Lekcja 10. Funkcje cz. I

Cel lekcji

Celem lekcji jest wprowadzenie do tematu funkcji.

QUIZ

Uczestnicy wykonują quiz, do którego mają dostęp z poziomu **panelu ucznia** (zakładka *Dashboard* lub *Moje Kursy*).
Quiz do wglądu dla trenera:

https://quiz.giganciprogramowania.edu.pl/wdp-python1-q2

Powyższy link możemy udostępnić uczestnikom TYLKO jeśli mają problemy z zalogowaniem się do panelu ucznia. Wyniki wtedy dla tych osób musimy ręcznie zapisać w liście obecności (przycisk *Info* przy danym uczestniku).

Zapoznaj się z treścią quizu przed zajęciami.
Pamiętaj żeby po zakończeniu quizu omówić pytania i odpowiedzi.

Przebieg zajęć

1. Pytania wstępne z poprzednich zajęć
2. Wprowadzenie

3. Własna pierwsza funkcja
4. Argumenty
5. Wartości zwracane
6. Podsumowanie oraz pytania powtórzeniowe

1. **Przykładowe pytania z poprzednich zajęć**

Od jakiej wartości indeksujemy elementy w liście?

Czy z listy możemy wyciągnąć pod-listę? Więcej elementów za jednym razem?

Czy lista może zmieniać swój rozmiar w trakcie działania programu?

Za pomocą jakiej komendy dodać element do listy?

Za pomocą jakiej instrukcji odczytać rozmiar listy - liczbę elementów?

Zadanie rozgrzewkowe:

Napisz program, który pyta użytkownika o liczbę całkowitą, a następnie tworzy listę, która zawiera tyle losowych liczb.

Wygenerowana lista powinna zostać wyświetlona w konsoli.

2. **Wprowadzenie**

Na zajęciach o pętlach pokazaliśmy sobie co należy robić, aby nie musieć powtarzać tego samego kodu. Co w sytuacji, w której kod się powtarza, ale dla trochę innych danych - jak rozwiązać taką sytuację?

Przykład. Wyświetl w konsoli krótki raport pracowniczy, na którym wyświetli się imię i nazwisko oraz staż pracy w formie: “Osoba X Y pracuje w naszej firmie od Z lat”. Wykonaj to dla 5 osób (wymyśl dane).

Jakie mamy tutaj problemy:

- monotonny kod, kolejne instrukcje tylko trochę różnią się od poprzedniej,
- gdybyśmy chcieli zmienić treść komunikatu, należałoby ręcznie poprawić każdy komunikat - **bardzo dużo pracy**

Taki kod będziemy mogli opakować w funkcję, którą następnie będziemy używać w kodzie. Jakie są z tego korzyści:

- zmiana komunikatu wymaga zmiany tylko w jednym miejscu (nawet, jeśli wyświetlamy komunikat dla 100 osób),
- poprawi się czytelność kodu,

- jeżeli chcielibyśmy dodać powitanie w innej części kodu możemy skorzystać z gotowej funkcji (w której jest gotowy format powitania), a nie musimy zastanawiać się jaki jest aktualny format powitania,
- możemy wykorzystać stworzoną funkcję i skorzystać z niej w dowolnym miejscu w naszym kodzie - nie musimy duplikować kodu.

3. Własna pierwsza funkcja

Pierwszym krokiem jest poznanie sposobu zapisu funkcji. Tak jak pętle, czy instrukcje warunkowe - python musi zrozumieć co chcemy napisać.

```
def powitanie():
    print("Hej, Tomek!")
    pass
```

Słowo kluczowe **def** rozpoczyna deklarację funkcji, po tym słowie zapisujemy nazwę funkcji oraz nawiasy. Na końcu linijki należy napisać dwukropek - dokładnie tak jak z pętlą i instrukcją warunkową. W ciele funkcji - we wcięciu - zapisujemy kod, który wykona się jeśli uruchomimy naszą funkcję. Dla czytelności kodu można dodać słowo kluczowe **pass** na końcu wcięcia.

Uruchomienie powitania w programie:

```
powitanie()
```

Aby uruchomić naszą funkcję należy w kodzie programu podać jej nazwę wraz z nawiasami - to nawiasy sprawiają, że python odróżnia zmienną od funkcji.
Ważne: wywołanie funkcji musi mieć miejsce po deklaracji funkcji.

4. Argumenty

Nawiasy w zapisie funkcji przeznaczone są dla argumentów. Argumenty pozwalają przekazywać do funkcji informacje z zewnątrz. Argumentów może być wiele, ale również nie musi być ich wcale.
Edytujemy funkcję powitania tak, aby podać imię jako argument.

```
def powitanie(imie):
```

```
    print(f"Hej, {imie}!")
    pass
```

Dzięki tej edycji funkcja skorzysta z otrzymanego argumentu, który wyświetli w komunikacie. Aby wywołać tę wersję funkcji należy napisać:

powitanie("Tomek")	nick = "Tomasso" powitanie(nick)

Należy przekazać argument wewnątrz nawiasów. Argumentem może być konkretna wartość lub zmienna. Wewnątrz funkcji argument traktujemy jak zwykłą zmienną, możemy ją edytować, odczytywać, itd. Należy pamiętać jednak o tym, że istnieje ona tylko wewnątrz funkcji - jej zasięg nie wykracza poza ciało funkcji.

Funkcja może używać więcej argumentów:

```
def powitanie(imie, powtorzenia):
    for i in range(powtorzenia):
        print(f"Hej, {imie}!")
    pass

powitanie("Tomek", 0)

nick = "Tomasso"
powitanie(nick, 5)
```

Edytujemy funkcję tak, aby wyświetliła tyle powitań ile podamy wartości w drugim argumencie. Kolejność argumentów jest bardzo ważna, pomyłka będzie skutkować błędem podczas pracy programu - czyli dowiemy się o nim w ostatnim momencie.
Należy zaprezentować odwrotne użycie argumentów. Co jest powodem błędu?
Sama zamiana miejscami nie jest zła, dopiero próba użycia argumentów wewnątrz funkcji skutkuje błędem. Jeżeli argument *imie* przyjmie wartość liczbową nic złego się nie stanie - wyświetlił by się komunikat "*Hej, 0!*". Problemem jest użycie zmiennej *powtorzenia* w funkcji range, która nie może otrzymać wartości typu string.

Zadanie. Przygotuj funkcję obliczającą pole prostokąta (dodatkowo trójkąta prostokątnego). Funkcja ma przyjmować długości boków, a następnie obliczać i wyświetlać pole figury.

```python
def prostokat(a, b):
    pole = a * b
    print(f"Pole prostokata o bokach {a}x{b} = {pole}")
    pass
```

Zadanie Pole koła, pole kolejnych figur. Należy użyć modułu math

`import math`	math.pi

Wartości domyślne argumentów

Argumenty, które deklarujemy dla funkcji mogą posiadać wartości domyślne, czyli wartości przypisywane argumentom, jeśli nie podamy argumentu podczas wywołania funkcji.

```python
import time


def pasek_ladowania(gotowe, wszystko=100):
    # Znak '#' oznacza wykonaną część
    # Znak '-' oznacza niewykonaną część

    # Zmiana skali postępu z 'wszystko' na 10
    wykonane = round(10 * gotowe / wszystko)
    niewykonane = 10 - wykonane

    # Obliczenie ile znakow ma sie pojawic
    tekst_wykonane = '#' * wykonane
    tekst_niewykonane = '-' * niewykonane
    # [####------]
    print(f'\r[{tekst_wykonane}{tekst_niewykonane}] ', end=' ')
    pass


for i in range(100):
    pasek_ladowania(i, 200)
    # pasek_ladowania(i)
    time.sleep(0.1)
    pass
```

Przykładowa funkcja wypisująca na ekranie pasek ładowania, dla którego podajemy dwa argumenty: wartość gotową oraz wartość wszystkich elementów. Domyślnie wszystko przyjmuje wartość 100, ale nie ma problemu, aby wywołać funkcję dla innych parametrów, gdzie zliczamy od 1 do 200.

Należy przypomnieć za co odpowiada \r oraz end.

\r to przesunięcie karetki na początek linijki

end określa czym kończy się nasza komunikat - domyślnie \n czyli ‘enter’

Przekazywanie wartości funkcji, trzy sposoby

Argumenty muszą przyjąć jakąś nazwę, nie możemy pozostawić argumenty bez żadnej wartości. Poznaliśmy już wartość domyślną oraz przypisywanie wartości według kolejności. Istnieje jeszcze jeden sposób, zaprezentowany w funkcji print oraz argumencie *end* - przypisywanie po nazwie.

```
pasek_ladowania(gotowe=50, wszystko=200)
pasek_ladowania(wszystko=200, gotowe=50)
```

Możliwe jest połączenie podawania argumentów według kolejności oraz za pomocą nazw, ale nazwy mogą być wykorzystywane na końcu.

Nie można najpierw podać argumentu po nazwie, a potem podawać według kolejności.

```
pasek_ladowania(gotowe=50, 200)  # Źle
pasek_ladowania(50, wszystko=200)  # Dobrze
```

Zadanie Napisz funkcję, która przyjmuje dwa argumenty: n - liczba powtórzeń, a - liczba startowa. Funkcja ma generować kolejne kwadraty liczb, zaczynając od a. Ma wyświetlić n kolejnych liczb.

```
def foo(a, n):
    for i in range(a, a + n):
        print(f"{i} ** 2 = {i ** 2}")
    pass

foo(5, 4)
```

5. Wartości zwracane

Funkcja nie musi wyświetlać komunikatów w konsoli, aby była przydatna. Kolejną zaletą funkcji jest możliwość zwracania wartości - stałych lub wyliczonych na podstawie argumentów. Aby funkcja zwracała pewną wartość wystarczy dopisać do niej instrukcję **return** oraz dopisać zwracaną wartość.

Funkcja obliczająca pole sześciokąta na podstawie długości boku:

```python
from math import sqrt

def pole_szesciokota_foremnego(a):
    return 3 * sqrt(3) * a ** 2 / 2


pole = pole_szesciokota_foremnego(2)

print(pole)
```

Zadanie

Napisz funkcję tworzącą powitanie, które wykorzystuje jako argument *imie* a zwraca pełen tekst powitania.

```python
def powitanie(imie):
    return f"Hej, {imie}! :)"
```

Zadanie

Napisz funkcję obliczającą objętość graniastosłupa prawidłowego, który w podstawie ma sześciokąt (foremny, wiemy to dzięki słowu “prawidłowy”).

```python
def pole_szesciokota_foremnego(a):
    return 3 * sqrt(3) * a ** 2 / 2

def obj_gran(bok_pods, wysokosc):
    pole = pole_szesciokota_foremnego(bok_pods)
    return wysokosc * pole
```

Więcej zadań w materiałach powtórzeniowych

6. Podsumowanie oraz pytania powtórzeniowe

Jakie słowo kluczowe jest zarezerwowane dla deklaracji funkcji?

Jakie są korzyści płynące z korzystania z funkcji?

Czy funkcja w każdym uruchomieniu zachowuje się dokładnie tak samo?

Czym jest argument?

Ile możemy mieć argumentów?

Co to znaczy, że argument ma wartość domyślną?

Na jakie sposoby możemy przekazać funkcji wartości do argumentów?

Czy funkcja może coś zwrócić?

Czy funkcja może nie zwracać niczego?