МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра "Захист інформації"

Робота з логічними та числовими типами даних

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 2
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Затверджено
на засіданні кафедри
"Захист інформації"
протокол № 01 від 29.08.2024 р.

Львів – 2024

Робота з логічними та числовими типами даних: Методичні вказівки до лабораторної роботи № 2 з курсу «Програмування скриптовими мовами» для студентів спеціальності «Кібербезпека» / Укл. *Я. Р. Совин* – Львів: Національний університет "Львівська політехніка", 2024. – 24 с.

Укладач: Я. Р. Совин, канд. техн. наук, доцент

Відповідальний за випуск: В. Б. Дудикевич, д.т.н., професор

Рецензенти: А. Я. Горпенюк, канд. техн. наук, доцент
Ю. Я. Наконечний, канд. техн. наук, доцент

Мета роботи – ознайомитись з вбудованими логічними і числовими типами Python та операторами і функціями для роботи з ними**.**

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Структура програми мовою Python

Форматування коду. Програма на мові Python представляє собою звичайний текстовий файл з інструкціями. Кожна інструкція розташовується в окремому рядку. Якщо інструкція не є вкладеною, вона повинна починатися з початку рядка, інакше буде виведено повідомлення про помилку:

```
 print("Hello world!")    # Помилка, зайвий відступ
```

У цьому випадку перед інструкцією *print* розташований один зайвий пробіл, який привів до виводу повідомлення про помилку.

Ще однією відмінною особливістю мови Python є відсутність обмежувальних символів для виділення інструкцій всередині блоку. Блок циклу у мові Python буде виглядати так:

```
i = 1
while i < 11:
    print(i)
    i += 1
print("Кінець програми")
```

Перед усіма інструкціями всередині блоку повинна бути розташована однакова кількість пробілів. Саме так в мові Python виділяються блоки. Інструкції, перед якими розташовано однакову кількість пробілів, є тілом блоку. У нашому прикладі дві інструкції виконуються десять разів. Кінцем блоку є інструкція *print()*, яка виводить рядок *"Кінець програми"*. Якщо кількість пробілів всередині блоку буде різна, то інтерпретатор виведе повідомлення про фатальну помилку, і програма буде зупинена.

У мові Python прийнято використовувати чотири пробіли для виділення інструкцій всередині блоку.

Якщо блок складається з однієї інструкції, то допустимо розмістити її в одному рядку з основною інструкцією. Наприклад, код:

```
for i in range(1, 11):
    print(i)
print("Кінець програми")
```

можна записати так:

```
for i in range(1, 11): print(i)
print("Кінець програми")
```

Якщо інструкція є занадто довгою, то її можна перенести на наступний рядок, наприклад, так:

♦ в кінці рядка поставити символ \, після якого повинен слідувати перевід рядка. Інші символи (в тому числі і коментарі) неприпустимі. Приклад:

```
x = 15 + 20 \
    + 30
print(x)
```

♦ розташувати вираз всередині круглих дужок. Цей спосіб краще, оскільки всередині круглих дужок можна розмістити будь-який вираз. Наприклад:

```
x = (15 + 20     # Це коментар
    + 30)
print(x)
```

Коментарі. Коментарі інтерпретатор повністю ігнорує. Всередині коментаря може розташовуватися будь-який текст, включаючи інструкції, які виконувати не потрібно.

У мові Python присутній тільки однорядковий коментар. Він починається з символу #:

```
# Це коментар
```

Однорядковий коментар може починатися не тільки з початку рядка, але і розташовуватися після інструкції.

```
print("Hello world!") # Виводимо напис за допомогою функції print
```

Якщо ж символ коментаря розмістити перед інструкцією, то вона виконана не буде:

```
# print("Hello world!") Ця інструкція не буде виконана
```

Якщо символ # розташований всередині лапок і апострофів, то він не є символом коментаря:

```
print("# Це не коментар")
```

Так як в мові Python немає багаторядкового коментаря, то коментований фрагмент часто розміщують всередині потроєних лапок (або потроєних апострофів):

```
"""
Ця інструкція не буде виконана
print("Hello world!")
"""
```

Слід зауважити, що цей фрагмент коди не ігнорується інтерпретатором, оскільки він не є коментарем. В результаті виконання такого фрагмента буде створено об'єкт рядкового типу. Проте, інструкції всередині потроєних лапок виконуватися не будуть, оскільки інтерпретатор вважатиме їх простим текстом. Такі рядки є рядками документування, а не коментарями.

1.2. Ввід-вивід результатів роботи програми

Вивести результати роботи програм можна за допомогою функцій *print()*. Функція має наступний формат:

```
print([<Об'єкти>] [, sep = ''] [, end = '\ n'] [/ file = sys.stdout] [, flush = False])
```

Функція *print()* перетворює об'єкт в рядок і подає його на стандартний вивід *stdout*.

З допомогою параметру *file* можна перенаправити вивід в інше місце, наприклад, у файл.

Після виводу рядка автоматично додається символ переводу рядка:

```
print("Рядок 1")
print("Рядок 2")
```

Результат:

```
Рядок 1
Рядок 2
```

Якщо необхідно вивести результат у тому ж рядку, то у функціях *print()* дані вказуються через кому в першому параметрі:

```
print("Рядок 1", "Рядок 2")
```

Результат:

```
Рядок 1 Рядок 2
```

Як видно з прикладу, між рядками, що виводяться автоматично вставляється пробіл. З допомогою параметру *sep* можна вказати інший символ. Наприклад, виведемо рядки без пробілу між ними:

```
print("Рядок 1", "Рядок 2", sep = "")
```

Результат:

```
Рядок 1Рядок 2
```

Після виводу об'єктів в кінці додається символ переводу рядка. Якщо потрібно виконати подальший вивід в тому ж рядку, то в параметрі *end* слід вказати інший символ:

```
print("Рядок 1", "Рядок 2", end = "")
print("Рядок 3")
```

Результат:

```
Рядок 1 Рядок 2 Рядок 3
```

Якщо потрібно вставити символ переводу рядка, то функція *print()* вказується без параметрів:

```
for n in range(1, 5):
    print(n, end = " ")
print()
print("Цей текст в новому рядку")
```

Результат виконання:

```
1 2 3 4
Цей текст в новому рядку
```

Якщо потрібно вивести великий текстовий блок, його слід розмістити між потроєними лапками або потроєними апострофами. У цьому випадку текст зберігає своє форматування:

```
print("""Рядок 1
Рядок 2
Рядок 3""")
```

У результаті виконання цього прикладу отримаємо три рядки:

```
Рядок 1
Рядок 2
Рядок 3
```

Для введення даних у Python 3 призначена функція *input()*, яка отримує дані з стандартного вводу *stdin* та повертає рядок. Функція має наступний формат:

```
[<Значення> =] input([<Повідомлення>])
```

Приклад використання функції *input()*:

```
name = input("Enter name: ")
print("Hello,", name)
```

1.3. Логічні і числові типи даних Python

Розглянемо спочатку найпростіші типи даних Python – логічні значення, та чисельні типи – цілі числа, числа з плаваючою комою (дійсні числа) та комплексні

числа.

1.3.1. Логічний тип даних bool

bool – логічний тип даних. Може містити значення *True* або *False*, які ведуть себе як цілі числа 1 та 0 відповідно:

```
>>> type(False), type(True)
(<class 'bool'>, <class 'bool'>)
>>> int(False), int(True)
(0, 1)
>>> True + True # Еквівалентно 1 + 1
2
>>> True + 2    # Еквівалентно 1 + 2
3
>>> False + 2   # Еквівалентно 0 + 2
2
```

Будь-який об'єкт в логічному контексті може інтерпретуватися як істина (*True*) або як не істина (*False*). Для визначення логічного значення можна використовувати функцію *bool()*.

Значення *True* повертають такі об’єкти:

♦ будь-яке число, не рівне нулю:

```
>>> bool(1), bool(20), bool(-20)
(True, True, True)
>>> bool(1.0), bool(0.1), bool(-20.0)
(True, True, True)
```

♦ не порожній об'єкт:

```
>>> bool("0"), bool([None]), bool((None,))
(True, True, True)
```

Наступні об’єкти інтерпретуються як *False*:

♦ число, рівне нулю:

```
>>> bool(0), bool(0.0)
(False, False)
```

♦ порожній об'єкт:

```
>>> bool(""),bool(''), bool([]), bool(()), bool({})
(False, False, False, False, False)
```

♦ значення None:

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

1.3.2. Цілочисельний тип даних int

int – цілочисельний тип даних для представлення додатніх і від’ємних значень:

```
>>> type(2147483647), type(-2), type(0)
(<class 'int'>, <class 'int'>, <class 'int'>)
```

Числа типу *int* мають необмежену розрядність, яка збільшується по мірі необхідності, а їх розмір обмежений лише об’ємом оперативної пам’яті:

```
>>> 2 ** 547
46068877256122330959979914318704454234825698491411626322951737063545216826076035792066976625703842377215190124887253966061652644408261628815447195452059277876579532 8
```

Цілі числа можуть бути представлені у різних системах числення:

```
>>> 0b10 # двійкова система
2
>>> 0o10 # вісімкова система
8
>>> 0x10 # шістнадцяткова система
16
```

Взнати довжину числа в бітах можна наступним чином:

```
>>> x = 21 # 21 = 0b10101
>>> x.bit_length()
5
```

1.3.3. Тип даних з плаваючою комою float

float – тип даних з плаваючою комою. Тип *float* в мові Python представлений звичайними числами з плаваючою комою подвійної точності (64 біта), що відповідають стандарту IEEE 754. Це дозволяє забезпечувати представлення приблизно 17 значущих розрядів, з експонентою в діапазоні від -308 до 308. Це повністю відповідає типу *double* в мові C.

```
>>> type(5.1), type(-5.0)
(<class 'float'>, <class 'float'>)
```

Числа з плаваючою комою можуть бути записані в експоненційному форматі:

```
>>> 8.3e3 # 8.3*10^3
8300.0
>>> 8.3e-2 # 8.3*10^-2
0.083
>>> 8e3 # Число в експоненційному форматі завжди типу float
8000.0
```

Якщо один з операндів є з плаваючою комою, то результат завжди буде мати тип з плаваючою комою:

```
>>> type(2 + 3.0)
<class 'float'>
```

Результат операції ділення завжди має тип *float*:

```
>>> type(9 / 3)
<class 'float'>
```

Тип *float* підтримує такий корисний метод як *is_integer()*, який повертає *True*, якщо задане дійсне число не містить дробової частини, тобто фактично представляє собою ціле число:

```
>>> (2.0).is_integer(), (2.1).is_integer()
(True, False)
```

Слід пам’ятати, що числа типу *float* представляються з певною точністю:

```
>>> 3 * 0.1
0.30000000000000004
>>> 0.3 - 0.1 - 0.1 - 0.1
-2.7755575615628914e-17
```

1.3.4. Комплексний тип даних complex

complex – тип даних для представлення комплексних чисел. Комплексні числа складаються з двох частин: дійсної та уявної (з суфіксом *j*), кожна з яких є числом з плаваючою комою.

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

Для отримання дійсної частини використовується синтаксис атрибуту *x.real*, а для отримання уявної - *x.imag*.

```
>>> a = 2 + 3j
>>> a.real
2.0
>>> a.imag
3.0
```

1.3.5. Перевірка та перетворення типів даних

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

```
>>> a = 7   # Змінна типу int
>>> a = 7.0 # Тепер змінна має тип float
```

Визначити, на який тип даних посилається змінна, дозволяє функція *type (<Ім'я змінної>)*:

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

Після присвоєння змінній значення над нею можна робити операції, призначені лише для цього типу даних. Наприклад, рядок не можна скласти з числом, так як це призведе до виводу повідомлення про помилку:

```
>>> 2 + "25"
Traceback (most recent call last):
  File "<pyshell#174>", line 1, in <module>
    2 + "25"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Операції над числами різних типів повертають число, що має більш складний тип з типів, що беруть участь в операції. Цілі числа мають найпростіший тип, далі йдуть дійсні числа і найскладніший тип – комплексні числа. Таким чином, якщо в операції беруть участь ціле число і дійсне, то ціле число буде автоматично перетворено в дійсне число, потім буде проведена операція над числами, а результатом стане дійсне число.

```
>>> type(2 + 3.0)
<class 'float'>
```

Для явного перетворення типів даних призначені наступні функції:

♦ *bool ([<Об'єкт>])* - перетворює об'єкт в логічний тип даних. Приклади:

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

♦ *int ([<Об'єкт> [,<Система числення>]])* - перетворює об'єкт в число. Дійсні числа заокруглюються шляхом відкидання дробової частини. У другому параметрі можна вказати систему числення (значення за замовчуванням - 10). Приклади:

```
>>> int(7.2), int(7.5), int(7.8), int("71")
(7, 7, 7, 71)
>>> int("71", 10), int("71", 8), int("0o71", 8), int("A", 16)
(71, 57, 57, 10)
```

В якості приклада розглянемо можливість додавання двох чисел введених користувачем з допомогою функції *input()*. Функція *input()* повертає результат у вигляді рядка. Щоб підсумувати два числа, необхідно перетворити рядок в число:

```
>>> x = int(input("x = ")) # Вводимо 5
x = 5
```

```
>>> y = int(input("y = ")) # Вводимо 7
y = 7
>>> print(x + y)
12
```

♦ *float ([<Число або рядок>])* – перетворює ціле число або рядок у число з плаваючою комою. Приклади:

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

♦ *complex([real[, imag]])* – повертає комплексне число зі значенням *real + imag*j* або перетворює рядок чи число у комплексне число. Другий параметр не може бути рядком. Рядок не повинен містити пробілів. Приклади:

```
>>> complex()
0j
>>> complex(3, 5)
(3+5j)
>>> complex(3, 5j)
(-2+0j)
>>> complex('2+2j')
(2+2j)
```

1.4. Оператори

Оператори дозволяють зробити з даними певні дії. Наприклад, оператори присвоювання служать для збереження даних у змінній, математичні оператори дозволяють виконати арифметичні обчислення. Розглянемо оператори для логічних та чисельних типів даних.

1.4.1. Логічні оператори

. Оператори *and, or, not* для роботи з логічними типами даних представлені в табл. 1.

Таблиця 1

Логічні оператори

Операція	Опис	Вираз	Результат
not x	Логічне НІ	*not False* *not True*	*True* *False*
x and y	Логічне І	*False and False* *False and True* *True and False* *True and True*	*False* *False* *False* *True*
x or y	Логічне АБО	*False or False* *False or True* *True or False* *True or True*	*False* *True* *True* *True*

Пріоритет оператора *not* вищий, ніж у оператора *and*, пріоритет якого, в свою чергу, вище, ніж у оператора *or*. Наприклад:

```
>>> False and False or True
True
```

У логічному виразі можно вказувати зразу декілька умов:

```
>>> x = 10
>>> 1 < x < 20, 11 < x < 20
(True, False)
```

Логічні оператори *and* та *or* повернуть *True* або *False*, якщо їх операндами є логічні вирази:

```
>>> (2 > 4) and (45 > 3) # Комбінація False and True поверне False
False
```

Якщо операндами операторів *and* чи *or* є об'єкти, то в результаті Python поверне об'єкт:

```
>>> '' and 2 # False and True
''
```

Кілька логічних виразів можна об'єднати в один великий за допомогою наступних операторів:

♦ *and* – логічне І. Якщо *x* у виразі *x and y* інтерпретується як *False*, то повертається *x*, в протилежному випадку – *y*. Тобто для об'єктів-операндів Python обчислює операнди зліва направо і повертає перший об'єкт, який має значення *False* і припиняє подальші обчислення. Приклади:

```
>>> (1 < 5) and (2 < 5) # True and True  == True
True
>>> 10 and 20, 0 and 20, 10 and 0
(20, 0, 0)
```

♦ *or* – логічне АБО. Якщо *x* у виразі *x or y* інтерпретується як *False*, то повертається *y*, в протилежному випадку – *x*. Тобто для об'єктів-операндів Python повертає перший об'єкт, який має значення *True* і припиняє подальші обчислення. Приклади:

```
>>> (1 < 5) or (2 < 5) # True or True == True
True
>>> 10 or 20, 0 or 20, 10 or 0
(10, 20, 10)
```

1.4.2. Цілочисельні оператори

Математичні оператори для роботи з цілими типами даних представлені в табл. 2.

Таблиця 2

Математичні оператори для цілих типів

Вираз	Значення	Коментар
99	99	Цілочисельний літерал
+99	99	Унарний плюс
-99	-99	Унарний мінус
5 + 3	8	Додавання
5 – 3	2	Віднімання
5 * 3	15	Множення
8 / 4	2.0	Ділення, результат завжди типу *float*
5 // 3	1	Ціла частина від ділення
5 % 3	2	Залишок від ділення
5 ** 3	125	Піднесення до степені

Двійкові (бінарні) оператори призначені для маніпуляції окремими бітами.

Ці оператори мають сенс лише для цілих чисел. Мова Python підтримує такі двійкові оператори представлені в табл.

Таблиця

Двійкові оператори

Операція	Результат	Коментар
x | *y*	Побітове АБО *x* та *y*	Bitwise OR
x ^ *y*	Побітове виключне АБО *x* та *y*	Bitwise XOR
x & *y*	Побітове І *x* та *y*	Bitwise AND
x << *n*	*x* зсунуте вліво на *n* біт	Від'ємний зсув заборонений
x >> *n*	*x* зсунуте вправо на *n* біт	Від'ємний зсув заборонений
~*x*	Інверсія бітів *x*	Bitwise NOT

Розглянемо роботу двійкових операторів на прикладах.

♦ ~ – двійкова інверсія. Значення кожного біта замінюється на протилежне:

```
>>> x = 100 # 01100100
>>> x = ~x  # 10011011
>>> x
-101
```

♦ & – бінарне І:

```
>>> x = 100   # 01100100
>>> y = 75    # 01001011
>>> z = x & y # 01000000
>>> "{0:b} & {1:b} = {2:b}".format(x, y, z)
'1100100 & 1001011 = 1000000'
```

♦ | – бінарне АБО:

```
>>> x = 100   # 01100100
>>> y = 75    # 01001011
>>> z = x | y # 01101111
>>> "{0:b} & {1:b} = {2:b}".format(x, y, z)
'1100100 & 1001011 = 1101111'
```

♦ ^ – бінарне виключне АБО (сума за модулем 2):

```
>>> x = 100   # 01100100
>>> y = 250   # 11111010
>>> z = x ^ y # 10011110
>>> "{0:b} & {1:b} = {2:b}".format(x, y, z)
'1100100 & 11111010 = 10011110'
```

♦ << – зсув вліво – зсуває двійкове подання числа вліво на один або більше розрядів і заповнює розряди праворуч нулями:

```
>>> x = 100    # 01100100
>>> y = x << 1 # 11001000
>>> z = y << 1 # 10010000
>>> k = z << 2 # 01000000
```

♦ >> – зсув вправо – зсуває двійкове подання числа вправо на один або більше розрядів і заповнює розряди зліва нулями, якщо число позитивне:

```
>>> x = 100    # 01100100
>>> y = x >> 1 # 00110010
>>> z = y >> 1 # 00011001
>>> к = z >> 2 # 00000110
```

Якщо число від'ємне, то розряди зліва заповнюються одиницями:

```
>>> x = -127   # 10000001
```

```
>>> y = x >> 1 # 11000000
>>> z = y >> 2 # 11110000
>>> к = z << 1 # 11100000
>>> m = k >> 1 # 11110000
```

1.4.3. Оператори чисел з плаваючою комою

Математичні оператори для роботи з числами типу *float* представлені в табл. 3.

Таблиця 3

Математичні оператори для типів float

Вираз	Значення	Коментар
99.0	99.0	Літерал з плаваючою комою
+99.0	99.0	Унарний плюс
-99.0	-99.0	Унарний мінус
5.0 + 3.0	8.0	Додавання
5.0 – 3.0	2.0	Віднімання
5.0 * 3.0	15.0	Множення
8.0 / 4.0	2.0	Ділення
5.2 // 3.7	1.0	Ціла частина від ділення
5.2 % 3.7	1.5	Залишок від ділення
5.0 ** 3.0	125.0	Піднесення до степені

1.4.4. Оператори порівняння

У Python є вісім операторів порівняння представлених у табл. 4, які повертають логічне значення *True* або *False*. Всі вони мають однаковий пріоритет (вищий за пріоритет логічних операторів). Порівняння можуть поєднуватися довільним чином, наприклад, *x < y <= z* є еквівалентно до *x < y* і *y <= z*.

Таблиця 4

Оператори порівняння

Оператор	Опис	True	False
<	менше	*2 < 13*	*2 < 2*
<=	менше рівне	*2 <= 2*	*3 <= 2*
>	більше	*13 > 2*	*2 > 13*
>=	більше рівне	*3 >= 2*	*2 >= 3*
==	еквівалентно	*2 == 2*	*2 == 3*
!=	не еквівалентно	*3 != 2*	*2 != 2*
is	об’єкти ідентичні	*x is x*	*x is ~x*
is not	об’єкти не ідентичні	*x is not ~x*	*x is not x*

Наприклад:

```
>>> 2 < 3
False
>>> 2 > 3
True
>>> x = 5
>>> 2 < x < 10
True
>>> 2 < 3 < 4 < 5 < 6
```

```
True
```

Оператори <, <=, > та >= викликають виключення *TypeError* коли порівнюються комплексне число з іншими вбудованими числовими типами або якщо об'єкти є різних типів, що не можуть бути порівняні.

1.4.5. Оператори присвоювання

Оператори присвоювання призначені для збереження значення в змінній. У мові Python доступні наступні оператори присвоювання:

♦ = – присвоює змінній значення:

```
>>> x = 5; x
5
```

♦ += – збільшує значення змінної на зазначену величину:

```
>>> x = 5; x += 10 # Еквівалентно x = x + 10
>>> x
15
```

♦ -= – зменшує значення змінної на зазначену величину:

```
>>> x = 10; x -= 5 # Еквівалентно x = x - 5
>>> x
5
```

♦ *= – множить значення змінної на зазначену величину:

```
>>> x = 10; x *= 5 # Еквівалентно x = x * 5
>>> x
50
```

♦ /= – ділить значення змінної на зазначену величину:

```
>>> x = 10; x /= 3 # Еквівалентно x = x / 3
>>> x
3.3333333333333335
```

♦ //= – ціла частина від ділення на зазначену величину:

```
>>> x = 10; x //= 3 # Еквівалентно x = x // 3
>>> x
3
```

♦ %= – залишок від ділення на зазначену величину:

```
>>> x = 10; x %= 2 # Еквівалентно x = x % 2
>>> x
0
```

♦ **= – піднесення до зазначеного степеня:

```
>>> x = 10; x **= 2 # Еквівалентно x = x ** 2
>>> x
100
```

1.4.6. Пріоритет виконання операторів

В якій послідовності буде обчислюватися наведений далі вираз?

```
>>> x = 5 + 10 * 3/2
```

Це залежить від пріоритету виконання операторів (operator precedence). У даному випадку послідовність обчислення виразу буде такою:

1. Число 10 буде помножено на 3, так як пріоритет оператора множення вище пріоритету оператора додавання.

2. Отримане значення буде поділено на 2, так як пріоритет оператора ділення дорівнює пріоритету оператора множення (а оператори з рівними пріоритетами виконуються зліва направо – лівоасоціативні), але вище, ніж у

оператора додавання.

3. До отриманого значення буде додано число 5, так як оператор присвоювання = має найменший пріоритет.

4. Значення буде присвоєно змінній *x*.

```
>>> x
20.0
```

За допомогою дужок можна змінити послідовність обчислення виразу:

```
>>> x = (5 + 10) * 3/2
```

Тепер порядок обчислень стане іншим:

1. До числа 5 буде додано 10.
2. Отримане значення буде помножено на 3.
3. Отримане значення буде поділено на 2.
4. Значення буде присвоєно змінній *x*.

```
>>> x
22.5
```

У табл. 5 подані оператори у порядку спадання пріоритету.

Таблиця 5

Пріоритети операторів Python

Оператор	Опис	Коментар
**	Піднесення до степеня	Бінарний оператор
+x, -x, ~x	Унарний плюс, унарний мінус, інверсія бітів	Унарні оператори
**, /, //, %*	Множення, ділення, ціла частина, залишок	Бінарні оператори
+, -	Додавання і віднімання	Бінарні оператори
<<, >>	Зсуви	Бінарні оператори
&	Побітове І	Бінарний оператор
^	Побітове виняткове АБО	Бінарний оператор
|	Побітове АБО	Бінарний оператор
in, not in, is, not is, <, <=, >, >=, !=, ==	Порівняння, включно з перевіркою належності та ідентичності	Бінарні оператори
not x	Логічне НІ	Унарний оператор
and	Логічне І	Бінарний оператор
or	Логічне АБО	Бінарний оператор

Всі бінарні оператори за винятком присвоєння є лівоасоціативні.

Якщо унарні оператори розташовані зліва від оператора **, то піднесення в степінь має більший пріоритет, а якщо справа – то менший. Наприклад, вираз:

```
-10 ** -2
```

еквівалентний наступній розстановці дужок:

```
-(10 ** (-2))
>>> -2 ** 4
-16
>>> -(2 ** 4)
-16
>>> (-2) ** 4
16
```

У випадку сумнівів у порядку обчислень варто позначити пріоритети у вигляді круглих душок.

Порівняння дійсних чисел може привести до несподіваних, на перший погляд, результатів:

```
>>> (0.1 + 0.1) == 0.2
True
>>> (0.1 + 0.1 + 0.1) == 0.3
False
```

1.5. Вбудовані функції для числових типів даних

Інтерпретатор Python має набір вбудованих функцій (built-in functions), які завжди доступні представлені в табл. 6. Розглянемо ті з них, які працюють з числовими типами.

Таблиця 6

Вбудовані функції

abs()	*delattr()*	*hash()*	*memoryview()*	*set()*
all()	*dict()*	*help()*	*min()*	*setattr()*
any()	*dir()*	*hex()*	*next()*	*slice()*
ascii()	*divmod()*	*id()*	*object()*	*sorted()*
bin()	*enumerate()*	*input()*	*oct()*	*staticmethod()*
bool()	*eval()*	*int()*	*open()*	*str()*
breakpoint()	*exec()*	*isinstance()*	*ord()*	*sum()*
bytearray()	*filter()*	*issubclass()*	*pow()*	*super()*
bytes()	*float()*	*iter()*	*print()*	*tuple()*
callable()	*format()*	*len()*	*property()*	*type()*
chr()	*frozenset()*	*list()*	*range()*	*vars()*
classmethod()	*getattr()*	*locals()*	*repr()*	*zip()*
compile()	*globals()*	*map()*	*reversed()*	*__import__()*
complex()	*hasattr()*	*max()*	*round()*	

♦ *abs(x)* – повертає абсолютне значення числа *x*. Для комплексного числа повертає модуль:

```
>>> abs(-3), abs(-2.6), abs(3+4j)
(3, 2.6, 5.0)
```

♦ *round(number[, ndigits])* – повертає *number* заокруглений до *ndigits* цифр після коми. Для чисел з дробовою частиною менше 0.5 повертає число, округлене до найближчого меншого цілого, а для чисел з дробовою частиною більше 0.5 повертає число, округлене до найближчого більшого цілого. Якщо дробова частина дорівнює 0.5, то округлення проводиться до найближчого парного числа. Якщо *ndigits* відсутнє або *None* повертається найближче ціле число. Приклад:

```
>>> round(0.49), round(0.5), round(0.51), round(1.5)
(0, 0, 1, 2)
>>> x = 93.34836
>>> round(x), round(x, 2), round(x, 3), round(x, 0), round(x, -1)
(93, 93.35, 93.348, 93.0, 90.0)
```

♦ *pow(x, y[, z])* – повертає *x* у степені *y*, якщо *z* вказано, повертає *x* у степені *y*

за модулем *z*, обчислене більш ефективно ніж *pow(x, y) % z*. Форма з двома аргументами *pow(x, y)* еквівалентна до оператора піднесення до степені: *x****y*.

```
>>> pow(10, 2), 10 ** 2, pow(3, 3), 3 ** 3
(100, 100, 27, 27)
>>> pow(10, 2, 2), (10 ** 2) % 2, pow(3, 3, 2), (3 ** 3) % 2
(0, 0, 1, 1)
```

♦ *divmod(a, b)* – повертає пару чисел рівних *(a // b, a % b)*.

```
>>> divmod(13, 2)      # 13 = 6 * 2 + 1
(6, 1)
>>> divmod(13.5, 2.0) # 13.5 = 6.0 * 2.0 + 1.5
(6.0, 1.5)
```

♦ *bin(x)* – перетворює ціле число *x* в бінарний рядок з префіксом *"0b"*:

```
>>> bin(255), bin(1), bin(-45)
('0b11111111', '0b1', '-0b101101')
```

Якщо префікс *"0b"* є бажаним або ні, можна поступити наступним чином:

```
>>> format(14, '#b'), format(14, 'b')
('0b1110', '1110')
>>> f'{14:#b}', f'{14:b}'
('0b1110', '1110')
```

♦ *hex(x)* – перетворює ціле число *x* в шістнадцятковий рядок з префіксом *"0x"*:

```
>>> hex(10), hex(16), hex(255),hex(-42)
('0xa', '0x10', '0xff', '-0x2a')
```

Якщо префікс *"0x"* є бажаним або ні, чи потрібні великі шістнадцяткові букви можна поступити наступним чином:

```
>>> '%#x' % 255, '%x' % 255, '%X' % 255
('0xff', 'ff', 'FF')
>>> format(255, '#x'), format(255, 'x'), format(255, 'X')
('0xff', 'ff', 'FF')
>>> f'{255:#x}', f'{255:x}', f'{255:X}'
('0xff', 'ff', 'FF')
```

1.6. Бібліотеки для числових типів даних

Модуль *math* – один з найважливіших в Python. Цей модуль надає додаткові математичні, тригонометричні і логарифмічні функції для роботи з числами, а також стандартні константи. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції *import*:

```
import math
```

Модуль *math* надає наступні стандартні константи:

♦ *pi* – число π:

```
>>> math.pi
3.141592653589793
```

♦ *tau* – число 2π:

```
>>> math.tau
6.283185307179586
```

♦ *e* – число *e*:

```
>>> math.e
2.718281828459045
```

Функції модуля *math* представлені у табл. 7.

Таблиця 7

Функції модуля math

acos()	*cos()*	*floor()*	*isinf()*	*pow()*
acosh()	*cosh()*	*fmod()*	*isnan()*	*radians()*
asin()	*degrees()*	*frexp()*	*ldexp()*	*remainder()*
asinh()	*erf()*	*fsum()*	*lgamma()*	*sin()*
atan()	*erfc()*	*gamma()*	*log()*	*sinh()*
atan2()	*exp()*	*gcd()*	*log10()*	*sqrt()*
atanh()	*expm1()*	*hypot()*	*log1p()*	*tan()*
ceil()	*fabs()*	*isclose()*	*log2()*	*tanh()*
copysign()	*factorial()*	*isfinite()*	*modf()*	*trunc()*

Перелічимо основні функції для роботи з числами:

♦ *sin(), cos(), tan()* – стандартні тригонометричні функції. Значення вказується в радіанах.

♦ *asin(), acos(), atan()* – обернені тригонометричні функції (арксинус, арккосинус, арктангенс). Значення повертається в радіанах.

♦ *degrees()*– перетворює радіани в градуси:

```
>>> math.degrees(math.pi)
180.0
```

♦ *radians()* – перетворює градуси в радіани:

```
>>> math.radians(180.0)
3.141592653589793
```

♦ *exp()* – експонента.

♦ *log(<Число> [, <База>])* – логарифм по заданій базі. Якщо база не вказана, обчислюється натуральний логарифм (по базі *e*).

♦ *log10()* – десятковий логарифм.

♦ *log2()* – логарифм за основою 2.

♦ *sqrt()* - квадратний корінь:

```
>>> math.sqrt(100), math.sqrt(25)
(10.0, 5.0)
```

♦ *ceil()* – значення, округлене до найближчого більшого цілого:

```
>>> math.ceil(5.49), math.ceil(5.50), math.ceil(5.51)
(6, 6, 6)
```

♦ *floor()* – значення, округлене до найближчого меншого цілого:

```
>>> math.floor(5.49), math.floor(5.50), math.floor(5.51)
(5, 5, 5)
```

♦ *pow(<Число>, <Степінь>)* – підносить *<Число>* в *<Степінь>*:

```
>>> math.pow(10, 2), 10 ** 2, math.pow(3, 3), 3 ** 3
(100.0, 100, 27.0, 27)
```

♦ *fabs()* – абсолютне значення.

♦ *factorial()* – факторіал числа:

```
>>> math.factorial(5), math.factorial(6)
(120, 720)
```

Для роботи з комплексними числами необхідно використовувати модуль *cmath*, який містить більшість функцій, які є в модулі *math* (*acos, acosh, asin, asinh, atan, atanh, cos, cosh, exp, log, log10, sin, sinh, sqrt, tan, tanh*), які коректно працюють з комплексними числами та декілька власних функцій (*phase, polar,*

rect). Наприклад:

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
```

2. ЗАВДАННЯ

2.1. Домашня підготовка до роботи

1. Вивчити теоретичний матеріал.

2.2. Виконати в лабораторії

1. Написати програму, яка обчислює значення виразу, заданого в таблиці 2.1. Ввід аргументів здійснюється з клавіатури в діалоговому режимі. Програма повинна виводити номер варіанту, прізвище та ім'я студента, формулу, для обчислення виразу, введенні значення, результат обчислення виразу.

Номер варіанту відповідає номеру в списку групи.

3. ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Повний текст завдання згідно варіанту.
3. Лістинг програми.
4. Результати роботи програм (у текстовій формі та скріншот).
5. Висновок.

Табл. 2.1

ЗАВДАННЯ

Варіант	Вираз	Вивід результату
1	$(x+2)^3+\sqrt[5]{y+x}-\log_3\left(2+\left\|\frac{\sin(x+y)}{\cos(x+z)}\right\|\right)$	Заокруглити до 1 знаку після коми
2	$\ln(x^4+y^5+z^6)-tg(\cos(z))+\sqrt[4]{z+x+y^2}$	Заокруглити до 2 знаків після коми
3	$\log_5(\sin(x)+y^2)+\sqrt[3]{89y-x}-tg(xy)$	Заокруглити до 3 знаків після коми
4	$24.6\sin(z+y-x)+\sqrt[4]{89x^3+y}-tg(z^2+x)$	Заокруглити до 4 знаків після коми
5	$tg\left(\frac{2x+z}{y+4.5}\right)+x^{\sin(y)}-\sqrt[6]{\ln(2+x^2)}$	Заокруглити до меншого цілого
6	$2\cos(y)+\sqrt{\sin^4(x)}+\log_4\left(1+\left\|\frac{\sin(y)}{x^4-y^4}\right\|\right)$	Заокруглити до більшого цілого
7	$\sqrt[8]{\left\|x+tg^z(y^3)\right\|}+e^{x-z}-\log_5(\left\|x-y\right\|+3)$	Заокруглити до 1 знаку після коми
8	$\frac{\sin(z+x)}{\cos(y-z)}+(x+y)^z-\sin(\ln(1+\left\|x\right\|))$	Заокруглити до 2 знаків після коми
9	$x^{\cos(y)}-(tg(x))^{\sqrt{\left\|y-x\right\|}}-19.12\ln(1+y^6)$	Заокруглити до 3 знаків після коми
10	$\sqrt{\sin(x^2)+16yx}-e^{x+y}-\frac{1}{\cos(y)+x}$	Заокруглити до 4 знаків після коми

11	$137x^3+\cos(y^3/x^4)+tg(14y)-7x^6$	Заокруглити до меншого цілого
12	$(x+1)^2+\frac{2(y+z)}{x+y-z^2}+13\log_5(\cos(xy)+z)$	Заокруглити до більшого цілого
13	$\frac{xy+x^2y^2}{tg(x^5z^5)}-\cos(xy)+x\sqrt[6]{z^5}+173.11x$	Заокруглити до 1 знаку після коми
14	$16y^2+e^{yz}+\sqrt[3]{z+1.51+\sin(x)}+\ln(yz)$	Заокруглити до 2 знаків після коми
15	$3.14x^2-7y^{-2}+z\sin(y^3)+7\sqrt{z}+\ln(x+7\sqrt{y})$	Заокруглити до 3 знаків після коми
16	$\ln(x^3)+tg(z)+(y+x)^3-\frac{x\sin(y+1.6)}{177z^2}$	Заокруглити до 4 знаків після коми
17	$\sqrt{xy^2+y\sin(z)+142x^2y}+tg(xy)-\frac{142(z-x)}{16.32}$	Заокруглити до меншого цілого
18	$\sqrt{\lvert tg(x)\rvert}+(x^2+y)^{\ln(z)}$	Заокруглити до більшого цілого
19	$\ln(x^2+4xy+z^2)-12\cos(xy^4)+13x^6$	Заокруглити до 1 знаку після коми
20	$\sin(x^2)+3\cos(2x+\ln(z))-14(x^2+y)^3$	Заокруглити до 2 знаків після коми
21	$tg(\sqrt{x})+165z^6+\ln(\sqrt[4]{x^2-z})$	Заокруглити до 3 знаків після коми
22	$13zy+\ln(x/y)-\sqrt{17\sin(x^2-y+\sin(z))}$	Заокруглити до 4 знаків після коми
23	$y\sqrt[3]{xy}+y\sin(zy)-e^{-xy}\cos(z^2)+\log_7\left(\frac{7.1}{xy}\right)$	Заокруглити до меншого цілого
24	$3.6(\sin(x)+\cos(y^2))^3-\ln\left(\sqrt{\sin^2(z)+\cos^2(y)}\right)$	Заокруглити до більшого цілого
25	$4\sqrt[z]{xy}+y\sin^3(zy)-e^{-xy}\cos^3(z^2)+\log_2(x+z)$	Заокруглити до 1 знаку після коми

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які об’єкти повертають значення True ?
2. Який діапазон чисел можуть представляти типи int та float ?
3. Яким чином у Python реалізований пріоритет виконання операторів ?
4. Запишіть в один рядок за правилами мови Python наступні арифметичні вирази:

а) $\frac{-1}{x^2}$; б) $\frac{a}{bc}$; в) $\frac{(a+b)}{2}$; г) $\frac{-b+\sqrt{b^2-4ac}}{2a}$; д) $\frac{-b+\frac{1}{a}}{\frac{2}{c}}$; е) $\frac{1}{1+\frac{a+b}{2}}$.

5. СПИСОК ЛІТЕРАТУРИ

1. Learn to Program with Python 3. A Step-by-Step Guide to Programming, Second Edition / Irv Kalb. – Mountain View: Apress, 2018. – 361 p.
2. The Python Workbook. A Brief Introduction with Exercises and Solutions, Second Edition / Ben Stephenson. – Cham: Springer, 2014. – 218 p.
3. Python Pocket Reference, Fifth Edition / Mark Lutz. – Sebastopol: O’Reilly Media, Inc., 2014. – 264 p.
4. Learn Python 3 the Hard Way / Zed A. Shaw. – Boston: Addison-Wesley, 2017. – 321 p.
5. A Python Book: Beginning Python, Advanced Python, and Python Exercises / Dave Kuhlman. – Boston: MIT, 2013. – 278 p.

НАВЧАЛЬНЕ ВИДАННЯ

Робота з логічними та числовими типами даних

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи № 2
з курсу «Програмування скриптовими мовами»
для студентів спеціальності
«Кібербезпека»

Укладач: Я. Р. Совин, канд. техн. наук, доцент