Функції (частина 1)

№ уроку: 8 **Курс:** Python Starter

Засоби навчання: PyCharm

Огляд, мета та призначення уроку

Після завершення уроку учні матимуть уявлення про функції, у чому відмінність функції від процедури, як створити користувацьку функцію, її використовувати, види параметрів, а також функції з невідомою кількістю параметрів.

Вивчивши матеріал даного заняття, учень зможе:

- Розуміти, що таке функції та користуватися ними
- Створювати власні функції
- Користуватися іменованими аргументами під час виклику функцій, задавати аргументи у довільному порядку
- Користуватися опціональними параметрами

Зміст уроку

1. Поняття функції
2. Види параметрів
3. Іменовані параметри під час виклику функції
4. Значення аргументів за замовчуванням (опціональні параметри)
5. Функції від невідомої кількості аргументів

Резюме

Функція в програмуванні — іменований фрагмент програмного коду (підпрограма), до якого можна звернутися з іншого місця програми.
Функції потрібні головним чином, щоб уникнути повторення однакового коду.
Функція може приймати параметри і повинна повертати певне значення, можливо, пусте. Функції, які повертають пусте значення, часто називають процедурами.
У деяких мовах програмування оголошення функцій та процедур мають різні синтаксиси, зокрема, можуть використовуватися різні ключові слова. У Python на рівні мови немає різниці між процедурами та функціями.
Процедури Python завжди повертають значення None. Якщо функція сама не повернула значення, значення None повертається автоматично.
Параметр у програмуванні - прийнятий функцією аргумент. Термін «аргумент» передбачає, що і якій конкретної функції було передано, а параметр — як функція застосувала це прийняте. Тобто код, що викликає, передає аргумент у параметр, який визначений у члені специфікації функції.
Формальний параметр — аргумент, що вказується під час оголошення або визначення функції.
Фактичний параметр - аргумент, що передається у функцію під час її виклику.
При виклику функції формальні параметри заміняються фактичними.
Семантика використання формальних та фактичних параметрів називається стратегією обчислення. Задана стратегія обчислення диктує, коли слід обчислювати аргументи функції (методу, операції, відносини) і які значення слід передавати. Існує чимало різноманітних стратегій обчислення.
Дуже часто поняття стратегії обчислення також називають способом передачі параметрів, хоча для деяких стратегій обчислення (наприклад, виклик за потребою) такий термін некоректний.
Виклик за значенням (англ. call-by-value) є найбільш широко поширеною стратегією обчислень, її можна бачити в різних мовах, від Сі до Scheme. При виклику за значенням, вираз-аргумент обчислюється, і отримане значення пов'язується з відповідним формальним параметром

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

функції (зазвичай шляхом копіювання цього значення в нову область пам'яті). При цьому, якщо мова дозволяє функціям надавати значення своїм параметрам, зміни будуть стосуватися лише цих локальних копій, але видимі в місці виклику функції значення залишаться незмінними після повернення.
При виклику за посиланням (англ. call-by-reference) або передачі за посиланням (pass-by-reference), функція неявно отримує посилання на змінну, використану як аргумент, замість копії її значення. Зазвичай це означає, що функція може здійснювати модифікацію (тобто змінювати стан) змінної, переданої як параметр, і це матиме ефект у викликаю чому контексті.
Виклик з використанням або виклик з поділом ресурсів (англ. call-by-sharing), також виклик по об'єкту (call-by-object), також виклик з використанням-об'єкта або виклик з об'єктом (call-by-object-sharing) , має на увазі, що значення в мові засновані на об'єктах, а не на примітивних типах, тобто на тому, що всі значення обернуті (boxed). При виклику за використанням функція отримує значення, що містить копію посилання на об'єкт. Сам об'єкт не копіюється - він виявляється розділюваним, або використовуваним спільно. Як наслідок, привласнення аргументу в тілі функції не має ефекту в контексті, що викликає її, але привласнення компонентам цього аргументу — має.
Виклик за використанням використовується в мовах Python, Iota, Java (для посилань на об'єкти), Ruby, Scheme, OCaml, AppleScript та багатьох інших. Проте термінологія у спільнотах різних мов різниться. Наприклад, у спільноті Python використовується термін «виклик зі співкористування»; у спільнотах Java і Visual Basic ту ж семантику часто описують як «виклик за значенням, де значенням є посилання на об'єкт»; у спільноті Ruby кажуть, що Ruby «використовує виклик за посиланням» - незважаючи на те, що семантика виклику в цих мовах ідентична.
На практиці це означає, що при передачі параметрів у функції Python ми не можемо прив'язати фактичні параметри до інших об'єктів або змінити їх, якщо вони відносяться до незмінних типів (див. урок 2), однак можемо модифікувати їх значення, якщо вони відносяться до змінних типів, таких як списки, що були розглянуті на минулому уроці.
Не хвилюйтеся, якщо ви не до кінця зрозуміли ту частину теорії, яка стосується стратегій обчислення, оскільки вона наведена лише для ознайомлення і буде детально розібрана в курсі Python Essential.
Кожній функції слід виконувати лише одну логічно завершену дію. Не потрібно писати надто великі функції, краще розбивати код на невеликі незалежні частини, які легко написати, прочитати та протестувати на коректність.

Оголошення функції в Python:

```
def ім’я_функції(аргумент_1, аргумент_2, аргумент_n):
    оператори
```

Якщо функція не приймає жодних значень, порожній список аргументів все одно виділяється парою круглих дужок.
Для повернення значення функції використовується оператор return. Його також можна використовувати для дострокового завершення роботи функції.
Оператор return без параметрів повертає None.

Виклик функції:

```
ім'я_функції(аргумент_1, аргумент_2, аргумент_n)
```

При виклику функції фактичні параметри заміщають формальні у порядку, в якому зазначені. Можна змінити цей порядок, вказавши під час виклику імена відповідних формальних параметрів: ім'я_функції(аргумент_2=значення, аргумент_1=значення)
При виклику функції можна використовувати іменовані та неіменовані аргументи одночасно. У такому випадку спочатку вказуються неіменовані фактичні параметри, які заміщають формальні в тому порядку, в якому в заголовку функції описано відповідну кількість формальних параметрів, а потім зазначаються інші фактичні параметри разом з іменами відповідних формальних параметрів у довільному порядку.
Параметри функцій можна робити опціональними, тобто необов'язковими шляхом задання ним значень за замовчуванням:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

def ім'я_функції(аргумент_1, аргумент_2=значення):
оператори

У цьому разі, якщо значення відповідного параметра не задано під час виклику функції, буде використано стандартне значення.

В Python також є можливість створювати функції від невідомої кількості аргументів, але її використання вимагає знання та вміння працювати зі словниками та списками.

У програмуванні, якщо нам потрібно виконувати схожі дії, ми визначаємо функції багаторазового використання коду. Щоб виконати цю дію, ми викликаємо функцію з певним значенням — аргументом.

Припустимо, у нас є функція, яка складає три числа:

```
def adder(x, y, z):
   print("sum:", x + y + z)

adder(10, 12, 13)
```

Після запуску буде виведено sum: 35.

У фрагменті коду вище ми маємо функцію adder() з трьома аргументами: x, y і z. При передачі трьох значень цій функції на виході ми отримуємо їхню суму. Але що, якщо передати більше трьох аргументів на цю функцію?

```
def adder(x, y, z):
   print("sum: ",x + y + z)

adder(5, 10, 15, 20, 25)
```

Через те, що тут ми передаємо 5 аргументів, при запуску програми виводиться помилка TypeError: adder() 3 positional arguments but 5 were given.

У Python можна передати змінну кількість аргументів двома способами:

***args** для неіменовах аргументів;
****kwargs** для іменованих аргументів.

Ми використовуємо *args і **kwargs як аргумент, коли невідомо, скільки значень ми хочемо передати функції.

***args** потрібен, коли потрібно передати невідому кількість неіменованих аргументів. Якщо поставити * перед іменем, це ім'я буде приймати не один аргумент, а кілька. Аргументи передаються як кортеж і доступні всередині функції під тим самим ім'ям, що й ім'я параметра тільки без *. Наприклад:

```
def adder(*nums):
   sum = 0

   for n in nums:
      sum += n

   print("Sum: ", sum)

adder(3, 5)
adder(4, 5, 6, 7)
adder(1, 2, 3, 5, 6)
```

В результаті виконання програми ми отримаємо наступний результат:

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

```
Sum: 8
Sum: 22
Sum: 17
```

Тут ми використовували *nums як параметр, який дозволяє передавати змінну кількість аргументів на функцію adder(). Усередині функції ми проходимося в циклі за цими аргументами, щоб знайти їхню суму, і виводимо результат.

За аналогією з *args ми використовуємо **kwargs для передачі змінної кількості іменованих аргументів. Схоже з *args, якщо поставити ** перед ім'ям, це ім'я прийматиме будь-яку кількість іменованих аргументів. Словник із кількох переданих аргументів буде доступний під цим ім'ям. Наприклад:

```
def intro(**data):
    print("\nData type of argument: ", type(data))

    for key, value in data.items():
        print("{} is {}".format(key, value))

intro(Firstname="Sita", Lastname="Sharma", Age=22, Phone=1234567890)
intro(Firstname="John", Lastname="Wood", Email="johnwood@nomail.com", Country="Wakanda", Age=25, Phone=9876543210)
```

Під час запуску програми ми побачимо наступне:

```
Data type of argument: <class 'dict'>
Firstname is Sita
Lastname is Sharma
Age is 22
Phone is 1234567890

Data type of argument: <class 'dict'>
Firstname is John
Lastname is Wood
Email is johnwood@nomail.com
Country is Wakanda
Age is 25
Phone is 9876543210
```

***args** і ****kwargs** — спеціальний синтаксис, що дозволяє передавати змінну кількість аргументів у функцію. При цьому зовсім не обов'язково використовувати імена аргументів args і kwargs;
***args** використовується для неіменованих аргументів, з якими можна працювати як зі списком;
****kwargs** використовується для іменованих аргументів, з якими можна працювати як зі словником;
якщо ви хочете використовувати і *args, і **kwargs, це робиться так: func(fargs, *args, **kwargs), порядок дотримання аргументів важливий;

Закріплення матеріалу

- Що таке функція?
- Що таке процедура?
- Чи є в Python різниця між функціями та процедурами на рівні мови?
- Навіщо потрібні функції?
- Що таке формальні параметри функції?
- Що таке фактичні параметри функції?
- Як задаються функції в Python?
- Який оператор використовується для повернення значення з функції?

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- Яким чином можна задавати фактичні параметри у довільному порядку?
- Що таке опціональні параметри?
- Як в Python можна передати змінну кількість аргументів?

Додаткове завдання

Завдання 1
Створіть програму, яка складається з функції, яка приймає три числа і повертає їх середнє арифметичне, і головного циклу, що запитує у користувача числа і обчислює їх середні значення за допомогою створеної функції.

Завдання 2
Створіть програму, яка приймає як формальні параметри зріст і вагу користувача, обчислює індекс маси тіла і в залежності від результату повертає інформаційне повідомлення (маса тіла в нормі, недостатня вага або слідкуйте за фігурою). Користувач з клавіатури вводить значення росту та маси тіла та передає ці дані у вигляді фактичних параметрів під час виклику функції. Програма працює доти, доки користувач не зупинить її комбінацією символів «off».

Самостійна діяльність учня

Завдання 1
Створіть функцію, яка відображає привітання для користувача із заданим ім'ям. Якщо ім'я не вказано, вона повинна виводити привітання для користувача з Вашим ім'ям.

Завдання 2
Створіть дві функції, що обчислюють значення певних алгебраїчних виразів. На екрані виведіть таблицю значень цих функцій від -5 до 5 з кроком 0.5.

Завдання 3
Створіть програму-калькулятор, яка підтримує наступні операції: додавання, віднімання, множення, ділення, зведення в ступінь, зведення до квадратного та кубічного коренів. Всі дані повинні вводитися в циклі, доки користувач не вкаже, що хоче завершити виконання програми. Кожна операція має бути реалізована у вигляді окремої функції. Функція ділення повинна перевіряти дані на коректність та видавати повідомлення про помилку у разі спроби поділу на нуль.

Рекомендовані ресурси

Документація з Python
https://docs.python.org/3/reference/compound_stmts.html#function-definitions
https://docs.python.org/3/library/functions.html
https://docs.python.org/3/reference/simple_stmts.html#the-global-statement
https://docs.python.org/3/reference/simple_stmts.html#the-nonlocal-statement

Статті у Вікіпедії про ключові поняття, розглянуті на цьому уроці
https://ru.wikipedia.org/wiki/Функция_(программирование)
https://uk.wikipedia.org/wiki/Підпрограма
https://ru.wikipedia.org/wiki/Параметр_(программирование)
https://uk.wikipedia.org/wiki/Стратегії_обчислення
https://ru.wikipedia.org/wiki/Область_видимости
https://uk.wikipedia.org/wiki/Локальна_змінна
https://uk.wikipedia.org/wiki/Глобальна_змінна
https://uk.wikipedia.org/wiki/Рекурсія

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com