

ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ Python

Урок 1

Вступ у web- програмування на Python

Зміст

1. Основи програмування	6
1.1. Як працює комп'ютерна програма?.....	6
1.2. Природні мови та мови програмування.....	7
1.3. З чого складається мова?	9
1.4. Компіляція та інтерпретація	10
1.5. Що ж насправді робить інтерпретатор?	12
1.6. Компіляція та інтерпретація — переваги та недоліки	14
2. Python — інструмент, а не рептилія	16
2.1. Що таке Python?	16
2.2. Хто створив Python?	16
2.3. Програмний проєкт-хобі.....	18
2.4. Цілі Python	18
2.5. Чому Python особливий?	19
2.6. Конкуренти Python?.....	21
2.7. Де ми можемо побачити Python у дії?	22
2.8. Чому не Python?	23

2.9. Python не один.....	23
2.10. Python vs. CPython.....	26
2.11. Cython.....	27
2.12. Jython	27
2.13. PyPy та RPython	28
3. Почніть свою подорож з Python.....	30
3.1. Як встановити та використовувати Python	30
3.2. Як завантажити та встановити Python	31
3.3. Починаємо працювати з Python	32
3.4. Як написати та запустити вашу першу програму.....	34
3.5. Як написати та запустити вашу першу програму.....	35
3.6. Як зіпсувати та виправити свій код.....	37
4. Типи даних, змінні, основні операції вводу- виводу, основні оператори.....	41
4.1. Пишемо першу програму Hello, World!.....	41
4.2. Функція print()	42
4.3. Функція print() — інструкція	47
4.4. Функція print() — екранування символів та перехід на новий рядок	49
4.5. Функція print() — використання декількох аргументів.....	50
4.6. Функція print() — позиційний спосіб передачі аргументів	51
4.7. Функція print() — ключові аргументи.....	52
4.8. Ключові висновки	55
5. Літерали Python	57
5.1. Літерали — дані в собі.....	57

5.2. Цілі числа (Integers)	58
5.3. Цілі числа: вісімкові та шістнадцяткові числа ...	60
5.4. Числа з рухомою крапкою (Floating-point numbers)	61
5.5. Кодування чисел з рухомою крапкою	63
5.6. Рядки.....	64
5.7. Кодування рядків	65
5.8. Булеві значення (логічні типи даних).....	65
5.9. Ключові висновки	66
6. Оператори — інструменти керування даними.....	68
6.1. Python як калькулятор	68
6.2. Основні оператори.....	68
6.3. Арифметичні оператори: піднесення до степеня	69
6.4. Арифметичні оператори: множення.....	70
6.5. Арифметичні оператори: ділення	70
6.6. Арифметичні оператори: ділення цілих чисел...	71
6.7. Оператори: остача (ділення за модулем, з остачею)	73
6.8. Оператори: як не ділити	74
6.9. Оператори: додавання.....	74
6.10. Оператор віднімання, унарні та бінарні оператори	74
6.11. Оператори та їх пріоритети	75
6.12. Оператори та зв'язування	76
6.13. Оператори та зв'язування: піднесення до степеня	77
6.14. Список пріоритетів.....	77
6.15. Оператори та дужки	78

6.16. Ключові висновки	78
7. Змінні — поля у формі даних.....	80
7.1. Що таке змінні?	80
7.2. Правильні та неправильні імена змінних	81
7.3. Ключові слова.....	82
7.4. Створення змінних	83
7.5. Використання змінних.....	85
7.6. Присвоєння нового значення вже існуючої змінної.....	86
7.7. Вирішення простих математичних завдань.....	87
7.8. Скорочені форми запису	88
7.9. Ключові висновки	89
8. Коментар до коментарів	91
8.1. Коментарі в коді: навіщо, як і коли	91
8.2. Ключові висновки	93
9. Як спілкуватися з комп'ютером.....	95
9.1. Функція введення input()	95
9.2. Функція input() з аргументом	96
9.3. Результат функції input()	97
9.4. Функція input() — заборонені операції	97
9.5. Перетворення типів.....	98
9.6. Більше про input() та перетворення типів	99
9.7. Рядкові оператори — введення	101
9.8. Конкатенація (concatenation).....	101
9.9. Повторення рядка (replication)	102
9.10. Перетворення типів: str()	103
9.11. Повертаємось до прямокутного трикутника	104
9.12. Ключові висновки	104

1. Основи програмування

1.1. Як працює комп'ютерна програма?

Завдання цього курсу — розповісти вам про те, що таке мова програмування Python і для чого вона використовується. Давайте почнемо з основ.

Ми можемо користуватися комп'ютером завдяки програмам. Без програм комп'ютер, навіть найпотужніший, — просто якийсь об'єкт. Так само й піаніно без піаніста — не більше, ніж дерев'яна коробка.

Комп'ютери можуть виконувати дуже складні завдання, але ця здатність не є природною. Природа комп'ютера зовсім інша.

Він може виконувати тільки надзвичайно прості операції, наприклад, комп'ютер не може самостійно оцінити значення складної математичної функції, хоча це не виходить за межі можливого у найближчому майбутньому.



Рисунок 1

Сучасні комп'ютери можуть лише оцінювати результати простих арифметичних операцій, таких, як додавання або ділення, але вони це роблять дуже швидко і можуть повторювати ці дії будь-яку кількість разів.

Уявіть, що ви хочете дізнатися про вашу середню швидкість, з якою ви йшли протягом довгої подорожі. Ви знаєте відстань та час, але вам потрібна швидкість. Звичайно, комп'ютер зможе вирахувати це, але комп'ютер не розуміє, що таке відстань, швидкість або час. Отже, нам потрібно проінструктувати комп'ютер, щоб він:

- прийняв число, що становить відстань;
- прийняв число, що становить час у дорозі;
- розділив перше значення на друге та зберіг результат у пам'яті;
- вивів результат (який представляє середню швидкість) у форматі, який людина може прочитати.

Ці чотири прості дії утворюють програму. Звичайно, ці приклади не формалізовані і вони дуже далекі від того, що комп'ютер може зрозуміти, але їх достатньо, щоб перекласти це все на зрозумілу комп'ютеру мову.

Мова тут ключове слово.

1.2. Природні мови та мови програмування

Мова — це засіб (та інструмент) для вираження та запису думок. У світі є безліч мов. Деякі з них не вимагають ні мови, ні написання, наприклад мову тіла. Ви можете висловити свої найглибші почуття дуже точно, не кажучи жодного слова. Іншою мовою, якою ви користуєтеся кожен день, є ваша рідна мова, якою ви мислите та озвучуєте свої

бажання. Комп'ютери теж мають свою мову під назвою «**машинна мова**», але вона дуже примітивна.

Комп'ютер, навіть найскладніший технічно, позбавлений слідів інтелекту. Можна сказати, що він схожий на добре навченого собаку — він реагує лише на заздалегідь визначений набір відомих йому команд.

Команди, які він розпізнає, дуже прості. Можна уявити, що комп'ютер реагує на такі команди, як «взяти це число, розділити на інше і зберегти результат».

Повний набір відомих команд називається **списком інструкцій** (*instruction list*) і іноді скорочується до **ІЛ**. Комп'ютери розрізняються в залежності від розміру їх ІЛ, також інструкції можуть бути різними в різних моделях.

ПРИМІТКА: *машинні мови розробляються людиною.*

Жоден комп'ютер нині не здатний створювати нову мову. Однак це може скоро змінитись. Люди теж використовують безліч різних мов, але ці мови створили самі себе. Більше того, вони все ще розвиваються.

Нові слова утворюються щодня, а старі слова зникають. Ці мови називаються **природними**.

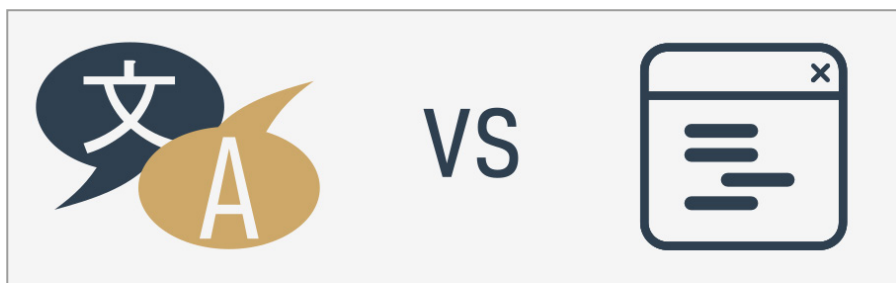


Рисунок 2

1.3. З чого складається мова?

Можна сміливо сказати, що кожна мова (машинна або природна, неважливо) складається з наступних елементів:

- **Алфавіт** — набір символів, що використовуються для побудови слів певної мови (наприклад, латинський алфавіт для англійської, кириличний алфавіт для української, кандзі для японської тощо).
- **Лексика** — Набір слів (словник), які мова пропонує своїм користувачам (наприклад, слово «*computer*» походить зі словника англійської мови, тоді як «*smotru*» — ні; слово «*chat*» присутнє як в англійській, так і у французькій словниках, але мають різні значення).
- **Синтаксис** — набір правил (формальних або неформальних, написаних або сприйманих інтуїтивно), які допомагають визначити, чи є той чи інший набір словосполучень правильним реченням (наприклад, «*I am a Python*» — синтаксично правильна фраза, а «*I a Python am*» — ні).
- **Семантика** — набір правил, визначальних, чи має сенс певна фраза (наприклад, «я з'їв пончик» має сенс, а «пончик з'їв мене» — ні).

Насправді, **список інструкцій** — це абетка машинної мови. Це найпростіший і найбільш базовий набір символів, який ми можемо використовувати, щоб віддавати команди комп'ютеру. Це рідна мова комп'ютера.

На жаль, ця мова дуже далека від рідної мови людей. Нам усім (і комп'ютерам, і людям) потрібно щось ще, спільну мову для комп'ютерів та людей або міст між двома різними світами.

Нам потрібна мова, якою люди можуть писати свої програми, і мова, яку комп'ютери можуть використовувати для виконання програм, мова, яка набагато складніша, ніж машинна мова, і все ж таки набагато простіша за природню мову.

Такі мови часто називають високорівневими мовами програмування. Вони чимось схожі на природні в тому, що вони використовують символи, слова та умовні позначення, які людина здатна зрозуміти. Ці мови дозволяють людям передавати комп'ютерам команди, які набагато складніші, ніж список інструкцій.

Програма, написана високорівневою мовою програмування, називається «**початковий код**» (на відміну від машинного коду, що виконується комп'ютерами). А файл, що містить початковий код, називається «початковий файл».

1.4. Компіляція та інтерпретація

Комп'ютерне програмування — це процес складання елементів вибраної мови програмування у порядку, який викликатиме бажаний ефект. Ефект може відрізнятися у кожному конкретному випадку. Все залежить від уяви, знань та досвіду програміста.

Звичайно, такий твір має бути правильним за багатьма пунктами:

- **алфавіт** — програма має бути написана відомим набором літер, наприклад, латиницею, кирилицею тощо;
- **лексика** — кожна мова програмування має свій словник, і вам потрібно освоїти його; на щастя, він набагато простий і менший за словник будь-якої природної мови;

- **синтаксис** — кожна мова має свої правила, і їх потрібно дотримуватися;
- **семантика** — програма має мати сенс.

На жаль, програміст може помилитися в кожному з чотирьох зазначених вище пунктів. Помилка в будь-якому з них може зробити програму абсолютно непотрібною.

Уявімо, що ви успішно написали програму. Як ми переконаємо комп'ютер її виконати? Вам потрібно перекласти вашу програму машинною мовою. На щастя, сам комп'ютер може виконати цей переклад, отже, це буде швидко та ефективно.

Є два різні способи перетворення програми з високорівневої мови програмування на машинну мову:

- **Компіляція** — початкова програма перекладається один раз (проте, цю дію необхідно повторювати щоразу, коли ви змінюєте початковий код), ви отримуєте файл (наприклад, файл *.exe*, якщо код призначений для запуску в MS Windows), який містить машинний код. Тепер ви можете розповсюджувати файл по всьому світу. Програма, яка виконує цей переклад, називається компілятором або перекладачем.
- **Інтерпретація** — ви (або будь-яка людина, яка використовує цей код) можете перекладати початкову програму щоразу, коли вона має бути запущена. Програма, що виконує таке перетворення, називається інтерпретатором, тому що вона інтерпретує код щоразу, коли він має бути виконаний; це також означає, що ви не можете просто взяти і розповсюдити початковий код, тому що кінцевому користувачеві також буде потрібний інтерпретатор для його виконання.

За деякими основними причинами, та чи інша високорівнева мова програмування має потрапити до однієї з цих двох категорій.

Існує дуже мало мов, які можна і компілювати, і інтерпретувати. Творці мов програмування, зазвичай, одразу вирішують, чи буде нова мова компілюватися, чи інтерпретуватися.

1.5. Що ж насправді робить інтерпретатор?

Давайте ще раз уявимо, що ви написали програму. Тепер вона існує як **комп'ютерний файл**: комп'ютерна програма насправді є фрагментом тексту, тому початковий код зазвичай зберігається в **текстових файлах**. Примітка: це має бути **чистий текст**, без якихось прикрас, таких, як: різні шрифти, кольори, вбудовані зображення, аудіо, відео і т. д. Тепер ви маєте викликати інтерпретатор, щоб він прочитав ваш початковий файл.

Інтерпретатор читає початковий код так, як це заведено у західній культурі: зверху вниз і зліва направо. Є деякі винятки — вони будуть розглянуті далі у цьому курсі.

Насамперед, інтерпретатор перевіряє правильність усіх наступних рядків (через призму тих чотирьох пунктів, які були розглянуті вище).

Якщо компілятор знаходить помилку, він негайно завершує свою роботу. Єдиним результатом у цьому випадку є **повідомлення про помилку**. Інтерпретатор повідомить вам, де знаходиться помилка і що її викликало. Однак ці повідомлення можуть заплутати, оскільки інтерпретатор не в змозі простежити ваші точні наміри і може повідомити про помилку на певній відстані від місця, яке, власне,

і містить цю помилку. Наприклад, якщо ви спробуєте використати об'єкт з невідомим ім'ям, це призведе до помилки, але інтерпретатор виявить помилку там, де він спробує використати об'єкт, а не там, де було введено ім'я нового об'єкта. Інакше кажучи, фактична причина зазвичай знаходиться трохи вище в коді, наприклад, у тому місці, де ви повинні були повідомити інтерпретатор, що ви збираєтеся використати об'єкт із цим ім'ям.

Якщо рядок виглядає правильно, інтерпретатор намагається виконати його.

ПРИМІТКА: *кожен рядок зазвичай виконується окремо, тому така послідовність, як «читання-перевірка-виконання» може повторюватися багато разів — більше, ніж фактична кількість рядків у вихідному файлі, оскільки деякі частини коду можуть виконуватися більше одного разу).*

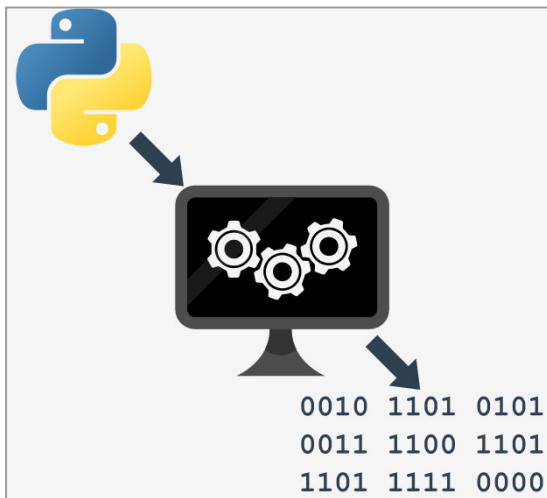


Рисунок 3

Також можливо, що значна частина коду може бути успішно виконана до того, як інтерпретатор виявить помилку. Це нормальна поведінка у цій моделі виконання.

У вас може виникнути логічне питання — а що ж краще: модель компіляції чи модель інтерпретації? Тут немає явної відповіді. Якби відповідь була, одна з цих моделей давно перестала б існувати. Кожна за них має свої переваги та недоліки.

1.6. Компіляція та інтерпретація — переваги та недоліки

Таблиця 1

	КОМПІЛЯЦІЯ	ІНТЕРПРЕТАЦІЯ
ПЕРЕВАГИ	Виконання перекладеного коду зазвичай відбувається швидше; тільки у програміста повинен бути компілятор — кінцевий користувач може використовувати код без нього; перекладений код зберігається за рахунок машинної мови, оскільки його дуже важко зрозуміти; ваші власні знахідки та прийоми програмування, ймовірно, залишаться вашим секретом.	Код можна запустити одразу, як тільки закінчите його — додаткових етапів перекладу немає; код зберігається з допомогою мови програмування, а не машинної мови — це означає, що його можна запускати на комп'ютерах, які використовують різні машинні мови; не потрібно компілювати код для кожної окремої архітектури.
НЕДОЛІКИ	Сама компіляція може бути дуже трудомістким процесом — ви не зможете запустити код одразу ж після будь-якої зміни; кількість компіляторів, що використовуються, має відповідати кількості апаратних платформ, на яких ви хочете, щоб ваш код працював.	Не думайте, що інтерпретація збільшить швидкість вашого коду — ваш код поділиться потужністю комп'ютера з інтерпретатором, отже на швидкість розраховувати не варто; і у вас, і у кінцевого користувача має бути інтерпретатор для запуску вашого коду.

Що це все означає?

- **Python є інтерпретованою мовою програмування.** Це означає, що Python успадковує усі описані вище переваги та недоліки. Звичайно, він додає й свої унікальні особливості до обох наборів.
- Якщо ви хочете програмувати на Python, вам знадобиться **інтерпретатор** Python. Ви не можете запустити код без нього. На щастя, Python **безкоштовний**. Це одна з його найважливіших переваг.

2. Python — інструмент, а не рептилія

Інтерпретовані мови програмування часто називають «мовами сценаріїв» (сценарними мовами, скриптовими мовами), а початкові програми, закодовані з їх використанням, називаються «сценарії» (скрипти).

2.1. Що таке Python?

Python — це широко використовувана, інтерпретована, об'єктно-орієнтована, високорівнева та багатоцільова мова програмування з динамічною семантикою.

Можливо, перше, що ви згадуєте, коли чуєте слово «пітон», це велика змія, проте назва мови програмування Python походить від старого комедійного серіалу BBC під назвою «Літаючий Цирк Монті Пайтона».

На піку свого успіху команда Monty Python розігрувала свої скетчі для своєї глядацької аудиторії по всьому світу, у тому числі на Голлівуд-Боулі, відомому концертному залі у вигляді амфітеатру.

Оскільки Монті Пайтон вважається однією з двох основних поживних речовин для програміста (друга — це піца), творець Python назвав мову на честь цього телешоу.

2.2. Хто створив Python?

Однією з дивовижних рис Python є той факт, що насправді це робота однієї людини. Зазвичай нові мови програмування розробляються і публікуються великими

компаніями, в яких працює багато професіоналів, і авторські права часто не дають змоги назвати когось із учасників проекту. Python — виняток.

Існує не так багато мов, авторів яких ми знаємо по-іменню. Python був створений **Гвідо ван Россумом**, який народився у 1956 році в Гарлемі, Нідерланди. Звичайно, Гвідо ван Россум не розробляв і не розвивав абсолютно всі складові Python.

Швидкість, з якою Python поширився по всьому світу, є результатом безперервної роботи тисяч (часто анонімних) програмістів, тестувальників, користувачів (багато з яких не є ІТ-фахівцями) та ентузіастів, але слід сказати, що найперша ідея (зерно, з якого виріс Python) прийшла лише одній людині — Гвідо.



Рисунок 4

2.3. Програмний проєкт-хобі

Python був створений у трохи дивних умовах. За словами Гвідо Ван Россума:

«У грудні 1989-го року я шукав проєкт, який став би хобі на різдвяні канікули. Офіс з усім обладнанням було закрито, але вдома я мав комп'ютер, а інших планів, чим зайнятися, не було. Я вирішив написати інтерпретатор для нової скриптової мови, про яку думав останнім часом: про нащадка ABC, який був би привабливим для Unix/C-хакерів. Я вибрав Python в якості робочої назви перебуваючи в трохи нешанобливому настрої (і будучи великим фанатом літаючого цирку Монті Пайтона)».

2.4. Цілі Python

У 1999 році Гвідо ван Россум визначив цілі Python:

- проста та інтуїтивно зрозуміла мова, така ж потужна, як і основні конкуренти;
- відкритий початковий код, щоб кожен міг зробити свій внесок у його розвиток;
- код повинен бути таким же зрозумілим, як і англійська мова;
- підходящий для повсякденних завдань, дозволяє вкластися в короткі терміни розробки.

Через 20 років стало зрозуміло, що всі цілі були здійснені. Деякі джерела говорять, що Python є найпопулярнішою мовою програмування у світі, в той час як інші стверджують, що вона займає третє або п'яте місце за популярністю.



Рисунок 5

Так чи інакше, Python, як і раніше, займає високе місце в першій десятці рейтингу мов програмування PYPL (*Popularity of Programming Language*) та в індексі TIOBE (*TIOBE programming community index*).

Python не молода мова. Він зрілий і заслуговує на довіру. Це не мова-одноденка. Це яскрава зірка на небосхилі програмування, і час, витрачений на вивчення Python є дуже гарною інвестицією.

2.5. Чому Python особливий?

Як вийшло, що програмісти, молоді та старі, досвідчені та початківці, хочуть використовувати цю мову? Як так сталося, що великі компанії визнали Python і створили свої флагманські продукти, використовуючи його? На це є багато причин — ми вже перерахували

деякі з них, але перерахуємо їх знову, в більш структурованому порядку:

- Python легко вчити — потребує, зазвичай, менше часу для вивчення, ніж деякі інші мови; це означає, що можна розпочати програмувати швидше;
- йому легко вчити — навчальне навантаження менше, ніж потрібно для інших мов; це означає, що вчитель може приділяти більше уваги загальним (незалежним від мови) методам програмування, не гаючи часу на екзотичні прийоми, дивні винятки та незрозумілі правила;
- його легко використовувати для написання нового програмного забезпечення — з Python на код часто йде набагато менше часу;
- його легко зрозуміти — також часто легше зрозуміти чужий код, якщо він написаний на Python;
- його легко отримати, встановити та впровадити — Python безкоштовний, відкритий та мультиплатформний; не всі мови можуть цим похвалитися.

Звичайно, у Python є й свої недоліки:

- він не супер-швидкісний — Python не забезпечує виняткову продуктивність;
- в деяких випадках він може бути стійким до деяких простих методів тестування — це означає, що налагодження коду Python може бути складніше, ніж в інших мовах; на щастя, і робити помилки в Python складніше.

Слід також зазначити, що Python не є єдиним подібним рішенням, доступним на ринку інформаційних технологій.

У нього багато послідовників, але багато хто віддає перевагу іншим мовам і навіть не розглядає Python для своїх проєктів.



Рисунок 6

2.6. Конкуренти Python?

Python має два прямих конкуренти з порівнянними властивостями і проблемами. Це:

- **Perl** — мова сценаріїв, створена Ларрі Воллом;
- **Ruby** — мова сценаріїв, створена Юкіхіро Мацумото.

Перша мова більш традиційна та консервативна, ніж Python, і нагадує деякі старі добрі мови, засновані на класичній мові програмування C.

А друга є більш інноваційною, з більш свіжими ідеями, ніж Python. Сам Python стоїть десь між цими двома мовами.

Інтернет повний форумів з нескінченними дискусіями про перевагу однієї з цих трьох мов над іншими, на випадок, якщо ви захочете дізнатися більше про кожен з них.

2.7. Де ми можемо побачити Python у дії?

Ми бачимо його щодня і мало не на кожному кроці. Він широко використовується для реалізації складних Інтернет-сервісів, наприклад, пошукових систем, хмарних сховищ та інструментів, соціальних мереж тощо. Щоразу, коли ви використовуєте якийсь із цих сервісів, ви насправді дуже близькі до Python, хоча ви цього й не знаєте.



Рисунок 7

Багато інструментів розробки реалізовані на Python. Все більше й більше щоденно використовуваних додатків

пишуться на Python. Більшість вчених відмовилися від дорогих фірмових інструментів та перейшли на Python. Велика кількість тестувальників ІТ-проектів почали використовувати Python для виконання повторюваних процедур тестування. Список великий.

2.8. Чому не Python?

Незважаючи на зростаючу популярність Python, є ще ніші, де Python зовсім відсутній або рідко зустрічається:

- низькорівневе програмування (іноді його називають програмуванням «ближче до заліза»): якщо ви хочете реалізувати надзвичайно ефективний драйвер або графічний движок, ви не станете використовувати Python;
- додатки для мобільних пристроїв: поки що Python не завоював цю територію, але, швидше за все, колись завоює.

2.9. Python не один

Існує два основних типи Python, які називаються Python 2 та Python 3.

Python 2 є старішою версією оригінального Python. З того часу його розвиток навмисно зупинився, хоча це не означає, що його не оновлюють. Навпаки, оновлення виходять регулярно, але вони не вносять жодних істотних змін у мову. Вони швидше виправляють будь-які нещодавно виявлені помилки та дірки у безпеці. Шлях розробки Python 2 вже зайшов у глухий кут, але сам Python 2 все ще живий.

Python 3 є новішою (точніше, поточною) версією мови. Він проходить свій власний шлях розвитку, створюючи

свої стандарти та звички. Перша мова більш традиційна та консервативна, ніж Python, і нагадує деякі старі добрі мови, засновані на класичній мові програмування C.

Ці дві версії Python не сумісні одна з одною. Скрипти Python 2 не працюватимуть в середовищі Python 3, і навпаки. Тому, якщо ви хочете, щоб старий код Python 2 виконувався інтерпретатором Python 3, єдине можливе рішення — переписати його, звичайно ж, не з нуля. Оскільки великі частини коду можуть бути незадіяні, але вам потрібно переглянути весь код, щоб знайти всі можливі несумісності. На жаль, цей процес може бути повністю автоматизований.

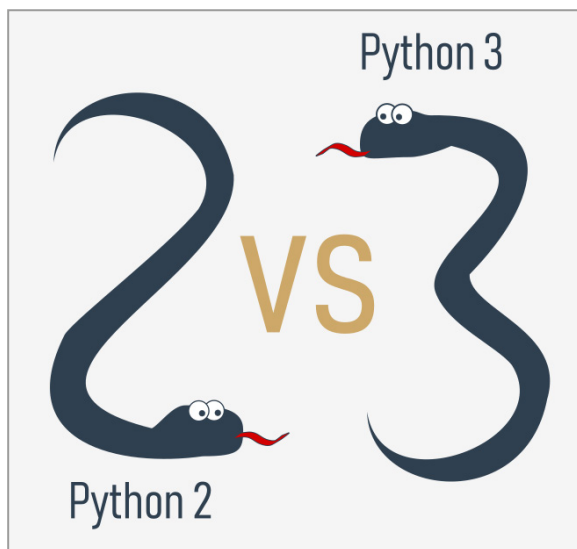


Рисунок 8

Перенесення старого додатка Python 2 на нову платформу — це надто складно, займає надто багато часу, надто дорого та надто ризиковано. Цілком імовірно, що

переписаний код міститиме нові помилки. Простіше і розумніше дати цим системам спокій і покращити існуючий інтерпретатор замість того, щоб намагатися працювати всередині вже працюючого початкового коду.

Python 3 — це не просто найкраща версія Python 2, це зовсім інша мова, хоч вона й дуже схожа на свого попередника. На перший погляд, мови здаються однаковими, але, якщо придивитися, можна помітити безліч відмінностей.

Якщо ви модифікуєте старе рішення, написане на Python, то ймовірно, що воно було написано саме на Python 2. Ось чому Python 2 все ще використовується. Зараз існує дуже багато додатків на Python 2, аби повністю від них відмовитися.

ПРИМІТКА. *Якщо ви бажаєте написати новий проєкт на Python, використовуйте Python 3. Саме цю версію Python ми будемо використовувати у цьому курсі.*

Важливо пам'ятати, що між наступними релізами Python 3 можуть бути маленькі або великі відмінності (наприклад, у Python 3.6 введені впорядковані словникові ключі за замовчуванням, під реалізацією CPython) — гарна новина полягає в тому, що нові версії Python 3 зворотно сумісні з попередніми версіями Python 3. Ми завжди намагатимемося підкреслити ці відмінності в курсі, якщо це буде важливо та корисно.

Усі приклади коду, які ви знайдете в цьому курсі, протестовані на Python 3.4, Python 3.6 та Python 3.7.

2.10. Python vs. CPython

Крім Python 2 та Python 3 існує більше однієї версії кожного з них. Насамперед, є мови Python, які підтримують люди, що належать до PSF ([Python Software Foundation](https://www.python.org/psf/)), — спільнота, яка прагне розвивати, покращувати, розширювати та популяризувати Python та його середовище. Президентом PSF є сам Гвідо ван Россум, тому ці мови Python називаються «**канонічними**». Вони також вважаються **референсними мовами Python**, оскільки будь-яка інша реалізація мови повинна відповідати всім стандартам, встановленим PSF.

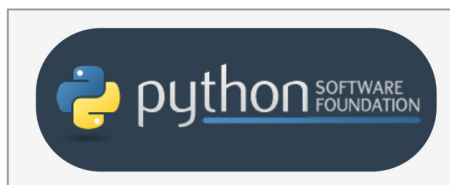


Рисунок 9

Гвідо ван Россум використовував мову програмування «С» для реалізації першої версії своєї мови, і це рішення залишається в силі. Усі мови Python, що надходять з PSF, написані мовою «С». Такий підхід обумовлений багатьма причинами і має багато наслідків. Один з них (ймовірно, найважливіший) полягає в тому, що завдяки цьому Python можна легко портувати та мігрувати на всі платформи з можливістю компіляції та запуску програм мовою С (практично всі платформи мають таку функцію, що відкриває безліч можливостей розширення для Python).

Ось чому реалізацію PSF часто називають **CPython**. Це найвпливовіший Python серед усіх мов Python у світі.

2.11. Cython

Інший член сім'ї Python — Cython.

Cython — це одне з можливих рішень для проблемної риси Python — недостатньої ефективності. Великі та складні математичні обчислення можна легко закодувати в Python (набагато простіше, ніж у «С» або будь-якій іншій традиційній мові), але виконання результуючого коду може бути надзвичайно трудомістким.

Як примирити ці дві суперечності? Одне з рішень — напишіть свої математичні ідеї з використанням Python, і коли ви будете повністю впевненими, що ваш код вірний і дає правильні результати, переведіть його в «С». Звичайно, «С» працюватиме набагато швидше, ніж «чистий» Python.

Саме це й робить Cython — автоматично переводить код Python («чистий» і зрозумілий, але не надто швидкий) у код «С» (складний і багатослівний, але гнучкий).



Рисунок 10

2.12. Jython

Ще одна версія Python називається **Jython**. «J» розшифровується як «Java». Уявіть собі Python, написаний на Java замість С. Це корисно, наприклад, якщо ви розробляєте великі та складні системи, повністю написані на Java, і хочете додати до них певну гнучкість Python.

Традиційний CPython може важко інтегруватися в таке середовище, оскільки С та Java живуть у повністю різних світах і не поділяють багато ідей.

Jython може ефективніше взаємодіяти з існуючою інфраструктурою Java. Ось чому деякі проекти вважають його корисним та потрібним.

ПРИМІТКА: *поточна реалізація Jython відповідає стандартам Python 2. Поки що не існує Jython, який відповідає Python 3.*



Рисунок 11

2.13. PyPy та RPython

Подивіться на логотип нижче. Це ребус. Чи можете ви його розгадати?



Рисунок 12

Це логотип PyPy — Python всередині Python. Іншими словами, він являє собою середовище Python, написане Python-подібною мовою і назване **RPython** (*Restricted*

Python, обмежений Python). Насправді це різновид Python. Початковий код PyPy не інтерпретується, натомість він перекладається мовою програмування C, а потім виконується окремо.

Це корисно в тому випадку, якщо вам потрібно протестувати якусь нову функцію, яка може бути (але не обов'язково) впроваджена у звичайну реалізацію Python, тоді її простіше перевірити з допомогою PyPy, ніж з CPython. Таким чином, PyPy, скоріш, — інструмент для тих, хто розробляють Python, ніж для інших користувачів. Звичайно, це не применшує заслуг PyPy і не робить його менш серйозним у порівнянні з CPython. До того ж, PyPy сумісний з мовою Python 3.

У світі є ще багато різних мов Python. Ви знайдете їх, якщо пошукайте, але у цьому курсі використовується CPython.

3. Почніть свою подорож з Python

3.1. Як встановити та використовувати Python

Є декілька способів отримати власну копію Python 3 залежно від операційної системи, що використовується.

У користувачів Linux, швидше за все, вже встановлено Python, оскільки інфраструктура Python інтенсивно використовується багатьма компонентами ОС Linux.

Наприклад, деякі дистриб'ютори можуть поєднувати свої інструменти з системою, і багато цих інструментів, такі як менеджери пакетів, часто написані на Python. Деякі елементи графічних середовищ, доступних у світі Linux, також можуть використовувати Python.

Якщо ви користувач Linux, відкрийте термінал/консоль і введіть:

```
python3
```

у командному рядку натисніть **Enter** та зачекайте.

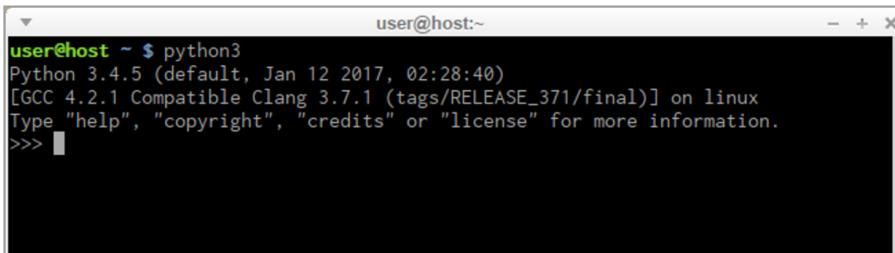
Якщо ви бачите щось на зразок цього:

```
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_371/final)]
on linux Type "help", "copyright", "credits" or
"license" for more information. >>>
```

тоді вам не потрібно більше нічого робити. Якщо Python 3 відсутній, зверніться до документації Linux, щоб дізнатися,

як використовувати менеджер пакетів для завантаження та встановлення нового пакета. Той, який вам потрібний, називається «python3» або його назва починається з цього слова.

Ті, хто не використовує Linux, можуть завантажити копію за цим посиланням: <https://www.python.org/downloads/> (рис. 13).



```
user@host:~$ python3
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_371/final)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Рисунок 13

3.2. Як завантажити та встановити Python

Оскільки браузер повідомляє сайту, на який ви зайшли, про ОС, яка використовується, єдине, що вам потрібно зробити, — клацнути на потрібну версією Python. У нашому випадку, це **Python 3**. Сайт завжди пропонує вам останню версію.

Якщо ви користувач Windows, запустіть завантажений файл `.exe` та виконайте всі кроки. Поки залиште стандартні налаштування, які пропонує інсталятор, за одним винятком — подивіться на пункт **Add Python 3.x to PATH** і виберіть його. Це багато чого спростить.

Якщо ви використовуєте MacOS, версія Python 2, можливо, вже була попередньо встановлена на ваш комп'ютер, але оскільки ми будемо працювати з Python 3, вам все одно

потрібно завантажити і встановити відповідну версію файлу `.pkg` з сайту Python (рис. 14).

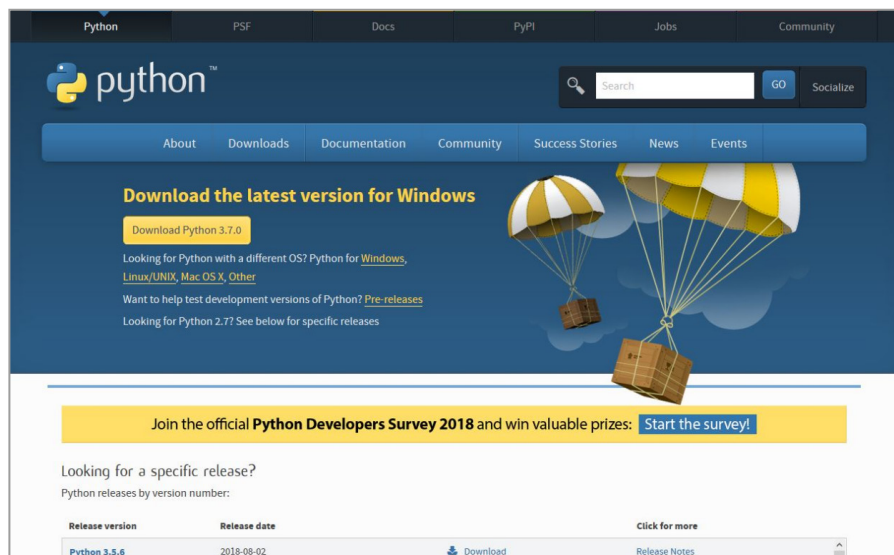


Рисунок 14

3.3. Починаємо працювати з Python

Тепер, коли у вас встановлено Python 3, настав час перевірити, як він працює, і вперше скористатися ним. Це буде дуже проста процедура, але її має бути достатньо, щоб переконати вас, що середовище Python є повноцінним та функціональним.

Існує безліч способів використання Python, особливо якщо ви маєте намір стати Python-розробником.

Щоб розпочати роботу, вам знадобляться наступні інструменти:

- **редактор**, який допоможе вам у написанні коду (у нього мають бути деякі спеціальні функції, недоступні

у простих інструментах); цей спеціальний редактор дасть вам більше, ніж стандартна програма вашої ОС;

- **консоль**, в якій ви можете запустити свій щойно написаний код і примусово зупинити його, коли він вийде з-під контролю;
- інструмент під назвою **«налагоджувач»**, який запускає покрокове виконання коду і дозволяє вам перевіряти його на кожному етапі виконання.

Крім безлічі корисних компонентів, стандартне встановлення Python 3 містить дуже простий, але надзвичайно корисний додаток під назвою IDLE.

IDLE — це аббревіатура від *Integrated Development and Learning Environment* (інтегроване середовище розробки та навчання).

Відкрийте меню вашої ОС, знайдіть IDLE десь під Python 3.x та запустіть її. Ось що у вас має з'явитися (рис. 15):

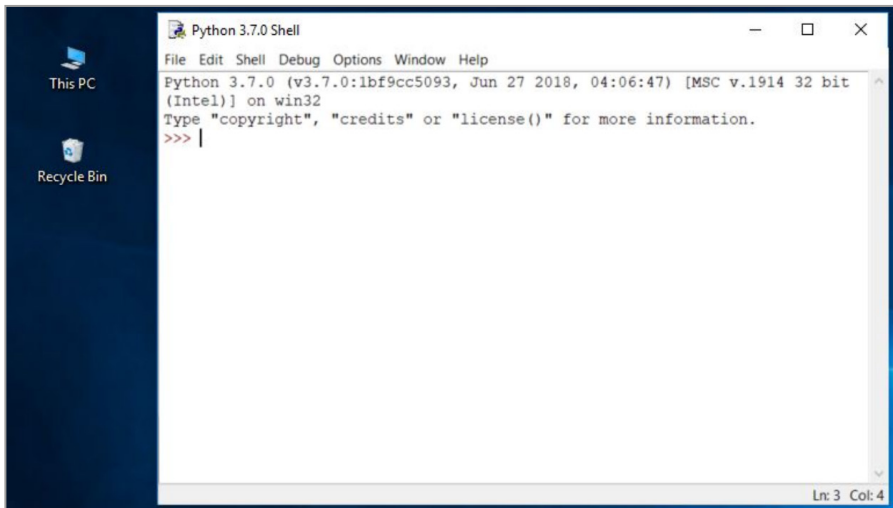


Рисунок 15

3.4. Як написати та запустити вашу першу програму

Тепер настав час написати та запустити вашу першу програму на Python 3. Поки що це буде дуже просто.

Перший етап — створити новий початковий файл та заповнити його кодом. Натисніть на **File** у меню IDLE і виберіть **New file** (рис. 16).

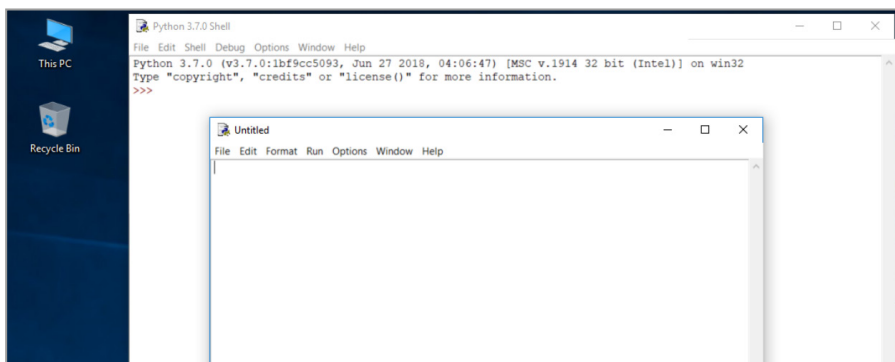


Рисунок 16

Як бачите, IDLE відкриває нове вікно. Ви можете скористатися ним, щоб писати та змінювати свій код.

Це **вікно редактора**. Його єдина мета — бути робочим місцем, в якому обробляється ваш початковий код. Не плутайте вікно редактора з вікном оболонки. Вони виконують різні функції.

Вікно редактора зараз без назви, але починати роботу рекомендовано з назви початкового файлу. Натисніть на **File** (у новому вікні), потім натисніть **Save as...**, виберіть папку для нового файлу (робочий стіл — непогане місце для перших спроб програмування) і виберіть ім'я нового файлу (рис. 17).

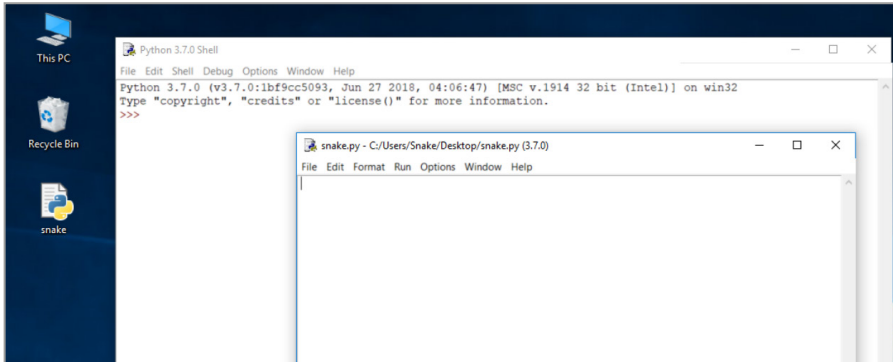


Рисунок 17

ПРИМІТКА: не встановлюйте розширення для назви файлу, який ви збираєтеся використовувати. Python вимагає розширення `.py` для своїх файлів, тому ви повинні покладатися на значення за замовчуванням у діалоговому вікні. Стандартне розширення `.py` дозволяє ОС правильно відкривати ці файли.

3.5. Як написати та запустити вашу першу програму

Тепер додайте лише один рядок у вікно редактора, яке ви недавно відкрили та назвали.

Рядок виглядає так:

```
print("Hissssssss...")
```

Можна скористатися буфером обміну, щоб скопіювати текст у файл.

Поки що ми не пояснюватимемо суть програми. Ви знайдете детальний розгляд щодо цього у наступному уроці.

Зверніть увагу на лапки. Це найпростіші лапки (нейтральні, прямі, подвійні і т. д.), які зазвичай використовуються у початкових файлах. Не намагайтеся використовувати друкарські лапки (закруглені, фігурні, книжкові і т. д.), які використовуються просунутими текстовими процесорами, оскільки Python їх не сприймає (рис. 18).

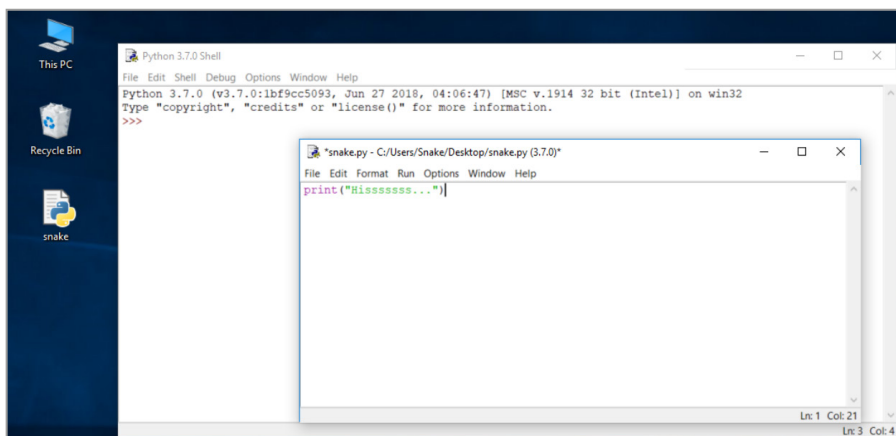


Рисунок 18

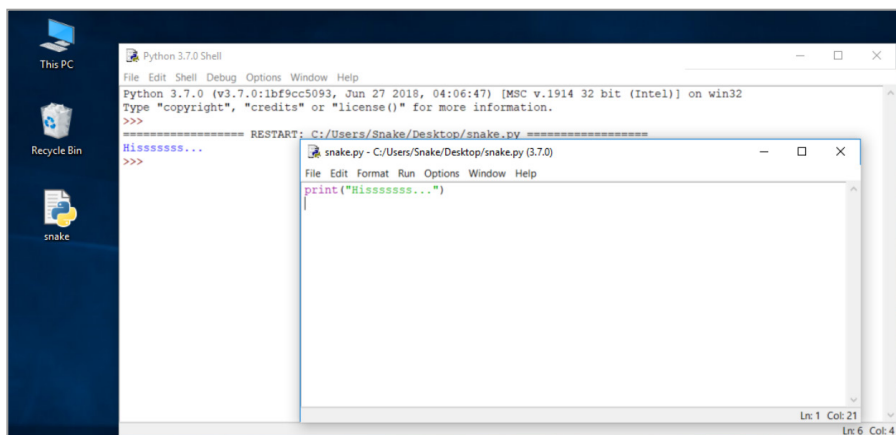


Рисунок 19

Якщо все йде добре і в коді немає помилок, у вікні консолі відобразяться ефекти, викликані запуском програми. У цьому випадку програма шипить. Спробуйте запустити її ще раз (рис. 19). І ще раз.

Тепер закрийте обидва вікна і перейдіть на робочий стіл.

3.6. Як зіпсувати та виправити свій код

Знову запустіть IDLE. Натисніть на **File, Open**, виберіть файл, який ви зберегли раніше, і дайте IDLE прочитати його.

Спробуйте запустити його знову, натиснувши **F5**, коли вікно редактора активне. Як бачите, IDLE може зберігати ваш код та витягувати його, коли він вам знадобиться знову.

IDLE має ще одну корисну функцію. Спочатку вида-літь закриваючу дужку. Потім додайте дужку знову. Ваш код має виглядати приблизно так (рис. 20):

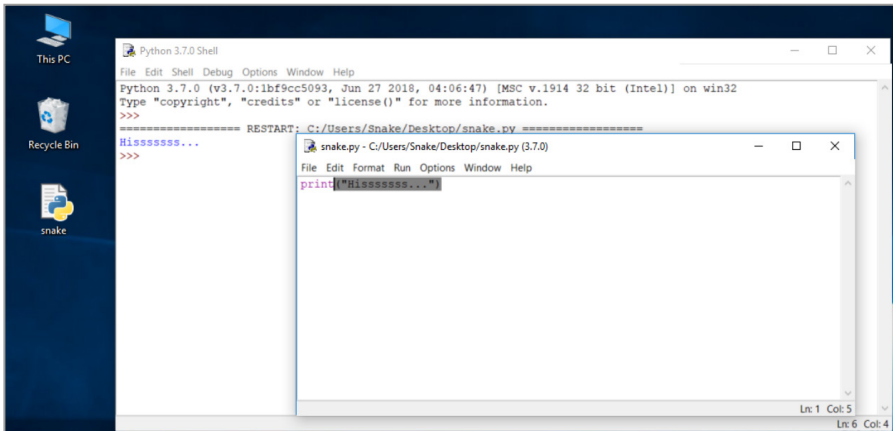


Рисунок 20

Кожного разу, коли ви ставите дужку у програмі, IDLE відображатиме ту частину тексту, яка обмежена парою

відповідних дужок. Це дозволить вам не забувати **додавати їх парами**.

Видаліть закриваючу дужку ще раз. Код видає помилку. Тепер це синтаксична помилка. IDLE не повинен дозволити вам запустити його. Спробуйте ще раз запустити програму. IDLE нагадає вам зберегти змінений файл. Дотримуйтесь інструкцій. Уважно дивіться на усі вікна.

З'явиться нове вікно з повідомленням, що інтерпретатор зіткнувся з EOF (*end-of-file*, кінець файлу) хоча (на його думку) код повинен містити ще якийсь текст.

Вікно редактора чітко вказує, де це сталося (рис. 21).

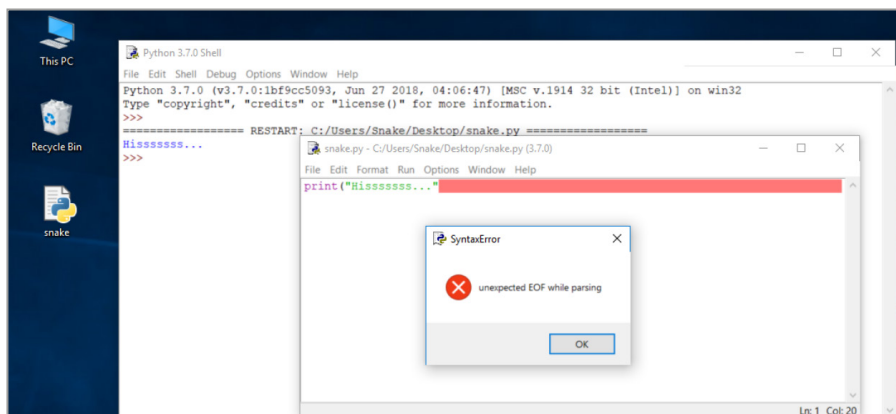


Рисунок 21

Виправте код. Це має виглядати так:

```
print("Hissssssss...")
```

Запустіть його, щоб побачити, чи «шипить» він знову. Давайте зіпсуємо код ще раз. Видаліть одну літеру зі слова **print**. Запустіть код, натиснувши **F5**. Як бачите, Python не може розпізнати помилку (рис. 22).

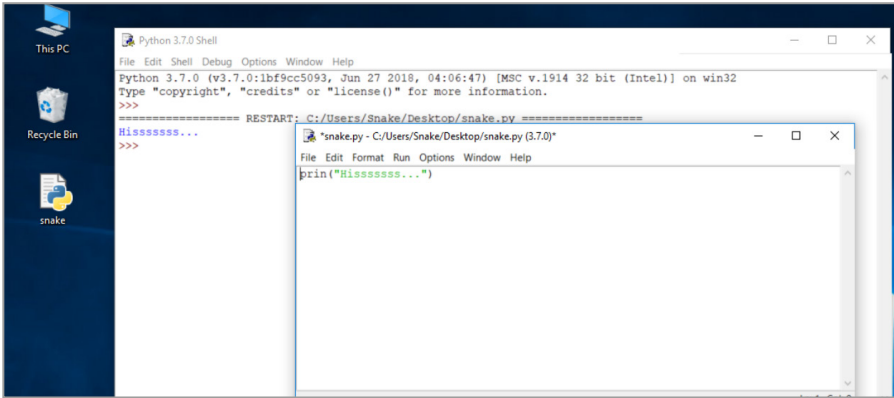


Рисунок 22

Можливо, ви помітили, що повідомлення про помилку, згенеровано для попередньої помилки, дуже відрізняється від першого (рис. 23).

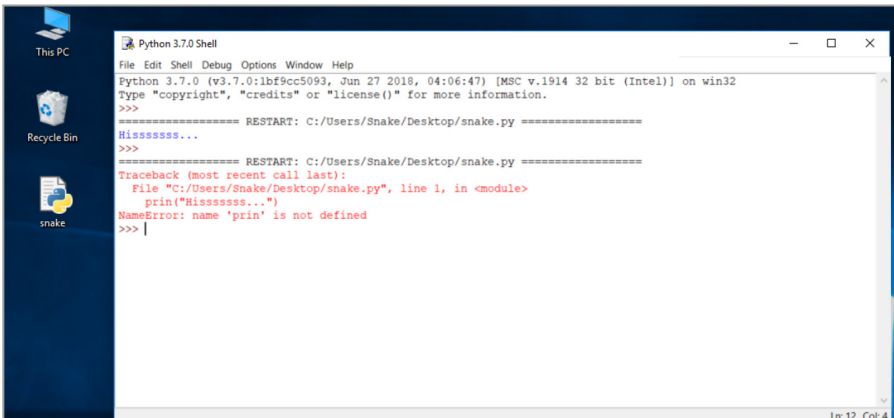


Рисунок 23

Це тому, що природа помилки інша, і помилка виявлена на іншому етапі інтерпретації.

Вікно редактора не дасть жодної корисної інформації про помилку, а от вікна консолі можуть.

Повідомлення (червоним) показує (у наступних рядках):

- зворотне трасування (це шлях, яким проходить код через різні частини програми. Ви можете поки що ігнорувати його, оскільки в такому простому коді він порожній);
- місцезнаходження помилки (ім'я файлу, який містить помилку, номер рядка та ім'я модуля); примітка: цифра може вводити в оману, оскільки Python зазвичай показує місце, де він вперше помічає наслідки помилки, а не саму помилку;
- зміст помилкового рядка; примітка: вікно редактора IDLE не показує номери рядків, але відображає точне положення курсору в нижньому правому куті; використовуйте його, щоб знайти рядок з помилкою у довгому початковому коді;
- назва помилки та коротке пояснення.

Проекспериментуйте зі створенням нових файлів та запуском вашого коду. Спробуйте вивести на екран інше повідомлення, наприклад «*roar!*», «*meow*» або навіть «*oink!*». Спробуйте зіпсувати та виправити свій код, і подивіться, що вийде.

4. Типи даних, змінні, основні операції вводу-виводу, основні оператори

4.1. Пишемо першу програму Hello, World!

Настав час почати писати справжній, працюючий код на Python. Він буде дуже простим. Враховуючи, що ми розбиратимемо фундаментальні поняття та терміни, ці фрагменти коду не будуть серйозними чи складними.

Запустіть IDLE, створіть новий початковий файл Python, заповніть його цим кодом: `print("Hello, World!")`, назвіть файл будь-якою назвою і збережіть його. Тепер запустіть код. Якщо все піде добре, ви побачите рядок у вікні консолі IDLE. Код, який ви запустили, має бути знайомим. Ви бачили щось дуже схоже, коли ми налаштовували середовище IDLE.

Тепер нам потрібно витратити якийсь час на пояснення того, що ви насправді бачите, і чому це виглядає саме так. Як бачите, перша програма складається з наступних частин:

- слово «*print*»;
- відкриваюча кругла дужка;
- перша пара лапок;
- рядок тексту: «*Hello, World!*»;
- ще одна пара лапок;
- закриваюча кругла дужка.

Кожна з перелічених вище складових відіграє дуже важливу роль у коді.

4.2. Функція `print()`

Подивіться на рядок коду нижче:

```
print("Hello, World!")
```

Слово «**print**» — це ім'я функції. Це не означає, що де би не з'явилося слово, воно завжди буде ім'ям функції. Значення слова походить із контексту, в якому це слово було використане.

Ви, мабуть, неодноразово зустрічали термін «функція» на уроках математики. А ще ви, напевно, можете згадати кілька назв математичних функцій, таких, як синус або логарифм.

Але функції Python більш гнучкі і можуть вміщати більше, ніж їх математичні брати.

Функція (у цьому контексті) — це окрема частина комп'ютерного коду, яка:

- викликає деякий ефект (наприклад, відправляє текст у термінал, створює файл, рисує зображення, відтворює звук тощо); це щось зовсім нечуване у світі математики;
- обчислює значення або деякі значення (наприклад, квадратний корінь значення або довжини тексту); це те, що споріднює функції Python з математичними поняттями.

Більше того, багато функцій Python можуть виконувати дві вищезгадані функції разом.

Звідки надходять функції?

- Вони можуть надходити від самого Python. Функція **print** є одним із них; ця функція наче бонус, який ви отримуєте разом з Python та його оточенням (вона

вбудована), тобто вам не потрібно робити нічого особливого (наприклад, просити когось зробити що-небудь), якщо вам потрібно нею скористатися;

- вони можуть походити від одного або декількох надбудов Python, які називаються «модулі»; деякі модулі постачаються з Python, інші можуть вимагати окремого налаштування, у будь-якому випадку всі вони мають бути явно пов'язані з вашим кодом (ми скоро покажемо вам, як це зробити);
- ви можете і самі написати їх, додаючи до вашої програми стільки функцій, скільки вам потрібно, щоб зробити її простіше, зрозуміліше та елегантніше.

Ім'я функції має бути зрозумілим (ім'я функції `print` очевидне, що означає «вивести», «відобразити»). Звичайно, якщо ви збираєтеся використовувати вже існуючу функцію, ви не можете впливати на її ім'я, але коли ви почнете писати власні функції, потрібно ретельно продумувати вибір імен.

Як ми вже говорили, функція може мати: ефект та результат. Є також третій, дуже важливий компонент функції — аргумент(-и). Математичні функції зазвичай приймають один аргумент, наприклад: `sin (x)` приймає «`x`», який є мірою кута.

Але функції Python є більш універсальними. Залежно від індивідуальних потреб, вони можуть прийняти будь-яку кількість аргументів — стільки, скільки необхідно для виконання їх завдань.

ПРИМІТКА: *будь-яке число включає нуль — деякі функції Python не потребують аргументів.*

Незважаючи на кількість необхідних/наданих аргументів, функції Python вимагають наявності пари дужок — відкриваючої та закриваючої відповідно.

Якщо ви бажаєте передати функції один або декілька аргументів, потрібно помістити їх всередину дужок. Якщо ви збираєтеся використовувати функцію, яка не приймає жодних аргументів, вам все одно потрібні круглі дужки.

ПРИМІТКА: щоб відрізнити звичайні слова від імен функцій, додайте пару порожніх дужок після імені функції, навіть якщо відповідна функція потребує одного чи кількох аргументів. Це стандартна угода.

Тут ми говоримо про функцію `print()`. Чи є у функції `print()` з нашого прикладу аргументи? Звісно, є, але хто вони? Єдиний аргумент, переданий у функцію `print()` у цьому прикладі — це рядок:

```
print("Hello, World!")
```

Як бачите, рядок відокремлюється лапками. Насправді, лапки утворюють рядок, вони вирізають частину коду і надають йому інше значення.



Рисунок 24

Ви можете уявити, що лапки повідомляють нам щось на зразок: «Текст між нами — не код. Він не призначений для виконання, і ви повинні прийняти його як є». Майже все, що поміщено в лапки, сприйматиметься буквально, не як код, а як дані. Спробуйте проекспериментувати саме з цим рядком: змініть його, введіть нове значення, видаліть частину існуючого значення.

Є декілька способів вказати рядок всередині коду Python, але цього поки що достатньо.

Отже, ви дізналися про дві важливі частини коду: функції та рядок. Ми говорили про них із погляду синтаксису, але тепер настав час обговорити їх і з погляду семантики.

Ім'я функції (у нашому випадку `print`), разом з круглими дужками та аргументом (-ами), утворює виклик функції.

Ми скоро обговоримо це докладніше, а зараз розберемо лише основи. Що відбувається, коли Python зустрічає виклик, подібний до наведеного нижче?

```
function_name(argument)
```

Давайте подивимося:

- по-перше, Python перевіряє, чи є вказане ім'я дозволеним (він переглядає свої внутрішні дані, щоб знайти існуючу функцію з цим ім'ям; якщо цей пошук не вдався, Python перериває код);
- по-друге, Python перевіряє, чи дозволяє умова функції за кількістю аргументів викликати функцію саме таким чином (наприклад, якщо певна функція вимагає саме двох аргументів, будь-який виклик, що передає лише один аргумент, вважатиметься помилковим і перерве виконання коду);

- по-третє, Python ненадовго залишає код і переходить у функцію, яку ви хочете викликати; звичайно, він також приймає ваші аргументи та передає їх функції;
- по-четверте, функція виконує свій код, викликає бажаний ефект (якщо він є), оцінює бажаний результат(-и) (якщо вони є) і завершує завдання;
- і наостанок, Python повертається до вашого коду (у точку одразу після виклику) і відновлює його виконання.

Три важливих питання, на які потрібно відповісти якнайшвидше:

1. Який ефект викликає функція `print()`?

Ефект дуже корисний та дуже видовищний. Функція:

- приймає аргументи (може приймати більше одного аргументу або менше одного аргументу);
- за необхідності, перетворює їх на читабельну форму (ви вже напевно здогадалися, що рядки цього не вимагають, тому що рядок вже й так читабельний);
- і надсилає отримані дані на пристрій виводу (зазвичай консоль); іншими словами, все, що ви поміщаєте у функцію `print()`, з'явиться на екрані.

Не дивно, що тепер ви будете часто використовувати `print()`, щоб побачити результати ваших операцій та обчислень.

2. Яких аргументів очікує `print()`?

Будь-яких. Ми скоро покажемо вам, що функція `print()` здатна працювати практично з усіма типами даних, які пропонує Python. Рядки, числа, символи, логічні

значення, об'єкти — будь-які з них можуть бути успішно передані у `print()`.

3. Яке значення обчислює функція `print()`?

Жодне. Їй достатньо того, що вона виконує, — `print()` нічого не обчислює.

4.3. Функція `print()` — інструкція

Ви вже знаєте, що програма містить один виклик функції. У свою чергу, виклик функції є одним із можливих видів інструкції у Python. Отже, ця програма складається лише з однієї інструкції.

Звичайно, будь-яка складна програма зазвичай містить набагато більше інструкцій ніж одну. Питання в тому, як поєднати більше однієї інструкції в коді Python?

Синтаксис Python досить специфічний у цій галузі. На відміну від більшості мов програмування, Python вимагає, щоб рядок не містив більше однієї інструкції.

Рядок може бути порожнім (тобто він може взагалі не містити інструкції), але він не повинен містити дві, три або більше інструкцій. Це суворо заборонено.

ПРИМІТКА: *Python робить один виняток з цього правила — він дозволяє одній інструкції поширюватися на більш ніж один рядок (що може бути корисним, коли ваш код містить складні конструкції).*

Давайте трохи розширимо код, як на прикладі нижче:

```
print("The itsy bitsy spider climbed up the waterspout.")
print("Down came the rain and washed the spider out.")
```

Запустіть код і подивимось, що з'явилось в консолі:

*The itsy bitsy spider climbed up the waterspout.
Down came the rain and washed the spider out.*

Давайте озвучимо деякі спостереження:

- програма викликає функцію `print()` двічі, і тепер у консолі два окремі рядки. Це означає, що `print()` починає виведення з нового рядка щоразу, коли починається виконання; ви можете змінити цю поведінку або скористатися нею за потреби;
- кожен виклик функції `print()` містить інший рядок, як видно з її аргументу та вмісту консолі, а це означає, що інструкції в коді виконуються в тому самому порядку, в якому вони були поміщені у вихідний файл; наступна інструкція не виконується доти, доки попередня не буде завершена (є деякі винятки з цього правила, але їх поки що можна ігнорувати).

Ми трохи змінили приклад і додали один порожній виклик функції `print()`.

```
print("The itsy bitsy spider climbed up the waterspout.")  
print()  
print("Down came the rain and washed the spider out.")
```

Порожній тому що ми не надали функції жодних аргументів. Запустіть код. Що відбувається? Якщо все добре, ви побачите щось на кшталт цього:

The itsy bitsy spider climbed up the waterspout.

Down came the rain and washed the spider out.

Як бачите, порожній виклик `print()` не такий вже й порожній, як здавалося, якщо він виводить порожній рядок або (ця інтерпретація теж правильна) його висновок — просто новий рядок.

Це не єдиний спосіб додати новий рядок у консолі виведення. Зараз ми покажемо вам інший спосіб.

4.4. Функція `print()` — екранування символів та перехід на новий рядок

Ми знову змінили код. Уважно подивіться на нього.

```
print("The itsy bitsy spider \nclimbed up the waterspout.")  
print()  
print("Down came the rain \nand washed the spider out.")
```

Рисунок 25

У ньому дві дуже маленькі зміни — ми помістили у вірш дивну пару символів, які виглядають так: `\n`. Цікаво, що людина бачить два символи, а Python — один. Зворотна скісна риска (`\`) має особливе значення при використанні всередині рядків — це називається «екрануванням символу» (*escape character*). Слово «*escape*» (втеча) тут слід розуміти буквально — це означає, що послідовність символів у рядку ненадовго «втікає» (дуже ненадовго) і замінюється на спеціальну підстановку.

Інакше кажучи, зворотна скісна риска сама по собі нічого не означає, це лише своєрідне оголошення про те, що наступний символ після зворотної скісної риски також має інше значення. Літера «`n`», яка йде за зворотною скісною рискою, це скорочення від «*newline*» (новий рядок).

І зворотна скісна риска, і літера **n** формують спеціальний символ, який називається «символ нового рядка». Він змушує консоль почати виведення з нового рядка.

Запустіть код. Тепер ваша консоль має виглядати так:

```
The itsy bitsy spider  
climbed up the waterspout.
```

```
Down came the rain  
and washed the spider out.
```

Як бачите, два нові рядки з'являються у вірші, у тому місці, де використовується символ `\n`.

Ця угода має два важливі наслідки:

1. Якщо ви хочете помістити тільки одну зворотну скісну риску всередині рядка, не забувайте про її екрановану властивість, — ви повинні подвоїти її, наприклад, такий виклик викличе помилку: `print("\n")`, а такий — ні: `print("\\n")`.
2. Не всі пари в екрануванні символів (зворотна скісна риска у поєднанні з іншим символом) щось означають.

Проекспериментуйте з вашим кодом у редакторі, запустіть його та подивіться, що станеться.

4.5. Функція `print()` — використання декількох аргументів

Досі ми тестували поведінку функції `print()` або без аргументів, або з одним аргументом. Варто також спробувати додати до функції `print()` два й більше аргументів. Подивіться на код:

```
print("The itsy bitsy spider" , "climbed up" ,  
      "the waterspout.")
```

У ньому один виклик функції `print()`, але він містить **три аргументи**. Усі вони — рядки. Аргументи **розділені комами**. Ми виділили їх пробілами для того, щоб вони стали помітнішими, але в цьому немає необхідності і ми більше так не робитимемо.

У цьому випадку коми, що розділяють аргументи, грають зовсім іншу роль, ніж кома всередині рядка. Перший є частиною синтаксису Python, останній призначений для відображення в консолі. Якщо ви подивитеся на код ще раз, ви побачите, що всередині рядків немає пробілів.

Запустіть код і подивіться, що станеться. Тепер консоль має відображати наступний текст:

The itsy bitsy spider climbed up the waterspout.

Пробіли, видалені з рядків, з'явилися знову. Ви можете пояснити, чому?

З цього прикладу випливають два висновки:

- якщо функція `print()` викликається з двома та більше аргументами, вона виводить їх в одному рядку;
- функція `print()` ставить пробіл між виведеними аргументами за власною ініціативою.

4.6. Функція `print()` — позиційний спосіб передачі аргументів

Тепер, коли ви знаєте про порядок використання функції `print()`, давайте подивимося, як його змінювати.

Ви вже можете передбачити результат, не запускаючи цей код в редакторі.

```
print("My name is", "Python.")  
print("Monty Python.")
```

Те, як ми передаємо аргументи у функцію `print()`, є найбільш поширеним способом у Python, і називається **позиційними аргументами** (назва з'явилася через те, що значення аргументу продиктовано його позицією, наприклад, другий аргумент буде виведено після першого, а не навпаки).

Запустіть код та перевірте, чи відповідає виведення вашим передбаченням.

4.7. Функція `print()` — ключові аргументи

Python пропонує інший механізм для передачі аргументів, який може бути корисним, якщо ви хочете, щоб функція `print()` трохи змінила свою поведінку. Ми не будемо зараз це докладно пояснювати. Ми плануємо зробити це, коли говоритимемо про функції взагалі. Наразі ми просто хочемо показати вам, як це працює. А ви можете сміливо використовувати це у своїх програмах.

Такий механізм називається «**ключові аргументи**». Назва відображає той факт, що значення цих аргументів береться не з його розташування (позиції), а зі спеціального (ключового) слова, що використовується для їхньої ідентифікації.

У функції `print()` є два ключові аргументи, які ви можете використовувати для своїх цілей. Перший з них

називається **end**. Нижче наведено дуже простий приклад використання ключового аргументу:

```
print("My name is", "Python.", End=" ")
print("Monty Python.")
```

Перш ніж використовувати його, потрібно знати деякі правила:

- ключовий аргумент складається з трьох елементів: **ключове слово**, яке визначає аргумент (тут, **end**); **знак «дорівнює» (=)**; і **значення**, призначене на цей аргумент;
- будь-які ключові аргументи повинні йти **після останнього позиційного аргументу** (це дуже важливо). У нашому прикладі ми використали ключовий аргумент **end** і помістили його в рядок, що містить один пробіл.

Запустіть код, щоб побачити, як він працює. Тепер консоль має відображати наступний текст:

My name is Python. Monty Python.

Як бачите, ключовий аргумент **end** визначає символи, які функція **print()** відправляє для виведення, щойно доходить до кінця своїх позиційних аргументів.

Поведінка за умовчанням відображає ситуацію, коли ключовий аргумент **end** використовується **опосередковано** наступним чином: **end="\n"**.

А тепер настав час спробувати щось складніше. Якщо ви уважно подивитеся, ви побачите, що ми використали аргумент **end**, але призначений йому рядок є порожнім (в ньому немає символів взагалі).

Що буде далі? Запустіть програму в редакторі, щоб побачити. Через те, що аргумент `end` не містить нічого, функція `print()` також нічого не виводить, коли її позиційні аргументи вичерпані.

Тепер консоль має відображати наступний текст:

My name is Monty Python.

ПРИМІТКА: нові рядки не були надіслані на виведення.

Рядок, наданий ключовому аргументу `end` може бути будь-якої довжини. Проекспериментуйте з ним.

Раніше ми говорили, що функція `print()` поділяє свої виведені аргументи пробілами. Цю поведінку також можна змінити. Ключовий аргумент, який може це зробити, називається `sep` (від «*separator*» — роздільник).

Подивіться на код нижче і запустіть його.

```
print("My", "name", "is", "Monty", "Python.", sep="-")
```

Аргумент `sep` видає наступний результат:

My-name-is-Monty-Python.

Тепер функція `print()` використовує тире замість пробілу для розділення виведених аргументів.

ПРИМІТКА: значенням аргументу `sep` може бути й порожній рядок. Спробуйте самі.

Обидва ключові аргументи можуть бути використані в одному виклику так само, як у коді, наведеному нижче.

```
print("My", "name", "is", sep="_", end="*")
print("Monty", "Python.", sep="*", end="*\n")
```

Приклад не має сенсу, але він наочно показує взаємодію між `end` та `sep`. Чи можете ви передбачити, що отримаємо на виході? Запустіть код і подивіться, чи відповідає він вашим прогнозам?

Тепер, коли ви розумієте функцію `print()`, ви готові розглянути зберігання та обробку даних у Python.

Без `print()` ви не зможете побачити жодних результатів.

4.8. Ключові висновки

1. Функція `print()` є вбудованою функцією. Вона виводить вказане повідомлення на екран/консоль.
2. Вбудовані функції, на відміну від функцій користувача, завжди доступні і не потребують імпорту. Python 3.7.1 має 69 вбудованих функцій. Повний список в алфавітному порядку знаходиться в Стандартній бібліотеці Python (*Python Standard Library*).
3. Для виклику функції потрібно використовувати ім'я функції та круглі дужки. Можна передати аргументи у функцію, помістивши їх у круглі дужки. Розділіть аргументи комою, наприклад: `print("Hello, ", "world!")`. «Порожня» функція `print()` виводить порожній рядок.
4. Рядки Python розділені лапками, наприклад: «Я рядок» або 'Я теж рядок'.
5. Комп'ютерні програми — це набори інструкцій. Інструкція — це команда, яка виконує певне, поставлене завдання під час запуску, наприклад: виведення певного повідомлення на екран.
6. У рядках Python зворотна скісна риска (`\`) є спеціальним символом, який повідомляє, що наступний символ має

інше значення, наприклад, `\n` (символ нового рядка) починає виведення з нового рядка.

7. Позиційні аргументи — це ті, суть яких визначається їх станом, наприклад: другий аргумент виводиться після першого, третій виводиться після другого і т. д.
8. Ключові аргументи — це ті, значення яких визначається не їх розташуванням, а спеціальним (ключовим) словом, що використовується для їх ідентифікації.
9. Параметри `end` та `sep` можуть використовуватися для форматування виведення функції `print()`. Параметр `sep` вказує на роздільник між аргументами (наприклад, `print("H", "E", "L", "L", "O", sep="-")`), тоді як параметр `end` вказує, що виводиться наприкінці оператора `print`.

5. Літерали Python

5.1. Літерали — дані в собі

Тепер, коли ви трохи познайомилися з деякими потужними особливостями функції `print()`, настав час дізнатися про нові аспекти і запам'ятати один важливий термін — літерал. **Літерали** — це дані, значення яких визначаються самим літералом.

Оскільки це складна для розуміння концепція, звернімося до прикладу. Погляньте на наступний набір цифр: «123». Ви можете вгадати, яке значення він має? Звичайно можете — 123. Але як щодо цього: «c».

У нього є якесь значення? Можливо. Це може бути символом швидкості світла, наприклад. Це також може бути постійною інтегрування. Або навіть довжиною гіпотенузи за теоремою Піфагора. Тут багато варіантів. Не маючи додаткової інформації, складно сказати, який з них правильний. А ось і підказка: 123 — це літерал, а c — ні.

Літерали використовуються, щоб кодувати дані та поміщати їх у свій код. Тепер нам потрібно розглянути деякі угоди, які ви повинні дотримуватися під час програмування на Python.

Почнемо з простого експерименту — подивіться на фрагмент коду нижче.

```
print("2")
print(2)
```

Перший рядок виглядає знайомим. У другому, здається, є помилка, бо не вистачає лапок.

Спробуйте запустити його. Якщо все пройшло добре, ви побачите два однакові рядки. Що сталося? Що це означає? У цьому прикладі два різні типи літералів:

- рядок, з яким ви вже знайомі;
- ціле число — щось зовсім нове.

Функція `print()` виводить їх абсолютно однаково — цей приклад очевидний, тому що їх легкозчитуване представлення також однакове. Внутрішньо, у пам'яті комп'ютера, ці два значення зберігаються зовсім по-різному: рядок існує просто як рядок, тобто набір літер. Число перетворюється на машинне представлення (набір бітів). Функція `print()` виводить їх у зрозумілій для людини формі.

Зараз ми розглянемо числові літерали і чим вони живуть.

5.2. Цілі числа (Integers)

Можливо, ви вже знаєте про те, як комп'ютери виконують операції з числами. Можливо, ви чули про двійкову систему числення і знаєте, що це система, яку комп'ютери використовують для зберігання чисел, і що вони можуть виконувати з ними будь-які операції.

Ми не вивчатимемо тут тонкощі позиційних систем числення, але ми скажемо, що числа, які обробляють сучасні комп'ютери, бувають двох типів:

- цілі, тобто ті, що позбавлені дробової частини;
- і з рухомою крапкою, що містять (або можуть містити) дробову частину.

Це визначення не зовсім точне, але поки що його цілком достатньо. Ця різниця дуже важлива, і межа між цими двома типами чисел дуже чітка. Обидва типи чисел істотно різняться по тому, як вони зберігаються в пам'яті комп'ютера, і по діапазону допустимих значень.

Характеристика числового значення, що визначає його вид, діапазон та застосування, називається типом.

Якщо закодувати літерал і помістити його в код Python, форма літерала визначає уявлення (тип), яке Python буде використовувати для його зберігання в пам'яті.

А поки що відкладемо числа з рухомою крапкою (ми скоро до них повернемося) і розглянемо питання, як Python розпізнає цілі числа.

Цей процес дуже схожий на те, як би ви написали їх олівцем на папері — це просто ряд цифр, які становлять число. Але із застереженням — ви не можете вставляти символи, які не є цифрами всередині числа.

Взяти, наприклад, число одинадцять мільйонів сто одинадцять тисяч сто одинадцять. Якби ви прямо зараз взяли в руку олівець, ви би написали число наступним чином: `11,111,111` або ось так: `11.111.111`, або навіть так: `11 111 111`.

Зрозуміло, що такий запис полегшує читання, особливо коли число складається із великої кількості цифр. Тим не менш, Python таких речей не розуміє. Це заборонено. А от що Python дозволяє, так це використання підкреслення у числових літералах.

ПРИМІТКА. У Python 3.6 введені підкреслення у числових літералах, що дозволяють розміщувати одинарні підкреслення між цифрами та специфікатори

після основи, щоб спростити читання великих чисел. Ця функція недоступна у старіших версіях Python.

Отже, ви можете написати це число так: `11111111` або ось так: `11_111_111`. Як від'ємні числа кодуються в Python? Як завжди — додається знак «мінус». Можна написати: `-11111111` або `-11_111_111`.

Додатні числа не обов'язково супроводжуються знаком «плюс», але це припустимо, якщо вам це потрібно. Наступні рядки описують те саме число: `+11111111` і `11111111`.

5.3. Цілі числа: вісімкові та шістнадцяткові числа

У Python є дві додаткові угоди, які не відомі світу математики. Перше дозволяє нам використовувати числа у **вісімковому** представленні. Якщо за цілим числом йде префікс `0O` або `0o` (нуль-о), Python сприйматиме його як вісімкове значення. Це означає, що число має містити лише ті цифри, що знаходяться в діапазоні `[0...7]`.

`0o123` є **вісімковим** числом, десяткове значення якого дорівнює `83`.

Функція `print()` виконує перетворення автоматично. Спробуйте запустити цей код:

```
print(0o123)
```

Друга угода дозволяє нам використовувати **шістнадцяткові** числа. Після таких чисел має йти префікс `0x` або `0X` (нуль-х).

`0x123` — це **шістнадцяткове** число, десяткове значення якого дорівнює `291`. Функція `print()` може керувати і цими значеннями. Спробуйте запустити цей код:

```
print(0x123)
```

5.4. Числа з рухомою крапкою (Floating-point numbers)

Тепер поговоримо про інший тип, який призначений для представлення та зберігання чисел, які (як сказав би математик) мають **не пустий десятковий дріб**. Це числа, в яких є (або може бути) дробова частина після десяткової крапки, і хоча таке визначення дуже погане, цього, безумовно, достатньо для того, що ми розбиратимемо.

Щоразу, коли ми використовуємо такий термін, як «два з половиною» або «мінус нуль цілих чотири», ми маємо на увазі числа, які комп'ютер сприймає як числа з **рухомою крапкою**:

```
2.5 -0.4
```

ПРИМІТКА: *число два з половиною виглядає абсолютно нормально, коли ви записуєте його у програмі, але обов'язково переконайтеся, що у вашому числі немає інших ком.*

Python або не зможе його прочитати взагалі, або (у дуже рідких, але можливих випадках) може неправильно вас зрозуміти, оскільки сама кома має власне значення, зарезервоване в Python. Якщо ви бажаєте використовувати просто значення «два з половиною», ви маєте написати

його так, як показано вище. Зверніть увагу ще раз — між 2 та 5 стоїть крапка, а не кома.

Як ви вже, напевно, здогадалися, число «нуль цілих чотири» може бути написано на Python так: 0.4. Але не забувайте одне просте правило — ви можете опустити нуль, якщо це єдина цифра перед або після десяткової коми.

По суті ви можете написати значення 0.4 так: .4. А значення 4.0 може бути написано так: 4.. Це не змінить ні його тип, ні його значення.

Десятковий роздільник дуже важливий для розпізнавання чисел з рухомою крапкою в Python. Погляньте на ці два числа: 4 та 4.0. Можна подумати, що вони однакові, але Python бачить їх зовсім по-різному.

4 — це ціле число, тоді як 4.0 — це число з рухомою крапкою. «Рухається» саме крапка. З іншого боку, «рухаються» не лише крапки. Літера e має такий самий ефект.

Якщо ви хочете використати дуже великі або дуже маленькі числа, ви можете використати експоненційне представлення числа. Взяти, наприклад, швидкість світла, виражену в метрах за секунду. Якщо записати її буквально, це виглядатиме так: 300000000.

Щоб не записувати так багато нулів, у підручниках з фізики використовується скорочена форма запису, яку ви, мабуть, вже бачили: 3×10^8 . Читається це так: «три, помножене на десять у восьмому степені».

У Python той же ефект досягається трохи іншим способом:

3E8

Літера **E** (можна використати і малу літеру **e**, від слова «*exponent*» — степінь) — це короткий запис фрази «стільки-то, помножене на десять у такому-то степені».

ПРИМІТКА: *ступінь (значення після E) має бути цілим числом; основа (значення перед E) може бути цілим числом.*

5.5. Кодування чисел з рухомою крапкою

Давайте подивимося, як ця угода використовується для запису дуже маленьких чисел (в сенсі їх абсолютного значення, близького до нуля).

Фізична константа називається постійною Планка (і позначається як h), згідно з підручниками, вона становить: $6,62607 \times 10^{-34}$.

Якщо ви хочете використати її в програмі, записується вона так: 6.62607E-34.

ПРИМІТКА: той факт, що ви вибрали одну з можливих форм представлення значень з рухомою крапкою, не означає, що Python представить її в тій самій формі.

Іноді Python вибирає **іншу форму запису**. Наприклад, припустимо, ви вирішили використати наступний літерал з рухомою крапкою: `0.000000000000000000000001`.

Коли ви запустите цей літерал через Python:

[illegible]

результат буде таким:

 $1e-22$

Python завжди вибирає **більш економічну форму представлення числа**, і ви повинні врахувати це під час створення літералів.

5.6. Рядки

Рядки використовуються, коли вам потрібно обробити текст (наприклад, різні імена, адреси, романи і т. д.), а не цифри.

Ви вже трохи знаєте про них, наприклад, що **рядкам потрібні лапки**, так само й числам з рухомою крапкою потрібні крапки.

Ось повністю стандартний рядок: «*Я рядок*». Але тут є одна проблема. Вона полягає в тому, як закодувати лапки всередині рядка, якщо вони вже розділені лапками.

Наприклад, ми хочемо надрукувати дуже просте повідомлення: *I like «Monty Python»*. Як ми це зробимо без помилок? Є два можливі рішення.

Перше засноване на вже відомій нам концепції **екранування символів**, яке вводиться за допомогою **зворотної скісної риски**. Зворотна скісна риска може екранувати і лапки. Лапки, які йдуть після зворотної скісної риски, змінюють своє значення — це не роздільник, а просто лапки. Так що це спрацює, як і має бути:

```
print("I like \"Monty Python\"")
```

ПРИМІТКА: *всередині рядка є дві пари екранованих лапок — ви усі бачите?*

Друге рішення трохи неординарне. Python може використовувати **апостроф замість лапок**. Будь-який

з цих символів може розділяти рядки, але будьте **послідовні**.

Якщо ви відкриваєте рядок з лапками, ви повинні закрити його з лапками. Якщо ви починаєте рядок з апострофа, ви й закінчувати його маєте апострофом.

Цей приклад також працюватиме:

```
print('I like "Monty Python"')
```

ПРИМІТКА: *тут екранування не потрібне.*

5.7. Кодування рядків

Тепер наступне питання: як вставити апостроф у рядок, якщо він вже оточений апострофами? Ви вже повинні знати відповідь, або, якщо точніше, дві можливі відповіді. Спробуйте вивести рядок, який містить таке повідомлення: *I'm Monty Python*.

Як бачите, зворотна скісна риска — дуже потужний інструмент, який може екранувати не лише лапки, а й апострофи. Ми вже це продемонстрували, але давайте ще раз зробимо акцент на цьому явищі — **рядок може бути порожнім**, тобто в ньому може взагалі не бути символів.

Порожній рядок — все ще рядок: `' '`, `''`.

5.8. Булеві значення (логічні типи даних)

Щоб закінчити тему літералів у Python, давайте розглянемо ще два. Вони не такі очевидні, як попередні, тому що використовуються для представлення дуже абстрактного значення — правдивості.

Щоразу, коли ви запитуєте Python, чи є одне число більшим за інше, це питання призводить до створення деяких конкретних даних — логічних (булевих) значень.

Джордж Буль (1815-1864) — автор фундаментальної роботи «Дослідження законів мислення», в якій дано визначення булевої алгебри — це розділ алгебри, який використовує лише два значення: правда та хибність, які позначаються як **1** та **0** відповідно.

Програміст пише програму, а програма ставить запитання. Python виконує програму і дає відповіді. Програма має вміти реагувати відповідно до отриманих відповідей. На щастя, комп'ютери знають лише два види відповідей: «Так, це правда», «Ні, це брехня». Ви ніколи не отримаєте відповідь типу: «Я не знаю» або «Напевно, так, але точно не знаю». Python — двійкова «рептилія».

У цих логічних значень є суворі позначення в Python: **True** (Правда) **False** (Хибність). Ви не можете нічого змінити — ви повинні прийняти ці символи такими, якими вони є, включаючи регістр.

Завдання: яким буде результат наступного коду?

```
print(True > False) print(True < False)
```

Запустіть код, щоб перевірити. Чи можете ви пояснити результат?

5.9. Ключові висновки

1. **Літерали** — запис, який використовується для представлення деяких фіксованих значень коду. В Python є різні типи літералів, наприклад, літерал може бути

числом (числові літерали: `123`) або рядком (рядкові літерали: `«I am a literal.»`).

2. **Двійкова система** — це система чисел з основою `2`. Тому двійкове число може складатися тільки з `0` та `1`, наприклад: `1010` — це `10` у десятковій системі числення.
3. Основи вісімкової та шістнадцяткової систем числення — `8` та `16` відповідно. Шістнадцяткова система використовує десяткові числа та шість додаткових літер.
4. **Цілі числа** (`integers`, `ints`) — один із числових типів, який підтримує Python. Це числа, написані без дробової складової, наприклад, `256` або `-1` (від'ємні цілі числа).
5. Числа з **рухомою крапкою** (`floats`) — ще один числовий тип, який підтримує Python. Це числа, які містять (або можуть містити) дробову частину, наприклад, `1.27`.
6. Щоб закодувати апостроф або лапки всередині рядка, можна використовувати екранування символу, наприклад, `'I'm happy.'` або відкриваючі та закриваючі символи, протилежні тим, які ви хочете кодувати, наприклад, в `«I'm happy.»` закодований апостроф, а в `'He said «Python», not «typhoon»'` закодовані (подвійні) лапки.
7. **Логічні (булеві) значення** — це значення `True` і `False`, які використовуються для представлення істинності значень (у числовому виразі `1` — Правда, а `0` — Хибність).

Є ще один спеціальний літерал, який використовується в Python — `None`. Цей літерал є так званим `NoneType`-об'єктом і використовується для представлення відсутності значення. Незабаром ми обговоримо його докладніше.

6. Оператори — інструменти керування даними

6.1. Python як калькулятор

Тепер ми покажемо вам абсолютно новий бік функції `print()`. Ви вже знаєте, що ця функція може показати вам значення літералів, які передаються їй через аргументи.

Насправді вона може набагато більше. Подивіться на фрагмент коду:

```
print(2+2)
```

Скопіюйте код в редактор і запустіть його. Чи можете ви вгадати, що з'явиться на екрані? Має бути цифра «чотири». Не бійтеся експериментувати з іншими операторами.

Ось так легко і просто ви щойно виявили, що Python можна використовувати як калькулятор. Не самий зручний і, звичайно, не кишеньковий, але калькулятор.

Якщо глянути на нього трохи серйозніше, ми зараз входимо в область операторів і виразів.

6.2. Основні оператори

Оператор — це символ мови програмування, що оперує значеннями. Наприклад, як і в арифметиці, знак `+` (плюс) — це оператор, який може скласти два числа і видати результат додавання.

Однак не всі оператори Python так само очевидні, як знак додавання. Тому давайте розглянемо деякі оператори,

доступні в Python, і пояснемо, які правила визначають їх використання і як інтерпретувати операції, які вони виконують.

Почнемо з операторів, які пов'язані з найбільш відомими арифметичними операціями: `+`, `-`, `*`, `/`, `//`, `%`, `**`. Їх послідовність у цьому рядку не випадкова. Ми поговоримо про це докладніше, коли пройдемо їх усі.

ЗАПАМ'ЯТАЙТЕ! *Дані та оператори при з'єднанні утворюють вирази. Найпростішим виразом є сам літерал.*

6.3. Арифметичні оператори: піднесення до степеня

Знак `**` (подвійна зірочка) — оператор піднесення до степеня. Його лівий аргумент є **основою**, а правий — **степенем**.

Класична математика віддає перевагу запису з верхнім індексом, ось такий: 2^3 . «Чисті» текстові редактори не розуміють такого запису, тому Python використовує `**` замість нього, наприклад: `2 ** 3`.

Подивіться на приклади нижче.

```
print(2 ** 3)
print(2 ** 3.)
print(2. ** 3)
print(2. ** 3.)
```

ПРИМІТКА: у цьому прикладі ми оточили подвійні зірочки пробілами. Це не обов'язково, але це покращує читання коду.

Приклади показують дуже важливу особливість практично всіх **числових операторів** у Python. Запустіть код і уважно подивіться на результат. Чи є тут якась закономірність?

ЗАПАМ'ЯТАЙТЕ! *На основі цього результату можна сформулювати наступні правила:*

- якщо аргументи по обидві сторони від ****** є цілими числами, результат також буде цілим числом;
- якщо **хоча б один** аргумент ****** буде числом з рухомою крапкою, результат теж буде числом з рухомою крапкою.

Це дуже важливо, тому пам'ятайте про це.

6.4. Арифметичні оператори: множення

Знак ***** (зірочка) — оператор множення. Запустіть наведений нижче код і перевірте, чи працює наше правило цілих чисел і чисел з рухомою крапкою.

```
print(2 * 3)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
```

6.5. Арифметичні оператори: ділення

Знак **/** (скісна риска) — оператор ділення. Значення перед скісною рисою — ділене, значення після скісної риски — дільник.

Запустіть наведений нижче код та проаналізуйте результати.

```
print(6 / 3)
print(6 / 3.)
print(6. / 3)
print(6. / 3.)
```

Ви маєте знати, що є й виняток із правила. Результат, отриманий оператором ділення, завжди є числом з рухомою крапкою незалежно від того, чи буде результат, на перший погляд, десятковим числом: $1 / 2$ або він, швидше за все, буде цілим числом: $2 / 1$.

Чи може це стати проблемою? Так. Іноді трапляється так, що вам вкрай необхідне ділення, яке дає ціле значення, а не дріб. На щастя, Python може впоратися і з цим.

6.6. Арифметичні оператори: ділення цілих чисел

Знак `//` (подвійна скісна риска) — оператор **ділення цілих чисел**. Він відрізняється від стандартного оператора `/` у двох ситуаціях:

- в його результаті немає дробової частини. Вона відсутня (для цілих чисел) або завжди дорівнює нулю (для чисел з рухомою комою); це означає, що **результати завжди округляються**;
- це відповідає правилу цілих чисел і чисел з рухомою крапкою.

Запустіть приклад нижче та подивіться на результати:

```
print(6 // 3)
print(6 // 3.)
print(6. // 3)
print(6. // 3.)
```

Як бачите, якщо розділити ціле число на ціле число, то й **результат буде цілим числом**. У решті випадків ви отримаєте дріб.

Давайте виконаємо декілька складніших тестів. Погляньте на наступний фрагмент коду:

```
print(6 // 4)
print(6. // 4)
```

Уявімо, що ми використали `/` замість `//`. Ви зможете передбачити результат? Так, вийде `1.5` в обох випадках. Це зрозуміло. Але яких результатів нам очікувати з діленням через оператор `//`?

Запустіть код і подивіться. Ми отримуємо два числа — одне ціле та одне з рухомою крапкою. Результат ділення цілих чисел завжди округляється до найближчого цілого значення, яке менше справжнього (не округленого) результату. Це дуже важливо: **округлення завжди переходить до меншого цілого числа**.

Подивіться на код нижче і спробуйте спрогнозувати результати ще раз:

```
print(-6 // 4)
print(6. // -4)
```

ПРИМІТКА: деякі значення від'ємні. Вочевидь, це вплине на результат. Але як? Результат — дві від'ємні двійки. Реальний (не округлений) результат `-1.5` в обох випадках. Проте результати завжди округляються. Округлення йде до меншого цілого значення, а менше ціле — це `-2`, звідси: `-2` та `-2.0`.

6.7. Оператори: остача (ділення за модулем, з остачею)

Наступний оператор є досить своєрідним, тому що він не має еквівалента серед традиційних арифметичних операторів.

Графічно в Python він виглядає так: `%` (знак відсотка), який може спантеличити невідповідну людину. Просто думайте про це як про скісну риску (оператор ділення), яка оточена двома маленькими кружочками. Результатом оператора є **остача після ділення цілого числа**.

Іншими словами, це значення, яке залишається після ділення одного значення на інше, щоб отримати ціле число в частці.

ПРИМІТКА: *в інших мовах програмування цей оператор іноді називають діленням за модулем.*

Подивіться на фрагмент коду нижче та спробуйте передбачити його результат, а потім запустіть його:

```
print(14 % 4)
```

Як бачите, результат буде **2**. І ось чому:

- $14 // 4$ буде **3** → це ціле число, **частка**;
- $3 * 4$ буде **12** → як результат **множення частки на дільник**;
- $14 - 12$ буде **2** → це **остача**,

Наступний приклад трохи складніший:

```
print(12 % 4.5)
```

Яким буде результат?

Відповідь: 3.0 — не 3, а 3.0 (правило все ще працює: $12 // 4.5$ буде 2.0; $2.0 * 4.5$ буде 9.0; $12 - 9.0$ буде 3.0).

6.8. Оператори: як не ділити

Як ви, мабуть, знаєте, **ділення на нуль не працює**.

Не намагайтеся робити наступне:

- ділити на нуль;
- ділити ціле число на нуль;
- знаходити остачу від ділення на нуль.

6.9. Оператори: додавання

Оператор додавання — це знак **+** (плюс), що повністю відповідає математичним стандартам.

Знову ж таки, погляньте на фрагмент програми нижче:

```
print(-4 + 4)
print(-4. + 8)
```

Результат не повинен вас здивувати. Запустіть код, щоб перевірити

6.10. Оператор віднімання, унарні та бінарні оператори

Оператором віднімання, очевидно, буде знак **—** (мінус), хоча слід зауважити, що цей оператор також має й інше значення — він може змінювати знак числа.

Це чудова можливість представити дуже важливу різницю між унарними та бінарними операторами. В операціях віднімання оператор «мінус» приймає два аргументи: лівий (зменшуване число, якщо використовувати математичний термін) і правий (від'ємник). Тому оператор

віднімання вважається одним із бінарних операторів, так само, як оператори додавання, множення та ділення.

Але оператор «мінус» може бути використаний іншим (унарним) способом. Подивіться на останній рядок фрагмента нижче:

```
print(-4 - 4)
print(4. - 8)
print(-1.1)
```

До речі: там є й унарний оператор `+`. Його можна використовувати так:

```
print(+2)
```

Оператор зберігає знак єдиного аргументу справа. Хоча така конструкція синтаксично правильна, її використання не несе суттєвого значення, і буде важко вгадати причину його використання.

Подивіться на фрагмент коду вище — чи можете ви передбачити, що буде виведено на екран?

6.11. Оператори та їх пріоритети

Досі ми розглядали кожен оператор так, ніби він не пов'язаний із рештою операторів. Звичайно, така ідеальна та проста ситуація це рідкість у справжньому програмуванні. Крім того, дуже часто можна зустріти більше одного оператора в одному виразі, і тоді прогнозування результату вже не таке очевидне.

Розглянемо наступний вираз: `2 + 3 * 5`. Ви маєте пам'ятати зі школи, що **спочатку йде множення, а потім**

додавання. Тому, спочатку потрібно **3** помножити на **5**, запам'ятати отриманий результат — **15**, після цього **15** додати до **2**, і в результаті отримаємо **17**.

Феномен, який змушує деякі оператори діяти раніше за інших, називається **ієрархією пріоритетів**. Python точно визначає пріоритети всіх операторів і передбачає, що оператори з вищим пріоритетом виконують свої операції раніше операторів з нижчим пріоритетом.

Отже, якщо ви знаєте, що у множення ***** вищий пріоритет, ніж у додавання **+**, результат має бути для вас очевидним.

6.12. Оператори та зв'язування

Зв'язування оператора визначає послідовність обчислення операторів із рівним пріоритетом, які йдуть послідовно в одному виразі. Більшість операторів у Python мають лівостороннє зв'язування. Це означає, що обчислення виразу здійснюється зліва направо.

Цей простий приклад покаже вам, як це працює. Подивіться:

```
print(9 % 6 % 2)
```

Є два можливі способи оцінити цей вираз:

- зліва направо: перший **9 % 6** дає **3**, потім **3 % 2** дає **1**;
- справа наліво: перший **6 % 2** дає **0**, потім **9 % 0** видає **фатальну помилку**.

Запустіть приклад і подивіться, що вийде. Результат має бути **1**. У цього оператора **лівостороннє зв'язування**. Але є один цікавий виняток.

6.13. Оператори та зв'язування: піднесення до степеня

Повторіть експеримент, але тепер з піднесення до степеня. Скористайтеся таким фрагментом коду:

```
print(2 ** 2 ** 3)
```

Два можливі результати:

- $2 ** 2 \rightarrow 4; 4 ** 3 \rightarrow 64$
- $2 ** 3 \rightarrow 8; 2 ** 8 \rightarrow 256$

Запустіть код. Що ви бачите? Результат чітко показує, що **оператор піднесення до степеня використовує правостороннє зв'язування**.

6.14. Список пріоритетів

Оскільки ви тільки знайомитеся з операторами Python, ми не хочемо зараз давати вам повний список пріоритетів операторів. Натомість ми покажемо вам його скорочену форму і послідовно її розширюватимемо в міру появи нових операторів. Погляньте на таблицю 1.

Таблиця 2

Пріоритет	Оператор	
1	+, -	унарний
2	**	
3	*, /, %	бінарний
4	+, -	

ПРИМІТКА: ми *оказали оператори послідовно, від найвищого (1) до найнижчого (4) пріоритету*.

Спробуйте опрацювати наступний вираз:

```
print(2 * 3 % 5)
```

В обох операторів ($*$ та $\%$) однаковий пріоритет, тому результат можна передбачити лише у тому випадку, якщо ви знаєте напрямок зв'язування. Як думаєте, яким буде результат? Відповідь: 1.

6.15. Оператори та дужки

Звичайно, завжди можна використовувати дужки, які можуть змінити природний порядок обчислень. Відповідно до арифметичних правил, підвирази у дужках завжди обчислюються першими. Ви можете використовувати стільки дужок, скільки вам потрібно, і вони часто використовуються для покращення читання виразу, навіть якщо вони не змінюють послідовність операцій.

Приклад виразу з кількома круглими дужками наведено тут:

```
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
```

Спробуйте самі обчислити значення, яке буде виведено в консоль. Яким буде результат функції `print()`? Відповідь: 10.0.

6.16. Ключові висновки

1. **Вираз** — це комбінація значень (або змінних, операторів, викликів функцій — ви скоро дізнаєтеся про них), яка обчислює значення, наприклад, $1+2$.
2. **Оператори** є спеціальними символами або ключовими словами, які можуть працювати зі значеннями і вико-

нувати (математичні) операції, наприклад оператор $*$ помножує два значення: $x * y$.

3. Арифметичні оператори в Python: $+$ (додавання), $-$ (віднімання), $*$ (множення), $/$ (класичне ділення — повертає значення з рухомою крапкою, якщо одне зі значень є числом з рухомою крапкою), $\%$ (ділення за модулем — ділить лівий операнд на правий операнд і повертає остачу операції, наприклад, $5\% 2 = 1$), $**$ (піднесення до степеня — лівий операнд підноситься до степеня правого операнда, наприклад, $2 ** 3 = 2 * 2 * 2 = 8$), $//$ (ділення цілих чисел — повертає число, отримане в результаті ділення, але округлене до найближчого цілого числа, наприклад, $3 // 2.0 = 1.0$).
4. **Унарний** оператор — це оператор лише з одним операндом, наприклад, -1 або $+3$.
5. **Бінарний** оператор — це оператор з двома операндами, наприклад, $4+5$ або $12\% 5$.
6. Деякі оператори виконуються раніше за інших — **ієрархія пріоритетів**:
 - в унарних $+$ та $-$ найвищий пріоритет;
 - потім йде $**$, потім $*$, $/$, та $\%$, а вже потім найнижчий пріоритет: двійковий $+$ та $-$.
7. Підвираз в **дужках** завжди обчислюються першими, наприклад, $15 - 1 * (5 * (1 + 2)) = 0$.
8. Оператор **піднесення до степеня** використовує **правостороннє зв'язування**, наприклад, $2 ** 2 ** 3 = 256$.

7. Змінні — поля у формі даних

7.1. Що таке змінні?

Здається досить очевидним, що Python має дозволяти кодування літералів, які містять і цифри, і текстові значення. Ви вже знаєте, що з цими числами можна виконувати деякі арифметичні операції: додавання, віднімання і т. д. Ви будемо використовувати їх неодноразово.

Абсолютно логічним буде питання, як зберігати результати цих операцій так, щоб використовувати їх в інших операціях і так далі. Як зберегти проміжні результати і використовувати їх знову для отримання наступних?

Python впорається і з цим. Він пропонує спеціальні «контейнери», які називаються «змінними» — сама назва припускає, що вміст цих контейнерів може бути змінений (майже) будь-яким способом.

Що є в кожній змінній Python?

- Ім'я;
- значення (вміст контейнера).

Почнімо з питань, пов'язаних з ім'ям змінної. Змінні не з'являються в програмі автоматично. Як розробник, ви повинні вирішити, скільки та які змінні використовувати у ваших програмах. І ви маєте назвати їх. Якщо хочете назвати змінну, ви повинні дотримуватися деяких суворих правил:

- ім'я змінної має складатися з великих або малих літер, цифр та символу підкреслення `_`;

- ім'я змінної має починатися з літери;
- символ підкреслення — це літера;
- великі й малі літери сприймаються по-різному (не так, як у реальному світі — Аліса та АЛІСА — одне й те саме ім'я, але в Python це два різних імені змінних і, отже, дві різні змінні);
- ім'я змінної не повинно збігатися із зарезервованими словами в Python (ключові слова, про які ми скоро розповімо докладніше).



Рисунок 26

7.2. Правильні та неправильні імена змінних

Зверніть увагу, що такі ж обмеження стосуються й назв функцій. Python не накладає обмежень на довжину імен змінних, але це означає, що довге ім'я змінної завжди краще короткого.

Ось кілька прикладів правильних, але незручних імен змінних: `MyVariable`, `i`, `t34`, `Exchange_Rate`, `TheNameIsSoLongThatYouWillMakeMistakesWithIt`, `counter`, `days_to_christmas`, `_`.

Крім того, Python дозволяє використовувати не тільки латинські літери, а й символи, які належать до інших алфавітів. Ці імена змінних також правильні: `Adiós_Señora`, `sûr_la_mer`, `Einbahnstraße`, змінна.

А тепер наведемо приклади неправильних імен: `10t` (починається не з літери), `Exchange Rate` (є пробіл).

ПРИМІТКА. *PEP 8 — Посібник з написання коду на Python рекомендує таку угоду про іменування змінних та функцій у Python:*

- **імена** функцій повинні складатися з маленьких літер, а слова розділятися символами підкреслення — це необхідно, щоб їх було легко прочитати (наприклад, `var`, `my_variable`);
- **імена** функцій дотримуються тих самих правил, як імена змінних (наприклад, `fun`, `my_function`);
- **стиль** `mixedCase` (змішаний регістр, наприклад, `myVariable`) допускається у тих місцях, де вже переважає такий стиль, для збереження зворотної сумісності.

7.3. Ключові слова

Подивіться на список слів, які відіграють особливу роль у кожній програмі Python: `['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']`.

Вони називаються «ключові слова» або точніше — «зарезервовані ключові слова». Вони називаються заре-

зервованими, тому що їх не можна використовувати, як імена ні для змінних, ні для функцій або будь-яких інших іменованих сутностей, які ви створюватимете.

Значення зарезервованого слова визначено заздалегідь і не має ніяк змінюватися. Через те, що Python чутливий до регістру, ви можете змінити будь-яке з цих слів, змінивши регістр будь-якої літери, і створити таким чином нове слово, яке більше не вважається зарезервованим.

Наприклад, змінну не можна назвати ось так: `import`. Таке ім'я змінних заборонене. Але ви можете назвати її так: `Import`. Можливо, ви не розумієте значення цих слів зараз, але ми скоро в усьому розберемося.

7.4. Створення змінних

Що можна помістити у змінну? Що завгодно. Змінна може зберігати будь-яке значення будь-якого з представлених типів даних, і багатьох інших, які ми ще не розбирали.

Значення змінної — це те, що ви помістили в неї. Воно може змінюватися так часто, як вам потрібно або як ви того хочете. Спочатку вона може бути цілим числом, а через секунду — числом з рухомою крапкою, що в результаті стає рядком.

Давайте зараз поговоримо про дві важливі речі — як створюються змінні і як помістити в них значення (точніше, як до них передати значення).

ЗАПАМ'ЯТАЙТЕ. *Змінна виникає внаслідок присвоєння їй значення. На відміну від інших мов програмування, вам не потрібно про це якось оголошувати по-особливому.*

Якщо ви надаєте якесь значення неіснуючій змінній, змінна створюється автоматично. Більше вам нічого не треба робити.

Створення (або синтаксис) гранично просте: використовуйте ім'я потрібної змінної, потім знак «дорівнює» (=) і значення, яке ви хочете помістити у змінну.

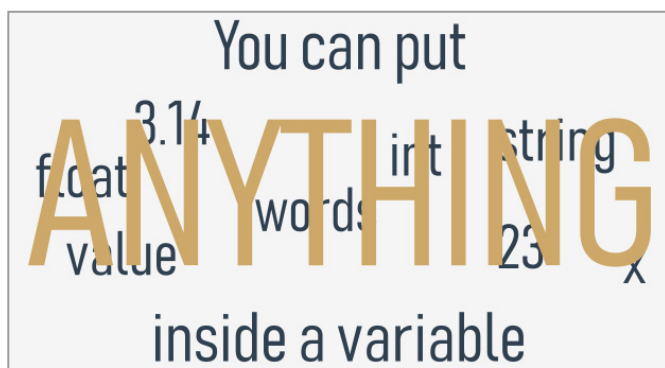


Рисунок 27

Погляньте на цей фрагмент коду:

```
var = 1  
print(var)
```

Він складається з двох простих інструкцій:

- перша створює змінну з ім'ям **var** і призначає літерал з цілим значенням, яке дорівнює **1**;
- друга виводить значення створеної змінної у консоль.

ПРИМІТКА: у функції `print()` є ще одна особливість — вона може обробляти і змінні. Ви знаєте, яке буде виведення цього фрагмента коду?

Відповідь: **1**.

7.5. Використання змінних

Можна використовувати стільки оголошень змінних, скільки вам потрібно для досягнення мети, наприклад:

```
var = 1
account_balance = 1000.0
client_name = 'John Doe'
print(var, account_balance, client_name)
print(var)
```

Не можна використовувати змінну, якої не існує (іншими словами, змінну, якій не було присвоєно значення).

Код у цьому прикладі **закінчуватиметься з помилкою**:

```
var = 1
print(Var)
```

Ми намагалися використати змінну з ім'ям **Var**, якій не надано значення (примітка: **var** і **Var** — різні сутності, і не мають нічого спільного з погляду Python).

ЗАПАМ'ЯТАЙТЕ. Можна використовувати вираз `print()` та об'єднати текст та змінні, використовуючи оператор `+` для виведення рядків та змінних, наприклад:

```
var = "3.7.1"
print("Python version: " + var)
```

Ви можете передбачити виведення фрагмента коду вище?

Відповідь: **Python version: 3.7.1.**

7.6. Присвоєння нового значення вже існуючої змінної

Як привласнити нове значення вже створеній змінній? Так само. Просто потрібно використати знак «дорівнює» («=»).

Знак «дорівнює» («=») — це, насправді, оператор присвоєння. Хоча це й може здатися дивним, але цей оператор має простий синтаксис і однозначну інтерпретацію. Він присвоює значення свого правого аргументу лівому, тоді як правий аргумент може бути довільно складним виразом, що включає літерали, оператори і певні змінні.

Подивіться на код нижче:

```
var = 1 print(var)
var = var + 1
print(var)
```

Код виводить два рядки в консолі:

```
1
2
```

Перший рядок фрагмента коду створює нову змінну, яка називається **var** і надає їй значення **1**. Вираз означає наступне: надати значення **1** змінній з ім'ям **var**. Можна сказати коротше: надати **var 1**. Хтось полюбить читати таке твердження наступним чином: **var** стає **1**.

Третій рядок надає ту ж змінну з новим значенням, яке береться з самої змінної, і додається до **1**. Якби математик побачив такий запис, він, напевно, заперечив би — жодне значення не може дорівнювати самому собі «плюс» одиниця. Це суперечність. Але Python сприймає

знак `=` не як «дорівнює», а як «присвоїти значення». Отже, як ви прочитаєте такий запис у програмі?

Прийняти поточне значення змінної `var`, додати до неї `1` і зберегти результат у змінній `var`. По суті, значення змінної `var` збільшили на одиницю, яка не має нічого спільного в порівнянні змінної з будь-яким значенням.

Чи знаєте ви, яким буде виведення наступного фрагмента коду?

```
var = 100
var = 200 + 300
print(var)
```

Відповідь: `500` — чому? По-перше, змінна `var` створюється і їй надається значення `100`. Потім до тієї ж змінної надається нове значення: результатом додавання `200` і `300`, який дорівнює `500`.

7.7. Вирішення простих математичних завдань

Зараз ви можете створити коротку програму, яка обчислює прості математичні завдання, наприклад, теорему Піфагора: *Квадрат гіпотенузи дорівнює сумі квадратів двох інших сторін*.

Наступний код обчислює довжину гіпотенузи (тобто найдовшу сторону прямокутного трикутника, протилежну прямому куту), використовуючи теорему Піфагора:

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5
print("c =", c)
```

ПРИМІТКА: нам потрібно скористатися оператором `**`, щоб знайти квадратний корінь у такий спосіб: $\sqrt{x} = x^{(1/2)}$, і в такий: $c = \sqrt{a^2 + b^2}$.

Ви можете передбачити виведення коду? Відповідь знаходиться нижче, але спочатку запустіть код редактора, щоб підтвердити свої прогнози.

Відповідь: `c = 5.0`.

7.8. Скорочені форми запису

Настав час для наступного набору операторів, які допомагають розробнику. Дуже часто нам потрібно використовувати ту саму змінну як справа, так і зліва від оператора `=`. Наприклад, якщо нам потрібно обчислити серію послідовних значень степенів двійки, можна використати такий фрагмент: `x = x * 2`.

Ви можете використати подібний вираз, якщо не можете заснути і намагаєтеся впоратися з безсонням, рахуючи овець: `sheep = sheep + 1`.

Python пропонує вам скорочену форму запису таких операцій, які можна кодувати так:

```
x *= 2
sheep += 1
```

Спробуємо уявити загальний опис цих операцій.

Якщо `оп` це оператор з двома аргументами (це дуже важлива умова), і оператор використовується у наступному контексті:

```
змінна = змінна оп вираз
```


Його можна спростити і записати так:

```
змінна оп = вираз
```

Подивіться на приклади нижче. Переконайтеся, що ви розумієте їх усі.

```
i = i + 2 * j => i += 2 * j
var = var / 2 => var /= 2
rem = rem % 10 => rem %= 10
j = j - (i + var + rem) => j -= (i + var + rem)
x = x ** 2 => x **= 2
```

7.9. Ключові висновки

1. Змінна є іменованим місцем, зарезервованим для зберігання значень у пам'яті. Змінна створюється або ініціалізується автоматично, коли ви надаєте їй значення вперше.
2. У кожної змінної має бути унікальне ім'я — ідентифікатор. Ім'я допустимого ідентифікатора має бути непустою послідовністю символів, воно має починатися з нижнього підкреслення (`_`) або літери, і воно не може бути ключовим словом, зарезервованим у Python. За першим символом можуть йти підкреслення, літери та цифри. Ідентифікатори в Python чутливі до регістру.
3. Python є динамічно типізованою мовою програмування, а це означає, що вам не потрібно оголошувати змінні. Щоб надати значення змінним, можна використовувати простий оператор присвоєння у вигляді знака «дорівнює» (`=`), тобто `var = 1`.

4. Також можна використовувати складові оператори присвоєння (оператори зі скороченою формою запису) для зміни значень, призначених змінним, наприклад: `var += 1` або `var /= 5 * 2`.
5. Можна надати нові значення вже існуючим змінним, використовуючи оператор присвоєння або один із складових операторів, наприклад:

```
var = 2
print(var)

var = 3
print(var)

var += 1
print(var)
```

6. Можна комбінувати текст та змінні, використовуючи оператор `+` та функцію `print()` для виведення рядків та змінних, наприклад:

```
var = "007"
print("Agent " + var)
```

8. Коментар до коментарів

8.1. Коментарі в кодї: навіщо, як і коли

Швидше за все, ви захочете додати кілька слів, щоб пояснити іншим, хто буде читати ваш код, як працюють прийоми, використані в кодї, або значення змінних, і, зрештою, щоб зберегти інформацію про те, хто є автором коду і коли програма була написана.

Коментар — це замітка, внесена в програму, яка ігнорується під час виконання.

Як залишити такий коментар у початковому кодї? Це потрібно зробити так, щоб Python не інтерпретував його як частину коду. Коли Python зустрічає коментар у програмі, він повністю невидимий для нього — з погляду Python, це лише один пробіл (незалежно від того, наскільки довгим буде справжній коментар).

Коментар у Python — це фрагмент тексту, який починається з `#` (знак решітки) і закінчується в кінці рядка. Якщо вам потрібен коментар, який займає декілька рядків, то кожен закоментований рядок потрібно починати зі знаку решітки.

Як тут:

```
# Ця програма вираховує гіпотенузу c.  
# a та b — довжина сторін.  
a = 3.0 b = 4.0 c = (a ** 2 + b ** 2) ** 0.5  
  
# Ми використовуємо ** замість квадратного кореня.  
print("c =", c)
```

Хороші, відповідальні розробники описують кожну важливу частину коду, наприклад, щоб пояснити роль змінних; хоча треба сказати, що найкращий спосіб коментувати змінні — назвати їх абсолютно однозначно і зрозуміло.

Наприклад, якщо певна змінна призначена для зберігання області деякого унікального квадрата, тоді ім'я «`squareArea`» вочевидь буде краще, ніж «`auntJane`». Зазвичай кажуть, що ім'я самокоментується.

Коментарі можуть бути корисні в іншому відношенні — ви можете використовувати їх для позначення фрагмента коду, який на даний момент не потрібен з якоїсь причини. Подивіться на приклад нижче, якщо ви розкоментуєте виділений рядок, це вплине на виведення коду:

```
# Це тестова програма.  
x = 1  
y = 2  
  
# y = y + x  
print (x + y)
```

Такий прийом часто використовують під час тестування програми, щоб ізолювати те місце, де може приховуватися помилка.

ПОРАДА. Якщо ви хочете швидко закоментувати або розкоментувати кілька рядків коду, виділіть рядки, які ви хочете змінити, і скористайтеся наступною комбінацією клавіш: `CTRL + /` (Windows) або `CMD + /` (Mac OS). Це дуже корисний прийом.

8.2. Ключові висновки

1. Коментарі можуть надавати додаткову інформацію у коді. Вони ігноруються під час виконання. Інформація, надана у початковому коді, адресована читачам. Коментар у Python — це фрагмент тексту, який починається з `#`. Коментар закінчується наприкінці рядка.
2. Якщо вам потрібен коментар, який займає декілька рядків, то кожен закоментований рядок потрібно починати зі знаку решітки (`#`). Крім того, за допомогою коментаря можна помітити фрагмент коду, який зараз не потрібен (див. останній рядок у фрагменті коду нижче), наприклад:

```
# Ця програма виводить  
# на екран вітання.  
print("Hello!") # Викликає функцію print()  
# print("I'm Python.")
```

3. Коли це можливо та виправдано, давайте самокоментуючі імена змінним, наприклад, якщо ви використовуєте дві змінні для зберігання довжини та ширини чогось, імена змінних `length` та `width` кращі за `MyVar1` та `myvar2`.
4. Важливо використовувати коментарі, щоб полегшити розуміння програм, а також використовувати прийнятні та доречні імена змінних в коді. Однак не менш важливо не використовувати імена змінних, які спантеличують. Не залишайте коментарі, які містять неправильну або перекручену інформацію!

5. Коментарі можуть бути корисними, коли ви згодом читаєте власний код (повірте, розробники забувають, що робить їх власний код), і коли інші читають ваш код (коментарі допомагають зрозуміти іншим, що роблять ваші програми і як це можна зробити швидше).

9. Як спілкуватися з комп'ютером

9.1. Функція введення `input()`

А зараз ми познайомимось із абсолютно новою функцією, яка на перший погляд здається дзеркальним відображенням функції `print()`. Чому? Тому що `print()` надсилає дані в консоль. А нова функція отримує дані із неї. У `print()` немає корисного результату. Сенс нової функції полягає в тому, щоб повернути корисний результат.

Функція називається `input()` (введення). Назва функції говорить сама за себе. Функція `input()` зчитує введені користувачем дані та повертає ті ж дані у працюючу програму.

Програма може маніпулювати даними, додаючи інтерактивність у код. Практично будь-яка програма читає та обробляє дані. Програма, яка не отримує введення від користувача, це «глуха» програма. Подивіться на приклад нижче:

```
print("Tell me anything...")
anything = input()
print("Hmm...", anything, "... Really?")
```

Він демонструє дуже простий випадок використання функції `input()`.

ПРИМІТКА. Програма пропонує ввести деякі дані з консолі (швидше за все, з клавіатури, хоча це може бути і голосове введення або зображення).

Функція `input()` викликається без аргументів (це найпростіший спосіб використання функції); функція перемикає консоль в режим введення; ви побачите блимаючий курсор і зможете ввести кілька символів, натискати клавіші, і закінчити, натиснувши клавішу `Enter`; всі введені дані будуть відправлені у вашу програму через результат функції (вам потрібно присвоїти результат змінної; це дуже важливо — якщо пропустите цей крок, це призведе до втрати введених даних).

Далі ми використовуємо функцію `print()` для виведення отриманих даних, але з деякими застереженнями.

Спробуйте запустити код і погляньте, на що здатна ця функція.

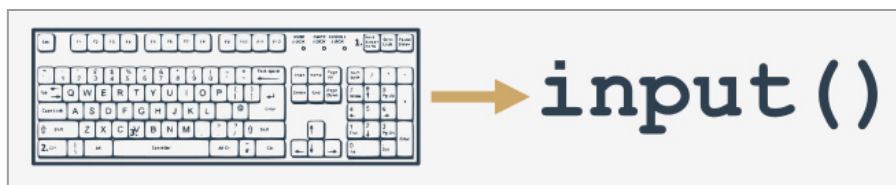


Рисунок 28

9.2. Функція `input()` з аргументом

Функція `input()` може ще дещо — давати підказки користувачу без використання `print()`.

Ми трохи змінили наш приклад, подивіться на код:


```
anything = input("Tell me anything...")
print("Hmm...", anything, "...Really?")
```

ПРИМІТКА:

- функція `input()` викликається з одним аргументом — це рядок, що містить повідомлення;
- повідомлення з'явиться в консолі до того, як користувач зможе щось ввести;
- і тоді `input()` просто виконає свою роботу.

Цей варіант виклику функції `input()` полегшує код і робить його більш зрозумілим.

9.3. Результат функції `input()`

Ми вже про це говорили, але варто ще раз повторити: результат функції `input()` — це рядок. Рядок, який містить усі символи, введені користувачем з клавіатури. Це не ціле число і не число з рухомою крапкою. Це означає, що її не потрібно використовувати як аргумент для арифметичних операцій. Наприклад, ви не зможете використовувати ці дані, щоб піднести їх у квадрат, поділити на якесь значення або поділити якесь значення на них.

```
anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

9.4. Функція `input()` — заборонені операції

Подивіться код нижче. Запустіть його, введіть будь-яке число та натисніть клавішу `Enter`.

```
# [Testing TypeError message

anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

Що відбувається? У вас мав відбутися такий вивід:

Traceback (most recent call last):

File «.main.py», line 4, in <module>

*something = anything ** 2.0*

*TypeError: unsupported operand type(s) for ** or pow():
'str' and 'float'*

Останній рядок все пояснює — ви спробували застосувати оператор ****** для **'str'** (рядка) з **'float'**. Це заборонено.

Але це очевидно — ви самі можете передбачити значення рядка «бути чи не бути», піднесеного до *другого* степеня? Не можете. От і Python також не може. Це глухий кут? Чи є вирішення цієї проблеми? Звісно є.

9.5. Перетворення типів

Python пропонує дві прості функції для вказування типу даних та вирішення цієї проблеми. Ось вони: **int()** та **float()**.

Їхні імена зрозумілі без додаткових коментарів:

- функція **int()** приймає один аргумент (наприклад, рядок **int(string)**) і намагається перетворити його на ціле число; якщо їй це не вдається, вся програма теж не спрацює (для цієї ситуації є обхідний шлях, але ми розберемо його трохи згодом);

- функція `float()` приймає один аргумент (наприклад, рядок `float(string)`) і намагається перетворити його на число з рухомою крапкою.

Це дуже просто та дуже ефективно. Більше того, можна викликати будь-яку з функцій, передавши результати `input()` безпосередньо до цих функцій. Тут немає необхідності використовувати змінні як проміжне сховище.

Ми реалізували цю ідею в редакторі, подивіться на код.

```
anything = float(input("Enter a number: "))
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

У вас є припущення про те, як рядок, введений користувачем, переходить з `input()` до `print()`?

Спробуйте запустити змінений код. Не забудьте ввести коректне число. Перевірте декілька різних значень, маленьких та великих, від'ємних та додатних. Нуль — теж непоганий варіант введення.

9.6. Більше про `input()` та перетворення типів

Така послідовність з функцій, як `input()-int()-float()` відкриває багато нових можливостей. У результаті ви зможете писати закінчені програми, приймати дані у вигляді чисел, обробляти їх та виводити результати.

Звичайно, ці програми будуть дуже примітивними і не дуже зручними, оскільки вони не зможуть приймати рішення, а отже, будуть не здатні по-різному реагувати на різні ситуації.

Хоча насправді це не проблема; ми скоро розглянемо способи подолання таких ситуацій.

У наступному прикладі ми використаємо раніше написану програму, яка знаходить довжину гіпотенузи. Давайте перепишемо код і зробимо так, щоб він міг зчитувати довжину сторін із консолі.

Подивіться на код, наведений нижче. Отак він виглядає зараз.

```
# [Testing TypeError message
anything = input("Enter a number: ")
something = anything ** 2.0
print(anything, "to the power of 2 is", something)
```

```
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
hypo = (leg_a**2 + leg_b**2) **.5
print("Hypotenuse length is", hypo)
```

Програма двічі просить користувача ввести довжину для обох сторін, вираховує гіпотенузу та виводить результат.

Запустіть код та введіть від'ємні значення. На жаль, програма не реагує на очевидні помилки. Поки що ми ігноруватимемо цей недолік, але скоро повернемося до нього.

Зверніть увагу, що змінна `hypo` використовується у цьому коді лише з однією метою — зберегти знайдене значення між виконанням сусідніх рядків коду.

Ви зможете **видалити цю змінну** з коду після того, як функція `print()` прийме вираз як аргумент.

Ось так:

```
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
print("Hypotenuse length is", (leg_a**2 + leg_b**2)
      ** .5)
```

9.7. Рядкові оператори — введення

Настав час повернутися до цих арифметичних операторів: `+` та `*`. Наша мета — показати вам, як їх ще можна використовувати, адже вони виконують не тільки функції додавання і множення. Ми знаємо, як вони працюють з аргументами у вигляді чисел (цілих або з рухомою крапкою).

А тепер ми розглянемо ситуації, коли вони обробляють рядки, хоча цей процес дуже специфічний.

9.8. Конкатенація (concatenation)

Якщо застосувати знак `+` (плюс) до двох рядків, він перетворюється на оператор конкатенації:

```
рядок + рядок
```

Він просто конкатенує (об'єднує, «склеює») два рядки в один. Звичайно, цей оператор можна використовувати більше одного разу в одному виразі і в цьому випадку він використовує лівостороннє зв'язування.

У програмуванні, на відміну від математики, оператор конкатенації не є перестановним, тобто вираз `"ab" + "ba"` — це не те саме, що `"ba" + "ab"`.

Не забувайте, що якщо знак **+** вам потрібен для конкатенації, а не для додавання, то обидва аргументи мають бути рядками. Тут не можна змішувати різні типи даних.

Ця проста програма демонструє можливості знака **+** у другому варіанті його використання:

```
fnam = input("May I have your first name, please? ")
lnam = input("May I have your last name, please? ")
print("Thank you.")
print("\nYour name is " + fnam + " " + lnam + ".")
```

ПРИМІТКА: використання **+** для об'єднання рядків дозволяє вам точніше вибудовувати виведені дані, ніж за допомогою «чистої» функції `print()`, навіть якщо в неї є ключові аргументи `end=` та `sep=`.

Запустіть код та подивіться, чи відповідає виведення вашим передбаченням.

9.9. Повторення рядка (replication)

Якщо знак ***** (зірочка) застосувати до рядка і числа (або числа і рядка, оскільки зміна місць у цьому випадку не впливає на результат), то він стане **оператором повторення рядка**:

```
рядок * число
число * рядок
```

Він повторює рядок стільки разів, скільки вказано у числі.

Наприклад, результатом цього коду:

"James" * 3 буде «JamesJamesJames».

3 * "an" буде «ananan»

5 * "2" (or "2" * 5) буде «22222» (не 10!)

Запам'ятайте. Якщо число менше або дорівнює нулю, результатом виведення буде порожній рядок.

Ця проста програма «рисуює» прямокутник, використовуючи оператор (+) у новій ролі:

```
print("+" + 10 * "-" + "+")
print(("|" + " " * 10 + "|\\n") * 5, end="")
print("+" + 10 * "-" + "+")
```

Зверніть увагу, як ми використовували круглі дужки у другому рядку коду.

Потренуйтеся створювати інші фігури або навіть витвори мистецтва!

9.10. Перетворення типів: str()

Ви вже знаєте, як використовувати функції `int()` та `float()` для перетворення рядка на число.

Цей тип перетворення працює в обидві сторони. Можна перетворити число на рядок, а це набагато простіше та безпечніше, ніж у зворотному перетворенні.

Така функція називається `str()`:

```
str(число)
```

Насправді це не межа її можливостей, але ми повернемося до цього пізніше.

9.11. Повертаємось до прямокутного трикутника

Повторимо код програми, яка вираховує гіпотенузу:

```
leg_a = float(input("Input first leg length: "))
leg_b = float(input("Input second leg length: "))
print("Hypotenuse length is " +
      str((leg_a**2 + leg_b**2) **.5))
```

Ми трохи змінили його, щоб показати вам, як працює функція `str()`. Завдяки їй ми зможемо передати весь результат функції `print()`, як один рядок, і забути про коми.

Ви досягли успіхів на шляху до програмування на Python. Ви вже знаєте основні типи даних і набір фундаментальних операторів, як організувати виведення і як отримати дані від користувача.

9.12. Ключові висновки

1. Функція `print()` надсилає дані у консоль, а функція `input()` отримує дані з консолі.
2. У функції `input()` є необов'язковий параметр: рядок підказки (`prompt`). Вона дозволяє вам вивести повідомлення до користувацького вводу, наприклад:

```
name = input("Enter your name: ")
print("Hello, " + name + ". Nice to meet you!")
```

3. Коли відбувається виклик функції `input()`, потік програми зупиняється, курсор блимає (спонукаючи користувача до дії, коли консоль перемикається в режим введення) доти, доки користувач не введе якісь дані та/або не натисне клавішу `Enter`.

ПРИМІТКА. Ви можете перевірити на своєму комп'ютері, як працює функція `input()` у всіх можливих варіантах. Відкрийте IDLE, скопіюйте та вставте наведений вище фрагмент коду, запустіть програму і нічого не робіть — просто зачекайте кілька секунд, щоб побачити, що відбудеться.

ПОРАДА. Вищезазначену особливість функції `input()` можна використати, щоб запропонувати користувачу завершити програму. Подивіться на код нижче:

```
name = input("Enter your name: ")
print("Hello, " + name + ". Nice to meet you!")
print("\nPress Enter to end the program.")
input()
print("THE END.")
```

4. Результатом виконання функції `input()` буде рядок. Можна з'єднувати рядки один з одним, використовуючи оператор конкатенації (+). Подивіться на цей код:

```
num1 = input("Enter the first number: ") # Введіть 12
num2 = input("Enter the second number: ") # Введіть 21
print(num1 + num2) # програма поверне 1221
```

5. А ще рядки можна помножити (* — повторення рядка), наприклад:

```
myInput = ("Enter something: ")
# Приклад введення: hello
print(myInput * 3)
# Очікуване виведення: hellohellohello
```



Урок 1

Вступ у web- програмування на Python

© STEP IT Academy, www.itstep.org

© По матеріалам Cisco

Усі права на фото-, аудіо- і відеотвори, що охороняються авторським правом і фрагменти яких використані в матеріалі, належать їх законним власникам. Фрагменти творів використовуються в ілюстративних цілях в обсязі, виправданому поставленим завданням, у рамках учбового процесу і в учбових цілях, відповідно до законодавства про вільне використання твору без згоди його автора (або іншої особи, яка має авторське право на цей твір). Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає збитку нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора і правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними аналогами, що не охороняються авторським правом, і відповідають критеріям добросовісного використання і чесного використання.

Усі права захищені. Повне або часткове копіювання матеріалів заборонене. Узгодження використання творів або їх фрагментів здійснюється з авторами і правовласниками. Погоджене використання матеріалів можливе тільки якщо вказано джерело.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством.