Загальна характеристика мови Python.

Python - інтерпретована, об'єктно-орієнтована мова програмування високого рівня з динамічною типізацією, автоматичним управлінням пам'яттю і зручними високорівневими структурами даних.

Мову програмування Python створив в 1991 році голландець Гвідо ван Россум (Guido van Rossum). Офіційний сайт мови: http://python.org. Python названий на честь британського серіалу 70-х "Літаючий цирк Монті Пайтона". Python впевнено входить до п'ятірки найпопулярніших комп'ютерних мов. На Python написані програми, якими ви користуєтесь щодня, такі як Google, YouTube, Instagram, Netflix. Python - найпопулярніша мова для навчальних курсів інформатики у провідних американських коледжах, офіційна мова навчання для вузів у Франції.

Останнім часом Python став надзвичайно популярним у світі штучного інтелекту, науки про дані (data science) та машинного навчання (machine learning). Якщо ви хочете отримати добре оплачувану роботу з програмування в цікавій сфері, зараз Python є хорошим вибором.

Python працює майже скрізь і має "batteries included" (батареї в комплекті) - корисні програми у своїй стандартній бібліотеці.

Історично склалися дві версій мови: Python 2 і Python 3. Python 2 - це минуле, а Python 3 - майбутнє. Підтримка мови Python 2 припинилася у січні 2020 року. Ми будемо вивчати Python 3.

Встановлення Python.

Для завантаження останньої версії Python 3 відвідайте сторінку сайту за адресою www.python.org/downloads. Розглянемо інсталяцію на операційну систему Windows. Після того, як ви завантажили інсталятор Python, його слід виконати. Настійно рекомендується на першій сторінці вибрати обидва пункти внизу екрана: *Install launcher for all users* та *Add Python 3.x to PATH*. Потім слід натиснути кнопку *Customize installation*, а в наступному діалоговому вікні, де пропонується вибрати компоненти для установки, треба залишити всі вибрані прапорці і натиснути *Next*.

На наступному кроці потрібно задати деякі додаткові налаштування і вибрати шлях установки. Перевірте, чи встановлені прапорці *Associate files with Python, Create shortcuts for installed applications, Add Python to environment variables, Precompile standard library*.

Тепер уточнимо шлях, по якому буде встановлено Python. Рекомендуємо задати *C:\Python39*, тобто безпосередньо в корінь диску. В цьому випадку ми уникнемо проблем при установці додаткових бібліотек.

Задавши всі необхідні параметри, натискаємо кнопку *Install* і позитивно відповідаємо на попередження UAC що з'явилося на екрані. Після завершення установки відкриється вікно, в якому слід натиснути кнопку *Close* для виходу з програми установки. Після успішної установки інтерпретатора можна приступати до написання коду на Python.

Встановлення редактора Visual Studio Code

Інсталюйте редактор Visual Studio Code (https://code.visualstudio.com/). Він нам знадобиться для виконання лабораторних робіт і вправ даного навчального курсу. Після завершення установки запустіть редактор і встановіть плагін від фірми Microsoft для кращої підтримки мови Python. Для цього на панелі зліва виберіть піктограму з підказкою Extensions, в рядку пошуку введіть “Python”, в випадаючому списку, що з’явиться, виберіть самий верхній пункт “Python extension for Visual Studio Code” від Microsoft і натисніть Install.

Запуск Python.

З встановленим інтерпретатором Python можна працювати в двох режимах: в *інтерактивному режимі*, в якому кожна інструкція Python подається і зразу виконується, і в *режимі виконання скриптів*, тобто програм на мові Python, які знаходяться в файлах з розширенням *.py*. Ми будемо в основному використовувати другий режим. Для цього нам буде потрібен або текстовий редактор, або IDLE (Integrated Development and Learning Environment).

IDLE - це інтегроване середовище розробки, яке об'єднує кілька інструментів розробки в одну програму, включаючи наступне:

- Оболонка Python (Python shell), яка працює в інтерактивному режимі. Ви можете ввести оператори Python у підказці оболонки і негайно їх виконати. Ви також можете запускати повні програми Python.
- Текстовий редактор, колір якого кодує ключові слова Python та інші частини програм.
- Інструмент “check module”, який перевіряє програму Python на наявність синтаксичних помилок без запуску програми.
- Інструменти пошуку, які дозволяють знаходити текст в одному або кількох файлах.
- Інструменти форматування тексту, які допомагають підтримувати послідовні рівні відступу в програмі Python.
- Налагоджувач, який дозволяє проходити покроково програму Python і спостерігати за змінами змінних під час виконання кожного оператора.
- Кілька інших додаткових інструментів для розробників.

Програмне забезпечення IDLE поставляється в комплекті з Python. Коли ви встановлюєте інтерпретатор Python, автоматично встановлюється і IDLE. Після встановлення Python у вашій системі група програм Python з’явиться у списку програм меню Пуск. Один із пунктів у групі програм матиме назву IDLE (Python GUI). Натисніть цей пункт, щоб запустити IDLE, і ви побачите вікно оболонки Python. Підказка >>> вказує на те, що інтерпретатор чекає, поки ви введете оператор Python.

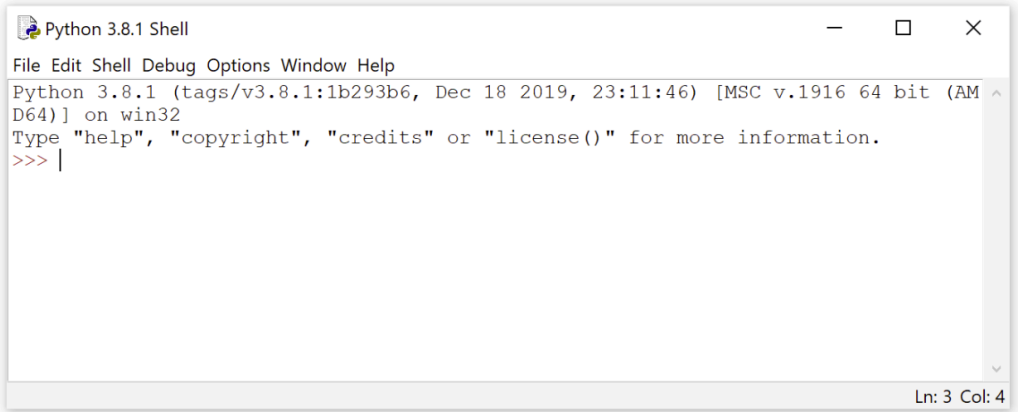

Рис. 1 Так виглядає Python shell (оболонка Python)

Оболонка Python - дуже зручний інструмент для тестування команд, особливо на перших етапах роботи. Але якщо ви вийдете з оболонки і ввійдете в неї знову, то всі команди, що були введені, зникнуть. Крім того, оболонку не можна використовувати для створення реальної програми. Щоб створити справжню програму, потрібно написати код в текстовому файлі і зберегти його з розширенням .py. Такий файл називається сценарієм Python. Щоб створити сценарій Python треба натиснути File => New File у верхньому меню оболонки Python. З'явиться текстовий редактор, тощо.

Але в данному навчальному курсі рекомендованим способом створення програм (сценаріїв) Python є використання текстового редактора Visual Studio Code.

Базові типи даних Python

Таблиця 3 показує основні типи даних у Python. Другий стовпець (Type) містить ім'я Python цього типу. Третій стовпець (Mutable?) Вказує, чи можна змінити значення після створення. Examples показують один або кілька прикладів літералів такого типу.

Таблиця 3. Основні типи даних Python

Name	Type	Mutable?	Examples
Boolean	bool	no	True, False
Integer	int	no	47, 25000, 25_000
Floating point	float	no	3.14, 2.7e5
Complex	complex	no	3j, 5 + 9j
Text string	str	no	'alas', "alack", '''a verse attack'''
List	list	yes	['Winken', 'Blinken', 'Nod']
Tuple	tuple	no	(2, 4, 8)
Bytes	bytes	no	b'ab\xff'
ByteArray	bytearray	yes	bytearray(...)
Set	set	yes	set([3, 5, 7])
Frozen set	frozenset	no	frozenset(['Elsa', 'Otto'])
Dictionary	dict	yes	{'game': 'bingo', 'dog': 'dingo', 'drummer': 'Ringo'}

У Python, якщо ви хочете знати тип чого-небудь (змінна або значення літерала), ви можете використовувати type(thing). type()-одна з вбудованих функцій Python. Якщо ви хочете перевірити, чи вказує змінна на об'єкт певного типу, використовуйте isinstance (type):

```
>>> type(7)
<class 'int'>
>>> type(7) == int
True
>>> isinstance(7, int)
True
```

Ідентифікатори Python

Ідентифікатор Python використовується для позначення змінної, класу, функції, модуля або будь-якого іншого об'єкта. Python чутливий до регістру, тому великі та малі літери вважаються різними.

Правила декларування ідентифікатора:

- Єдиними дозволеними символами є літери, цифри та символ підкреслення (_).
- Ідентифікатор не повинен починатися з цифри.
- Ідентифікатори за своєю природою чутливі до регістру.
- Якщо ідентифікатор починається з підкреслення (_), то це приватний ідентифікатор.
- Символ долара ($) заборонений у Python.

Зарезервовані ключові слова

Це ключові слова, зарезервовані мовою програмування, і не дозволяють користувачеві або програмісту використовувати їх як ідентифікатор у програмі.

Існує 33 зарезервовані ключові слова (залежить від версії), як показано в Таблиці 4. Отримати список ключових слів можна подавши команду

```
>>> help("keywords")
```

Таблиця 4. Зарезервовані ключові слова Python

Reserved keywords				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
And	del	global	not	with
As	elif	if	or	yield
Assert	else	import	pass	
Break	except	in	raise	

Літерали

Літерали - це числа, рядки або символи, які з'являються безпосередньо в програмі. Мова програмування містить дані з точки зору введення та виведення, а будь-який вид даних може бути представлений з точки зору значення. Значення може бути будь-якого типу, наприклад літерали, що містять цифри, символи та рядки. Щоб знати тип значення в Python, існує вбудована функція під назвою type.

```
Syntax: type(value)
```

Приклади літералів:

20

25.5

'A'

"PYTHON"

Рис. 2 демонструє використання функції `type` для літералів.

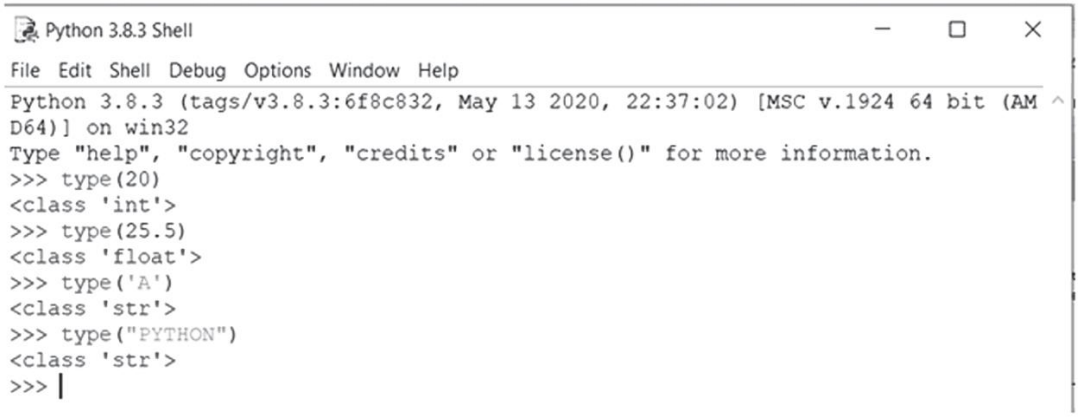

Рис. 2 Застосування функції `type` для літералів

Введення даних

Програми Python можуть читати введення з клавіатури, викликаючи функцію *input*. Ця функція змушує програму зупинятися і чекати, поки користувач

щось набере. Коли користувач натискає клавішу Enter, символи, введені користувачем, повертаються функцією введення. Потім програма продовжує виконання. Вхідні значення зазвичай зберігаються у змінній за допомогою оператора присвоєння, щоб їх можна було використовувати пізніше в програмі. Функція input() може приймати необов'язковий аргумент-запрошення строкового типу; при виконанні функції повідомлення буде з'являтися на екрані і інформувати користувача про запитувані дані. Наприклад, наступний вираз читає значення, введене користувачем, і зберігає його у змінній з ім'ям a.

```
a = input("Введіть значення змінної a ")
```

Функція *input* завжди повертає рядок, тобто послідовність символів. Якщо значення, яке читається, - це ім'я людини, назва книги чи назва вулиці, то доцільно зберігати значення як рядок. Але якщо значення є числовим, наприклад, вік, температура або вартість страви в ресторані, то рядок, введений користувачем, зазвичай перетворюється на число. Перетворення в ціле число виконується шляхом виклику функції *int*, тоді як перетворення в число з плаваючою комою здійснюється шляхом виклику функції *float*.

```
n1 = int(input("Enter n1 : "))
n2 = int(input("Enter n2 : "))
sum = n1 + n2
print("n1 + n2 = ", sum)
```

Виведення даних

Вивести значення на екран можна за допомогою функції *print*.

```
print("When x is", x, "the value of y is", y)
```

Функція print () має кілька "прихованих" аргументів, які задаються за замовчуванням в момент виклику:

```
>>>help(print)
. . . . .
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False),
```

де *value* – перераховуються через кому об'єкти, які необхідно вивести на екран; *sep* – рядок-розділювач між об'єктами (за замовчуванням стоїть пробіл); *end* – рядок, що розміщений після останнього об'єкту (за замовчуванням – перехід на новий рядок).

Отже, функція print() додає пробіл між кожним виведеним об'єктом, а також символ нового рядка в кінці:

```
>>> print(1, 2, 3, 4, 5)
1 2 3 4 5
>>> print(1, 6, 7, 8, 9, sep=':')
1:6:7:8:9
```

Основні принципи синтаксису мови Python

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

```
x = 5
print(2 + x)
```

Вкладені інструкції об'єднуються у блоки за величиною відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий (рекомендується робити відступ 4 пробіли).

```
x = 1
if x:
    y = 2
    if y:
        print('block2')
    print('block1')
print('block0')
```

Вкладені інструкції в Python записуються відповідно до одного і того ж шаблону, коли основна інструкція завершується двокрапкою, слідом за чим розташовується вкладений блок коду, зазвичай з відступом під рядком основної інструкції.

Числові типи даних

Цілі числа (Integer Numbers)

Ціле число - це комбінація додатніх і від'ємних чисел, включаючи (нуль) 0. У програмі цілочисельні літерали записуються без коми та з провідним знаком мінус для позначення від'ємного значення.

Приклад:

```
a = 10
print(a)
print(type(a))
10
<class 'int'>
```

Цілі значення відображаються такими способами:

A) Десяткова форма (основа 10)

Система числення за замовчуванням; дозволені значення 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

B) Двійкова форма (Base-2)

Дозволені значення - 0 і 1; літеральне значення має префікс 0b або 0B

C) вісімкова форма (Base-8)

Дозволені значення 0, 1, 2, 3, 4, 5, 6, 7; літеральні значення мають префікс 0o або 0O.

D) Шістнадцяткова форма (Base-16)

Дозволені цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a – f; літеральні значення повинні мати префікс 0x або 0X

Приклад:

```
p = 100
q = 0o100
r = 0X10c
s = 0B1010
print(p)
print(q)
print®
print(s)
```

```
100
64
268
10
```

Цілочисельні операції

Operator	Description	Example	Result
+	Addition	5 + 8	13
-	Subtraction	90 - 10	80
*	Multiplication	4 * 7	28
/	Floating-point division	7 / 2	3.5
//	Integer (truncating) division	7 // 2	3
%	Modulus (remainder)	7 % 3	1
**	Exponentiation	3 ** 4	81

Числа з плаваючою точкою

Числа з плаваючою точкою складаються з цілого числа, десяткової точки і дробової частини. Довжина дійсних чисел може бути нескінченною, тобто кількість цифр у дробовій частині не обмежена. Таким чином, Python використовує числа з плаваючою точкою для представлення дійсних чисел.

Приклад:

```
f = 13.22
print(type(f))
f1 = 13.22e4
print(f1)
<class 'float'>
132200.0
```

Комплексні числа

Комплексні числа виражаються у вигляді (a + bj), де a і b - дійсні числа, а j - уявна частина.

Приклад:

```
c = 30+2.6j
print(c)
30+2.6j
print(type(c))
<class 'complex'>
```

Boolean Type

У Python булевий тип даних представлений як bool. Це примітивний тип даних зі значеннями True або False. Значення True представлено як 1, а False - як 0. Крім того, якщо додати два True, воно видасть 2 як результат (True + True = 2), тоді як коли False віднято від True, воно видасть 1 як вихід (True - False = 1).

Приклад:

```
b = True
print(type(b))
<class 'bool'>
```

Рядки

У Python 3 рядок (тип str) - це незмінна послідовність, що зберігає символи Unicode. Рядки можуть міститися як в одиночних, так і в подвійних лапках. Однак, початок і кінець рядка повинен відкриватися та закриватися однаковим типом лапок. Метою використання двох видів лапок є можливість створювати рядки, що містять лапки чи апострофи. Отже, усередині одинарних лапок можна розташувати подвійні та навпаки. Можна також використовувати три одинарні або три подвійні лапки, наприклад, щоб створити багаторядкові рядки.

У мові Python немає окремого символьного типу. Символ – це просто рядок довжини 1.

Базові операції

- Конкатенація (додавання)

```
>>>
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print(S1 + S2)
'spameggs'
```

- Дублювання рядка

```
>>>
>>> print('spam' * 3)
```

spamspamspam

- Довжина рядка

```
>>>
>>> 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 негативного індексу, при цьому відлік йде від кінця рядка.

- Отримання зрізу

Оператор вилучення зрізу: [X: Y]. X - початок зрізу, а Y - закінчення; символ з номером Y в зріз не входить. За умовчанням 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'
```

Приклад. Визначити, чи має ім’я файлу розширення .py

```
if s[len(s)-3:len(s)] == '.py':
    print("This is a Python program.")
```

f-рядок

Новим для Python 3.6 функціоналом є форматоване рядкове значення, або f-рядок, який забезпечує простий спосіб підстановки значень в послідовності символів.

```
>>> name = 'Johnny'
>>> print(f'Hello {name}.')
Hello Johnny.
```

Методи рядків

S.find(str, [start],[end]) - Пошук підрядка в рядку. Повертає номер першого входження або -1.

S.replace(шаблон, заміна) - Заміна шаблону.

S.split(символ) - Розбиття рядка по роздільнику.

S.join(список) - Збірка рядка зі списку з роздільником S.

S.isdigit() - Чи складається рядок з цифр.

S.islower() - Чи складається рядок із символів в нижньому регістрі.

S.upper() - Перетворення рядка до верхнього регістру.

S.startswith(str) - Чи починається рядок S з шаблону str.

S.count(str, [start],[end]) - Повертає кількість неперетинаючихся входжень підрядка в діапазоні [початок, кінець] (0 і довжина рядка за замовчуванням).

S.lstrip([chars]) - Видалення символів пробілів на початку рядка.

S.rstrip([chars]) - Видалення символів пробілів в кінці рядка.

S.strip([chars]) - Видалення символів пробілів на початку і в кінці рядка.

S.format(*args, **kwargs) - Форматування рядка.

Більш детальну інформацію про ці та інші методи рядків дивись в документації (https://docs.python.org/3/library/stdtypes.html#string-methods).

Форматування

Ви бачили, що можна об’єднати рядки за допомогою +. Давайте розглянемо, як інтерполювати значення даних у рядки за допомогою різних форматів. Ви можете використовувати це для створення звітів, форм та інших результатів, де вигляд тексту має значення.

Крім функцій у попередньому розділі, Python має три способи форматування рядків:

- Old style (старий стиль, підтримується в Python 2 і 3);
- New style (новий стиль, - Python 2.6 і вище);
- f-рядки (Python 3.6 і вище).

Old style: %

Старий стиль форматування рядків має форму `format_string % data`. Усередині рядка форматування є послідовності інтерполяції. Таблиця 5 ілюструє, що найпростіша послідовність - це %, за яким слідує буква, що вказує тип даних для форматування.

Таблиця 5. Типи перетворення

%s	string
%d	decimal integer
%x	hex integer
%o	octal integer
%f	decimal float
%e	exponential float
%g	decimal or exponential float
%%	a literal %

Ви можете використовувати %s для будь-якого типу даних, і Python відформатує його як рядок без зайвих пробілів. Нижче наведено кілька простих прикладів. Спочатку ціле число:

```
>>> '%s' % 42
'42'
>>> '%d' % 42
'42'
>>> '%x' % 42
'2a'
>>> '%o' % 42
'52'
```

Число з плаваючою точкою:

```
>>> '%s' % 7.03
'7.03'
>>> '%f' % 7.03
'7.030000'
>>> '%e' % 7.03
'7.030000e+00'
>>> '%g' % 7.03
'7.03'
```

Ціле число і літерал %:

```
>>> '%d%%' % 100
'100%'
```

Давайте спробуємо деяку рядкову та цілочисельну інтерполяцію:

```
>>> actor = 'Richard Gere'
>>> cat = 'Chester'
>>> weight = 28
>>> "My wife's favorite actor is %s" % actor
"My wife's favorite actor is Richard Gere"
>>> "Our cat %s weighs %s pounds" % (cat, weight)
'Our cat Chester weighs 28 pounds'
```

Ви можете додати інші значення у рядку форматування між % та специфікатором типу для позначення мінімальної та максимальної ширини, вирівнювання та заповнення символів. Давайте коротко розглянемо ці значення:

- Початковий символ "%".
- Необов'язковий символ вирівнювання: нічого або '+' означає вирівнювання по правому краю, а '-' означає вирівнювання по лівому краю.
- Необов'язкову ширину поля minwidth для використання.
- Необов'язковий "." символ для розділення minwidth і maxchars.
- Необов’язковий параметр maxchars (якщо тип конвертації s), який вказує, скільки символів надрукувати зі значення даних. Якщо тип перетворення f, це вказує точність (скільки цифр надрукувати після десяткової коми).
- Символ типу перетворення з попередньої таблиці.

Це все дещо заплутано, тому ось кілька прикладів для рядка форматування:

```
>>> thing = 'woodchuck'
>>> '%s' % thing
'woodchuck'
>>> '%12s' % thing
'   woodchuck'
>>> '%+12s' % thing
'   woodchuck'
>>> '%-12s' % thing
'woodchuck   '
>>> '%.3s' % thing
'woo'
>>> '%12.3s' % thing
'         woo'
>>> '%-12.3s' % thing
'woo         '
```

Варіанти форматування для дійсних чисел:

```
>>> thing = 98.6
>>> '%f' % thing
'98.600000'
>>> '%12f' % thing
'   98.600000'
>>> '%+12f' % thing
'  +98.600000'
>>> '%-12f' % thing
```

```
'98.600000   '
>>> '%.3f' % thing
'98.600'
>>> '%12.3f' % thing
'      98.600'
>>> '%-12.3f' % thing
'98.600      '
```

Варіанти форматування для цілих чисел:

```
>>> thing = 9876
>>> '%d' % thing
'9876'
>>> '%12d' % thing
'        9876'
>>> '%+12d' % thing
'       +9876'
>>> '%-12d' % thing
'9876        '
>>> '%.3d' % thing
'9876'
>>> '%12.3d' % thing
'        9876'
>>> '%-12.3d' % thing
'9876        '
```

Для цілого числа %+12d просто змушує друкувати знак, а рядки форматування з .3 не мають для цілих жодного ефекту, вони працюють для float.

New style: {} and format()

Форматування старого стилю все ще підтримується у Python 2. Для Python 3 використовуйте форматування “нового стилю”, описане в цьому розділі. Якщо у вас Python 3.6 або новіша версія, f-рядки ще кращі.

“New style” форматування має форму **format_string.format(data)**.

Рядок формату не зовсім такий, як у попередньому розділі. Найпростіше використання показано тут:

```
>>> thing = 'woodchuck'
>>> '{}'.format(thing)
'woodchuck'
```

Аргументи функції format () мають бути в порядку як заповнювачі {} у рядку формату:

```
>>> thing = 'woodchuck'
>>> place = 'lake'
>>> 'The {} is in the {}.'.format(thing, place)
'The woodchuck is in the lake.'
```

За допомогою форматування в новому стилі ви також можете вказати аргументи за позицією, наприклад:

```
>>> 'The {1} is in the {0}.'.format(place, thing)
'The woodchuck is in the lake.'
```

Значення 0 відноситься до першого аргументу, place, а 1 - до thing.

Аргументи до format () також можуть бути іменованими

```
>>> 'The {thing} is in the {place}'.format(thing='duck',
place='bathtub')
'The duck is in the bathtub'
```

або словником:

```
>>> d = {'thing': 'duck', 'place': 'bathtub'}
```

У наступному прикладі {0} є першим аргументом format () (словник d):

```
>>> 'The {0[thing]} is in the {0[place]}.'.format(d)
'The duck is in the bathtub.'
```

Усі ці приклади друкували свої аргументи у форматах за замовчуванням. Форматування нового стилю має дещо інше визначення формату рядка від старого стилю (наведені приклади):

• Початкова двокрапка (':').

• Необов'язковий символ заповнення (за замовчуванням ' ') для вставки рядка значення, якщо він коротший за minwidth.

• Додатковий символ вирівнювання. Вирівнювання вліво - за замовчуванням. '<' також означає ліворуч, '>' означає праворуч, а '^' означає центр.

• Додатковий знак для цифр. Знак мінус для від'ємних чисел і пробіл (' ') для додатних.

• Необов'язкова minwidth. Необов'язковий період ('.') Для розділення minwidth та maxchars.

• Необов'язкова maxchars.

• Тип перетворення.

```
>>> thing = 'wraith'
>>> place = 'window'
>>> 'The {} is at the {}'.format(thing, place)
'The wraith is at the window'
>>> 'The {:10s} is at the {:10s}'.format(thing, place)
'The wraith     is at the window    '
>>> 'The {:<10s} is at the {:<10s}'.format(thing, place)
'The wraith     is at the window    '
>>> 'The {:^10s} is at the {:^10s}'.format(thing, place)
'The   wraith   is at the   window  '
>>> 'The {:>10s} is at the {:>10s}'.format(thing, place)
'The     wraith is at the     window'
>>> 'The {:!^10s} is at the {:!^10s}'.format(thing, place)
'The !!wraith!! is at the !!window!!'
```

Newest Style: f-strings

f-рядки з'явилися в Python 3.6 і тепер є рекомендованим способом форматування рядків.

Щоб створити f-рядок:

• Введіть букву f або F безпосередньо перед початковою лапкою.

• Додайте імена змінних або вирази до фігурних дужок ({}), щоб отримати їх значення у рядку.

Це схоже на форматування "нового стилю" попереднього розділу, але без функції format () і без порожніх дужок ({}) або позиційних ({1}) у рядку форматування.

```
>>> thing = 'wereduck'
>>> place = 'werepond'
>>> f'The {thing} is in the {place}'
'The wereduck is in the werepond'
```

Вирази також допускаються всередині фігурних дужок:

```
>>> f'The {thing.capitalize()} is in the {place.rjust(20)}'
'The Wereduck is in the              werepond'
```

f-рядки використовують таку саму мову форматування (ширину, відступ, вирівнювання), що й форматування нового стилю, після ':'.

```
>>> f'The {thing:>20} is in the {place:.^20}'
'The             wereduck is in the ......werepond......'
```

Починаючи з Python 3.8, f-рядки отримують новий можливість, яка стане в нагоді, коли потрібно надрукувати імена змінних, а також їх значення. Це зручно під час налагодження. Хитрість полягає в тому, щоб мати = після імені у закритій частині рядка f: {}:

```
>>> f'{thing =}, {place =}'
thing = 'wereduck', place = 'werepond'
```

Кортежі (Tuples)

Більшість комп'ютерних мов може представляти послідовність елементів, індексованих їх цілочисельним положенням: перший, другий і так далі до останнього. Ви вже бачили рядки Python, які є послідовністю символів.

Python має ще дві структури послідовностей: кортежі та списки. Вони містять нуль або більше елементів. На відміну від рядків, елементи можуть бути різних типів. Фактично, кожен елемент може бути будь-яким об'єктом Python. Це дозволяє створювати такі глибокі та складні структури, як вам подобається.

Чому Python містить і списки, і кортежі? *Кортежі незмінні (immutable);* коли ви призначаєте елементи (лише один раз) кортежу, вони фіксуються і не можуть бути змінені. *Списки є змінними (mutable),* тобто ви можете вставляти та видаляти елементи.

Створення за допомогою ком та ()

Синтаксис створення кортежів трохи суперечливий, що демонструють наступні приклади. Почнемо зі створення порожнього кортежа за допомогою ():

```
>>> empty_tuple = ()
>>> empty_tuple
()
```

Щоб створити кортеж з одним або кількома елементами, поставте після кожного елемента кому. Це працює для одноелементних кортежів:

```
>>> one_marx = 'Groucho',
>>> one_marx
('Groucho',)
```

Ви можете укласти їх у дужки і при цьому отримати той самий кортеж:

```
>>> one_marx = ('Groucho',)
>>> one_marx
('Groucho',)
```

Ось трохи незрозуміло: якщо у вас є один елемент в дужках і пропущена кома, ви не отримаєте кортеж, а лише елемент (у цьому прикладі рядок "Groucho"):

```
>>> one_marx = ('Groucho')
>>> one_marx
'Groucho'
>>> type(one_marx)
<class 'str'>
```

Якщо у вас більше одного елемента, поставте всі коми, крім останньої:

```
>>> marx_tuple = 'Groucho', 'Chico', 'Harpo'
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

Python включає дужки при відображенні кортежу. Вони часто вам не потрібні, коли ви визначаєте кортеж, але використання дужок трохи безпечніше, і це допомагає зробити кортеж більш помітним:

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> marx_tuple
('Groucho', 'Chico', 'Harpo')
```

Кортежі дозволяють призначати кілька змінних одночасно:

```
>>> marx_tuple = ('Groucho', 'Chico', 'Harpo')
>>> a, b, c = marx_tuple
>>> a
'Groucho'
>>> b
'Chico'
>>> c
'Harpo'
```

Іноді це називають розпаковуванням кортежів (tuple unpacking).

Створення за допомогою tuple()

Функція перетворення tuple() робить кортежі з інших речей:

```
>>> marx_list = ['Groucho', 'Chico', 'Harpo']
>>> tuple(marx_list)
('Groucho', 'Chico', 'Harpo')
```

Об'єднання кортежів за допомогою +

```
>>> ('Groucho',) + ('Chico', 'Harpo')
('Groucho', 'Chico', 'Harpo')
```

Дублювання елементів за допомогою *

```
>>> ('yada',) * 3
('yada', 'yada', 'yada')
```

Порівняння кортежів

```
>>> a = (7, 2)
>>> b = (7, 2, 9)
>>> a == b
False
>>> a <= b
True
>>> a < b
True
```

Ітерація за допомогою for - in

```
>>> words = ('fresh','out', 'of', 'ideas')
>>> for word in words:
...     print(word)
...
fresh
out
of
ideas
```

Змінення кортежу

Ви це не можете! Як і рядки, кортежі є незмінними, тому ви не можете змінити існуючий. Як ви бачили нещодавно, ви можете об'єднати кортежі, щоб створити новий, як це можна за допомогою рядків:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop,')
>>> t1 + t2
('Fee', 'Fie', 'Foe', 'Flop')
```

Це означає, що ви можете змінити кортеж таким чином:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop,')
>>> t1 += t2
>>> t1
('Fee', 'Fie', 'Foe', 'Flop')
```

Але це не той самий t1. Python створив новий кортеж з оригінальних кортежів, на які вказують t1 і t2, і присвоїв цьому новому кортежу назву t1. Ви можете побачити за допомогою id (), коли назва змінної вказує на нове значення:

```
>>> t1 = ('Fee', 'Fie', 'Foe')
>>> t2 = ('Flop',)
>>> id(t1)
4365405712
>>> t1 += t2
>>> id(t1)
4364770744
```

Списки (Lists)

Списки хороші для відстеження елементів колекцій за їхнім порядком, особливо коли порядок і зміст можуть змінитися. На відміну від рядків, списки змінюються. Ви можете змінити наявний список, додати нові елементи та видалити або замінити наявні елементи. Одне і те ж значення може зустрічатися у списку кілька разів.

У Python замість масивів використовуються списки, словники, кортежі. Багатомірний масив реалізується за допомогою списку списків (вкладеного списку). За необхідності використовувати саме масиви, наприклад, при обробці великого об’єму даних, слід застосовувати бібліотеку NumPy, яка дозволяє з ними працювати [2].

Список в Python - це упорядкований набір об'єктів, в список можуть одночасно входити об'єкти різних типів. Список складається з нуля або більше елементів, розділених комами та оточених квадратними дужками:

```
lst = [12, ’b’, 34.6, ’derevo’]
empty_list = [ ]
weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday']
first_names = ['Graham', 'John', 'Terry', 'Terry', 'Michael']
```

Функція list() перетворює інші ітерабельні типи даних (такі як кортежі, рядки, множини та словники) у списки. Наступний приклад перетворює рядок у список односимвольних рядків:

```
>>> list('cat')
['c', 'a', 't']
```

Список створює метод split() рядка:

```
>>> day = '9/19/2019'
>>> day.split('/')
['9', '19', '2019']
```

Отримання елементів списку по індексу:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
'Groucho'
>>> marxes[1]
'Chico'
>>> marxes[2]
'Harpo'
```

Знову ж таки, як і у рядках, від’ємні індекси відлічуються назад від кінця:

```
>>> marxes[-1]
'Harpo'
>>> marxes[-2]
'Chico'
```

Отримання елементів за допомогою зрізів.

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0:2]
['Groucho', 'Chico']
```

Як і у випадку рядків, крок для зрізів може бути відмінним від одиниці. В наступному прикладі зріз починається з початку і йде з кроком 2:

```
>>> marxes[::2]
['Groucho', 'Harpo']
```

Тут ми починаємо з кінця і йдемо ліворуч через 2:

```
>>> marxes[::-2]
['Harpo', 'Groucho']
```

І, нарешті, хитрість реверсувати список:

```
>>> marxes[::-1]
['Harpo', 'Chico', 'Groucho']
```

Жоден із цих фрагментів не змінив сам список `marxes`, тому що ми не присвоїли їх `marxes`. Щоб змінити наявний список, скористайтеся list.reverse ():

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.reverse()
>>> marxes
['Harpo', 'Chico', 'Groucho']
```

Додавання елемента в кінець списку за допомогою методу append().

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
```

```
>>> marxes.append('Zeppo')
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo']
```

Додавання елемента за зміщенням за допомогою insert().

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.insert(2, 'Gummo')
>>> marxes
['Groucho', 'Chico', 'Gummo', 'Harpo']
```

Об'єднання списків за допомогою extend() або +

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes.extend(others)
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

Крім того, ви можете використовувати + або + =:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> others = ['Gummo', 'Karl']
>>> marxes += others
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Zeppo', 'Gummo', 'Karl']
```

Зміна елемента за допомогою [offset]

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[2] = 'Wanda'
>>> marxes
['Groucho', 'Chico', 'Wanda']
```

Змінювання елементів за допомогою зрізів

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = [8, 9]
>>> numbers
[1, 8, 9, 4]
```

Права частина навіть не повинна мати таку саму кількість елементів, як зріз ліворуч:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = [7, 8, 9]
>>> numbers
[1, 7, 8, 9, 4]
```

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = []
>>> numbers
[1, 4]
```

Насправді, справа не обов’язково має бути лише список, а може бути будь-який ітерабельний об’єкт Python:

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = (98, 99, 100)
>>> numbers
[1, 98, 99, 100, 4]
```

```
>>> numbers = [1, 2, 3, 4]
>>> numbers[1:3] = 'wat?'
>>> numbers
```

```
[1, 'w', 'a', 't', '?', 4]
```

Видалення елемента через Offset за допомогою del

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Gummo', 'Karl']
>>> marxes[-1]
'Karl'
>>> del marxes[-1]
>>> marxes
['Groucho', 'Chico', 'Harpo', 'Gummo']
```

Видалення елемента за значенням за допомогою remove ()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.remove('Groucho')
>>> marxes
['Chico', 'Harpo']
```

Отримання елемента за допомогою Offset та видалення його за допомогою pop ()

Ви можете отримати елемент зі списку та видалити його зі списку одночасно за допомогою pop(). Якщо ви викликаєте pop() зі зміщенням, він поверне елемент із цим зміщенням; без аргументів, - він використовує -1. Отже, pop (0) повертає заголовок (початок) списку, а pop() або pop (-1) повертає хвіст (кінець), як показано тут:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> marxes.pop()
'Zeppo'
>>> marxes
['Groucho', 'Chico', 'Harpo']
>>> marxes.pop(1)
'Chico'
>>> marxes
['Groucho', 'Harpo']
```

Знайдення зміщення елемента за значенням за допомогою index ()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> marxes.index('Chico')
1
```

Перевірка значення з in

Pythonic спосіб перевірити наявність значення у списку є використання in:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> 'Groucho' in marxes
True
>>> 'Bob' in marxes
False
```

Порахування кількості значень за допомогою count()

Щоб підрахувати, скільки разів певне значення зустрічається у списку, скористайтеся count ():

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.count('Harpo')
1
>>> marxes.count('Bob')
```

```
0
>>> skit = ['cheeseburger', 'cheeseburger', 'cheeseburger']
>>> skit.count('cheeseburger')
3
```

Перетворення списку в рядок за допомогою join()

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> ', '.join(marxes)
'Groucho, Chico, Harpo'
```

Змінення порядку елементів за допомогою sort() або sorted()

Вам часто доведеться сортувати елементи у списку за їх значеннями. Python пропонує дві функції:

• Метод списку sort(), що сортує сам список на місці.

• Загальну функцію sort(), яка повертає відсортовану копію списку.

Якщо елементи у списку є числовими, вони за замовчуванням сортуються у порядку зростання числа. Якщо це рядки, їх сортують за алфавітом:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> sorted_marxes = sorted(marxes)
>>> sorted_marxes
['Chico', 'Groucho', 'Harpo']
```

sort_marxes - це новий список, і його створення не змінило вихідний список:

```
>>> marxes
['Groucho', 'Chico', 'Harpo']
```

Але виклик метода списку sort() у списку marxes дійсно змінює marxes:

```
>>> marxes.sort()
>>> marxes
['Chico', 'Groucho', 'Harpo']
```

Якщо всі елементи вашого списку мають один і той же тип (наприклад, рядки в `marxes`), sort () буде працювати коректно. Іноді можна навіть змішувати типи - наприклад, цілі числа і плаваючі - тому що вони автоматично перетворюються один на одного за допомогою виразів Python:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 4.0]
```

За замовчуванням порядок сортування за зростанням, але ви можете додати аргумент reverse = True, щоб встановити його за спаданням:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort(reverse=True)
>>> numbers
[4.0, 3, 2, 1]
```

Отримання довжини за допомогою len ()

len () повертає кількість елементів у списку:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> len(marxes)
3
```

Присвоєння за допомогою =

Коли ви призначаєте один список для кількох змінних, зміна списку в одному місці також змінює його в іншому, як показано тут:

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> a[0] = 'surprise'
>>> a
['surprise', 2, 3]
```

То що зараз у b? Це все ще [1, 2, 3] або ['surprise', 2, 3]? Подивимось:

```
>>> b
['surprise', 2, 3]
```

Копіювання за допомогою copy(), list() або Slice

Ви можете скопіювати значення списку в незалежний, свіжий список, використовуючи будь-який із цих методів:

- Метод списку copy ()
- Функцію перетворення list ()
- Зріз списку [:]

Наш оригінальний список буде a. Копії b, c, d:

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> c = list(a)
>>> d = a[:]
```

Копіювати все за допомогою deepcopy ()

Функція copy() добре працює, якщо значення списку незмінні. Як ви бачили раніше, змінні значення (наприклад, списки, кортежі або словники) - це посилання. Зміна оригіналу або копії відображатиметься в обох. Давайте скористаємося попереднім прикладом, але зробимо останній елемент у списку списком [8, 9] замість цілого числа 3:

```
>>> a = [1, 2, [8, 9]]
>>> b = a.copy()
>>> c = list(a)
>>> d = a[:]
>>> a
[1, 2, [8, 9]]
>>> b
[1, 2, [8, 9]]
>>> c
[1, 2, [8, 9]]
>>> d
[1, 2, [8, 9]]
```

Все йде нормально. Тепер змініть елемент у цьому підсписку a:

```
>>> a[2][1] = 10
>>> a
[1, 2, [8, 10]]
>>> b
[1, 2, [8, 10]]
```

```
>>> c
[1, 2, [8, 10]]
>>> d
[1, 2, [8, 10]]
```

Значення [2] тепер є списком, і його елементи можна змінювати. Усі методи копіювання списку, які ми використовували, були неглибокими (поверхневими, shallow) .

Щоб виправити це, нам потрібно використовувати функцію deepcopy ():

```
>>> import copy
>>> a = [1, 2, [8, 9]]
>>> b = copy.deepcopy(a)
>>> a
[1, 2, [8, 9]]
>>> b
[1, 2, [8, 9]]
>>> a[2][1] = 10
>>> a
[1, 2, [8, 10]]
>>> b
[1, 2, [8, 9]]
```

deepcopy () може обробляти глибоко вкладені списки, словники та інші об’єкти.

Ітерація за допомогою for – in

```
>>> cheeses = ['brie', 'gjetost', 'havarti']
>>> for cheese in cheeses:
... if cheese.startswith('g'):
... print("I won't eat anything that starts with 'g'")
... break
... else:
... print(cheese)
...
brie
I won't eat anything that starts with 'g'
```

Створення списку за допомогою генератора списку (List comprehensions)

Python підтримує концепцію під назвою "list comprehensions" ("включення списку"), яка може допомогти вам створити списки дуже легко. Ось кілька прикладів:

```
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]
>>> [[x,x**2] for x in vec if x > 3] #x**2 in Python is x*x
[[4, 16], [6, 36]]
>>> [[0]*3 for i in range(3)]
[[0,0,0],[0,0,0],[0,0,0]]
```

Таку техніку створення списку також називають генератором списку. Розглянемо ще один приклад.

```
>>> number_list = [number for number in range(1,6)]
>>> number_list
```

```
[1, 2, 3, 4, 5]
```

У першому рядку перша змінна number потрібна для створення значень для списку: тобто для розміщення результату циклу у number_list. Друге number є частиною циклу for. Щоб показати, що перше number є виразом, спробуйте такий варіант:

```
>>> number_list = [number-1 for number in range(1,6)]
>>> number_list
[0, 1, 2, 3, 4]
```

Генератор списку може включати умовний вираз, який виглядає приблизно так:

```
[expression for item in iterable if condition]
```

Давайте зробимо новий генератор, який будує список лише непарних чисел між 1 і 5 (пам'ятайте, що число % 2 є істинним для непарних чисел та хибним для парних чисел):

```
>>> a_list = [number for number in range(1,6) if number % 2
== 1]
>>> a_list
[1, 3, 5]
```

Список списків

Для роботи з багатомірними масивами вам знадобляться списки списків.

```
>>> grid = [[1,2,3], [4,5,6], [7,8,9]]
>>> grid
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> grid[0][1]
2
>>> grid[2]
[7,8,9] # row 2 is itself a list within the 2D grid list
```

Функція enumerate()

Інтерпретатор Python має ряд вбудованих функцій і типів, які завжди доступні. Вони перераховані тут в алфавітному порядку (https://docs.python.org/3/library/functions.html).

		Built-in Functions		
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()	

```
enumerate(iterable, start=0)
```

Повертає об'єкт перерахування. iterable має бути послідовністю, ітератором або іншим об’єктом, що підтримує ітерацію. Метод __next __ () ітератора, який повертає enumerate (), повертає кортеж, що містить count (від початку, який за замовчуванням до 0) та значення, отримані в результаті ітерації.

Приклад:

```
>>>seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>>list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>>list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

Функція range()

class range(stop)

class range(start, stop[, step])

Замість того, щоб бути функцією, range насправді є незмінним типом послідовності. Дивись https://docs.python.org/3/library/stdtypes.html#typesseq-range .

Приклади:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
```

Функція any()

Синтаксис: `any(iterable)`, де `iterable` - ітерабельний об'єкт (list, tuple, dictionary).

Функція any() повертає True, якщо будь-який елемент в ітерабельному об’єкті є істинним, інакше повертає False. Якщо ітерабельний об'єкт порожній, функція any() поверне False.

Приклад:

```
>>> mytuple = (0, 1, False)
>>> x = any(mytuple)
>>> print(x)
True
```

Функція map()

Синтаксис:

`map(function, iterable, [iterable 2, iterable 3, ...])`,

де `function` - функція, `iterable` - ітерабельний об'єкт (list, tuple, dictionary).

Повертає ітератор, який застосовує функцію до кожного елемента iterable, повертаючи результат. Якщо передаються додаткові аргументи iterable, функція повинна приймати стільки ж аргументів і застосовуватись до елементів з усіх iterable паралельно. Із кількома iterable ітератор зупиняється, коли вичерпано найкоротший iterable.

Приклад 1:

```
>>> def square(number):
     return number ** 2

>>> numbers = [1, 2, 3, 4, 5]
>>> squared = map(square, numbers)
>>> list(squared)
[1, 4, 9, 16, 25]
>>>
```

Приклад 2:

```
>>> t1 = (1, 2, 3)
>>> t2 = (5.0, 6.0, 7.0)
>>> list(map(lambda x, y : x/y, t1, t2))
[0.2, 0.3333333333333333, 0.42857142857142855]
>>>
```

1.4 Питання для самоконтролю

1. Які базові операції роботи з рядками є у мові Python?
2. Як можна одержувати зрізи рядків? Якими бувають зрізи? Наведіть приклади.
3. Які методи роботи з рядками є у мові Python?

4. Що таке словники? Як з ними працювати? Для чого вони потрібні?
5. Які методи роботи зі словниками є у мові Python?

ЛІТЕРАТУРА

1. Яковенко А.В. Основи програмування. Python. Частина 1: підручник для студ. спеціальності 122 "Комп'ютерні науки". – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.
2. Програмування на мові Python. Методичні вказівки до виконання лабораторних робіт студентами денної та заочної форми навчання спеціальностей 123 "Комп'ютерна інженерія", 125 "Кібербезпека" / Укл.: Є.В. Мелешко – Кропивницький: ЦНТУ, 2017. – 58 с.
3. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
4. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. – Чернігів: ФОП Баликіна С.М., 2020. 180 с.
5. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування. Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.
6. Лутц М. Изучаем Python, том 1, 5-е изд. — СПб.: ООО "Диалектика", 2019. — 832 с.
7. Лутц М. Изучаем Python, том 2, 5-е изд. — СПб.: ООО "Диалектика", 2020. — 720 с.
8. Joakim Sundnes Introduction to Scientific Programming with Python. – Springer, 2020. 148 p.
9. Bill Lubanovic Introducing Python. – O'Reilly Media, 2020. 597