної>)`:

```
>>> type(a)
<class 'int'>
```

Перевірити тип даних, що зберігаються в змінній, можна такими способами:

- порівняти значення, що повертається функцією `type()`, з назвою типу даних:

```
>>> x = 10
>>> if type(x) is int: print("Це тип int")
```

- перевірити тип за допомогою функції `isinstance ()`:

```
>>> s = "Рядок"
>>> if isinstance(s, str):
    print("Це тип str")

Це тип str
```

Принципи перетворення типів даних

1. Після присвоювання значення в змінній зберігається посилання на об'єкт певного типу, а не сам об'єкт.

2. Якщо потім змінній присвоїти значення іншого типу, то змінна буде посилатися на інший об'єкт, і тип даних відповідно зміниться.

3. Таким чином, тип даних у мові Python — це характеристика об'єкта, а не змінної. Змінна завжди містить тільки посилання на об'єкт.

Операції залежать від типу значення

Після присвоювання змінній значення над цим значенням можна виконувати операції, призначені лише для цього типу даних.

Наприклад, рядок не можна додати до числа, тому що це приведе до виводу повідомлення про помилку:

```
>>> 2 + "25"
Traceback (most recent call last):
File "<pyshell#o>", line 1, in <module>
+ "25"
Typeerror:unsupported operand type(s)for+:'int'and'str'
```

Функції для перетворення типів даних

1. Перетворення об'єкта в логічний тип даних

`bool([<об'єкт>]), де[]`–необов'язковий параметр

Приклад 30.

```
>>>bool(0),bool(1),bool(""),bool("Рядок"),bool([1,2])
(False, True, False, True, True)

>>>bool([])
False
```

2. Перетворення об'єкта в число

```
int([<об'єкт>[,<Система числення>]])
```

У другому параметрі можна вказати систему числення (значення за замовчуванням — 10).

Приклад 31:

```
>>> int(7.5),int("71")
(7, 71)

>>>int("71",10),int("71",8),int("0o71",8),int("A",16)
(71, 57, 57, 10)
```

3. Перетворення цілого числа або рядка в дійсне число.

```
float ( [ <Число рядок> ])
```

Приклад 32:

```
>>> float(7),float("7.1")
(7.0, 7.1)

>>> float("Infinity"), float("-inf")
(inf, -inf)

>>> float("Infinity") + float("-inf")
nan
```

4. Перетворення об'єкта в рядок

```
str ([ <Об'єкт> ])
```

Приклад 32:

```
>>> str(125), str([1, 2, 3])
('125', '[1,2,3]')
>>> str((1,2,3)), str({"x": 5, "в": 10})
('(1, 2, 3)', "{'x': 5, 'y': 10}")

>>> str( bytes("a", "UTF-8") )
"b'\\xd0\\xb0'"
>>> str( bytes("F", "UTF-8") )
"b'F'"
>>> str( bytearray("З", "UTF-8") )
"bytearray(b'\\xd0\\xa1')"
>>> str( bytearray("L", "UTF-8") )
"bytearray(b'L')"
```

5. Перетворення послідовності в список

```
list(<послідовність>)
```

Приклад 33:

```
>>> list("12345")        #Рядок str⇒ list
['1', '2', '3', '4', '5']

>>> list("Рядок")        #Рядок  str⇒ list
['Р', 'я', 'д', 'о', 'к']

>>> list({1,2,3,4,5})   #Множина set⇒ list
[1, 2, 3, 4, 5]

>>> list({"x": 5, "y": 10})
 ['x', 'y']              #Словник  dict⇒ list
```

6. Перетворення послідовності в кортеж

```
tuple(<Послідовність>)
```

Приклад 34

```
 >>> tuple("123456")  #Рядок str⇒ tuple
('1', '2', '3', '4', '5', '6')

>>>tuple([1,2,3,4,5]) #Список list⇒ tuple
(1, 2, 3, 4, 5)

>>> tuple(({"x": 5, "y": 10}))
('x', 'y')                  #Словник dict⇒ tuple
```

Приклад з користувацьким введенням даних

Розглянемо можливість додавання двох чисел, введених користувачем.

Вводити дані дозволяє функція `input()`.

Приклад 35.

```
x = input("x = ")    # Уводимо 5
y = input("y = ")    # Уводимо 12
print (x + y)
```

```
512
```

```
x = str
y = ing
string
```

Результатом виконання цього скрипта є не число, а рядок 512. Таким чином, слід запам'ятати, що функція `input()` повертає результат у вигляді рядка. Щоб додати два числа, необхідно перетворити рядок на число

Приклад 36. Перетворення рядка в число

```
x = int(input("x = "))# Вводимо 5
y = int(input("y = "))# Вводимо 12
print (x + y)
17
```

У цьому випадку ми одержимо число 17, як і повинно бути. Однак якщо користувач замість числа введе рядок, то програма завершиться з фатальною помилкою.

```
x = w
Traceback (most recent call last):
  File "C:/PYTHON/Samples.py", line 1, in <module>
    x = int(input("x = "))   # Уводимо 5
Valueerror: invalid literal for int() with base 10: 'w'
```

Видалення змінної

Вилучити змінну можна за допомогою інструкції `del`:

```
Del <Змінна1>[, ... , <Змінна n>]
```

Приклад 37. Видалення однієї змінної:

```
>>> x=20; x
20
>>> del x
>>> x
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Nameerror: name 'x' is not defined
```

Приклад видалення декількох змінних:

```
>>> x, y = 10, 20
>>> del x, y
```