Język skryptowy lab2

Funkcje

Pewne funkcje użyte zostały już we wcześniejszej lekcji była nią np. funkcja `print()` . Były to gotowe funkcje dostarczone wraz z bibliotekami Pythona.

```
print("Hello world!")
a = []
a.extend(range(2, 20))
print(a)
print(str(12))
print(list(range(10, 20, 3)))
```

Poniżej przedstawiono przykład prostej funkcji jaką można napisać samemu. Każda nowo tworzona funkcja posiada nagłówek zbudowany ze słowa `def` nazwy funkcji oraz przyjmowanych przez nią argumentów umieszczonych w nawiasach. Potem jest oczywiście ciało funkcji wykonujące określone czynności. Funkcję wywołać można przez podanie jej nazwy oraz odpowiednich parametrów jakie przyjmuje. Prosta przykładowa funkcja oraz jej wywołanie zaprezentowano poniżej.

> Należy pamiętać, że blok kodu w każdej funkcji rozpoczyna się dwukropkiem i jest on wcięty:exclamation:

```
def my_func():
    print("spam")
    print("spam")
    print("spam")

my_func()
```

Żeby poprawnie wywołać funkcję należy ją napisać przed jej wywołaniem. Rezultatem poniższego wywołania kodu będzie błąd `NameError: name 'hello' is not definied`

```
hello()

def hello():
    print("hello")
```

Komentarze

By ułatwić czytelność kodu stosuje się komentarze. W Pythonie komentarz jednolinowy zapisuje się przez znak `#` na początku komentowanej linii. Natomiast aby utworzyć komentarz wieloliniowy należy użyć potrójnych cudzysłowów.

```
def shout(word):
    """
    komentarz wieloliniowy
    """
    # komentarz jednoliniowy
    print(word + "!")

shout("spam")
```

Argumenty funkcji

Argumenty funkcji używane są jak zmienne w ciele funkcji i nie mogą być użyte poza nią to samo dotyczy zmiennych utworzonych wewnątrz funkcji.

```
def print_with_exclamation(word):
    print(word + "!")

print_with_exclamation("spam")
print_with_exclamation("eggs")
print_with_exclamation("python")
```

```
def print_sum_twice(x, y):
    print(x + y)
    print(x + y)

print_sum_twice(2, 5)
```

Poniższa funkcja w swoim ciele używa parametru `var` który poza ciałem funkcji nie jest dostępny oznacza to, że te zmienne są lokalne

```
def function(var):
    var += 1
    print(var)

function(7)
print(var)
```

Zwracanie wartości z funkcji

Pewne funkcje takie jak `int` lub `str` zwracają wartość, która może być użyta później. Aby funkcja mogła zwracać wartość musi zawierać słowo kluczowe `return` .

```
def max(x, y):
    if x >= y:
```

```
        return x
    else:
        return y

print(max(4, 6))
z = max(9, 3)
print(z)
```

Gdy na pewnym poziomie zagnieżdżenia kodu znajduje się słowo kluczowe `return` żadna z instrukcji, która zostanie napisana potem nie zostanie wykonana.

```
def add_numbers(x, y):
    sum = x + y
    return sum
    print("Ten napis nie wyświetli się")

print(add_numbers(4, 5))
```

Funkcje jako obiekty

Chociaż są one tworzone inaczej niż normalne zmienne, funkcje są jak każda inna wartość. Można je przypisywać do zmiennych i nadpisywać, a następnie wywoływać ich z użyciem nazwy.

```
def multiply(x, y):
    return x * y

a = 4
b = 6
operation = multiply
print(operation(a, b))
```

Funkcja jako argument innej funkcji

Poniżej przedstawiony został przykład jak do jednej z funkcji jako parametr podać inną funkcję

```
def add(x, y):
    return x + y

def do_twice(func, x, y):
    return func(func(x, y), func(x, y))

a = 5
b = 10
print(do_twice(add, a, b))
```

Moduły

```
import random

for i in range(5):
    val = random.randint(1, 6)
    print(val)
```

> Aby zaimportować wszystkie obiekty z modułu należy wykorzystać * np `from math import *` :exclamation:

```
from math import pi

print(pi)
```

Jeśli wystąpi próba importu modułu, który nie jest dostępny np. `import dowolny_modul` generowany jest wyjątek `ImportError`

Importowane moduły lub wybrane obiekty w nich zawarte można używać pod inną nazwą przy pomocy definiowania dla nich słów kluczowych przy użyciu operatora `as`

```
from math import sqrt as square_root

print(square_root(100))
```

W Pythonie występuje 3 główne rodzaje modułów, napisane przez programistę, instalowane z dodatkowych źródeł oraz instalowane łącznie z Pythonem. Wywoływane wcześniej moduły znajdują się w standardowych bibliotekach Pythona. Python posiada wiele użytecznych modułów niektóre z nich umożliwiają opreacje na stringach, obiektach dat, operacje matematyczne, obsługi formatu json, działania na wielu watkach i wiele innych. Wiele z dodatkowych modułów do Pythona znajduje się w PyPI (Python Package Index) aby w prosty sposób móc doinstalować jakiś moduł, który jest potrzebny należy wykorzystać program zwany `pip` . Aby zainstalować interesujący moduł wystarczy przejść do wiersza poleceń i wpisać komendę `pip install nazwa_biblioteki` , po jej zainstalowaniu można już swobodnie wykorzystać jej możliwości we własnym kodzie.

Błędy

Błędy informują użytkownika o tym, że coś poszło nie tak. Gdy wykrywany jest błąd program natychmiast się zatrzymuje jednym z częstszych błędów jest błąd dotyczący dzielenia przez 0 i jest nim `ZeroDevisionError` Innymi często występującymi błędami są:

:small_blue_diamond: `ImportError`

:small_blue_diamond: `IndexError`

:small_blue_diamond: `NameError`

:small_blue_diamond: `SyntaxError`

:small_blue_diamond: `TypeError`

:small_blue_diamond: `ValueError`

W celu obsługi wyjątku, który może wystąpić wykorzystuje się instrukcje `try/except`. Blok `try` może mieć wiele różnych bloków `except`, aby wyłapywać różne rodzaje błędów. Różne rodzaje błędów mogą być także umieszczone w jednym bloku `except`, aby wyłapywać wszystkie z nich i wykonać dla nich tę samą akcję.

```
try:
    var = 100
    print(var + "witaj")
    print(var / 2)
except ZeroDivisionError:
    print("Dzielenie przez 0")
except (ValueError, TypeError):
    print("jakiś błąd")
```

Możliwy jest też zapis bloku `exception` bez informacji o jakimkolwiek konkretnym błędzie można w ten sposób przechwycić wszystkie błędy jakie mogą wystąpić i wykonać dla nich tę samą akcję.

```
try:
    word = "spam"
    print(word / 0)
except:
    print("wykryto błąd")
```

Blok finally

Umieszcza się za blokami `try` i `except`. Używany jest do wykonania jakiegoś fragmentu kodu bez względu na to czy wystąpi bądź nie wystąpi błąd w fragmencie kodu w bloku `try`

```
try:
    print("hello")
    # print(1/0)
except ZeroDivisionError:
    print("Dzielenie przez 0")
finally:
    print("Ta linia kodu wykona się zawsze o bloku try")
```

Instrukcja reise

Służy do poprawiania błędów trzeba podać typ zgłaszanego wyjątku

```
print(1)
raise ValueError
print(2)

name = "123"
raise NameError("błędna nazwa")
print(name)
```

Asercje

Asercje są kontrolą stanu jaki powinno się uzyskać po wykonaniu określonego fragmentu kodu testuje ona i jeśli wynik jest fałszywy to jest rzucany wyjątek

```
print(1)
assert 2 + 2 == 4
print(2)
assert 1 + 1 == 3
```

Asercje mogą również przyjmować drugi argument, który jest przekazywany, jeśli assercja zawiedzie

```
temp = -10
assert temp >= 0, "mniej niż 0"
```

> Gdy asercja się nie wykona poprawnie można wyłapać jej błąd używając bloku `try except` , jeśli to nie będzie zrobione program się zatrzyma:exclamation:

Otwieranie plików

Operacje na plikach w Pythonie przed edycją pliku potrzeba go otworzyć przy pomocy poniższego polecenia

```
myfile = open("spam.txt")
```

> Argumentem funkcji `open` jest ścieżką do pliku. Jeśli plik znajduje się w katalogu projektu wystarczy wpisać jego nazwę :exclamation:

Pliki można otwierać i przetwarzać w różnych trybach. Tryb ustawia się przez podanie go jako drugi argument funkcji `open` podanie `r` umożliwia otwarcie pliku w trybie tylko do odczytu co zresztą jest trybem domyślnym.

Parametr	Opis
w	zapewnia tryb zapisu zawartości pliku (nadpisuje wcześniejszą zawartość lub tworzy plik, gdy nie istnieje)

Parametr	Opis
a	zapewnia możliwość dopisywania do pliku bez utraty wcześniejszej jego zawartości
b	otwiera plik w trybie binarnym używane do plików innych niż tekstowe np. obrazy, dźwięk itp.

```
myfile = open("spam.txt", "w")
myfile = open("spam.txt", "r")
myfile = open("spam.txt", "wb")
```

Odczytywanie pliku umożliwia metoda `read` . Po wykonaniu operacji na pliku obowiązkową czynnością będzie jego zamknięcie przy pomocy wywołania metody `close` wywołanej na rzecz obiektu pliku.

```
file = open("filename.txt", "r")
cont = file.read()
print(cont)
file.close()
```

Do odczytywania można użyć metody `read` wraz z parametrem, który informuje o tym, ile bajtów powinna być odczytana. Metoda ta możne być wykonywana kilka razy z kolei dla tego samego obiektu pliku, w celu odczytania z pliku bajt po bajcie. Wywołanie metody `read` bez argumentu odczyta pozostałe bajty pliku. Gdy zawartość pliku zostanie odczytana w całości kolejna próba odczytania zwróci pusty ciąg znaków.

```
file = open("filename.txt", "r")
print(file.read(10))
print(file.read(5))
print(file.read(6))
print(file.read())
file.close()
```

Do odczytania poszczególnych linii z pliku wykorzystuje się metodę `readlines` która zwraca listę, w której każdy element jest linią z odczytanego pliku.

```
file = open("filename.txt", "r")
print(file.readlines())
file.close()
```

Do odczytu z pliku linia po linii można również wykorzystać pętlę `for`

```
file = open("filename.txt", "r")
for line in file:
```

```
    print(line)
file.close()
```

Zapisywanie plików

Do zapisywania plików wykorzystuję się metodę `write` zapisuje ona ciągi znaków do pliku.

```
file = open("newFile.txt", "w")
file.write("Ten tekst zostanie zapisany do pliku")
file.close()
```

Gdy plik otworzony jest w trybie zapisu istniejąca zawartość pliku zostanie usunięta co pokazuje poniższy fragment kodu.

```
file = open("newFile2.txt", "r")
print("Odczyt zawartości domyślnej")
print(file.read())
print("Koniec")
file.close()

file = open("newFile2.txt", "w")
print(file.write("Jakiś tekst"))
file.close()

file = open("newFile2.txt", "r")
print("Odczyt zawartości domyślnej")
print(file.read())
print("Koniec")
file.close()
```

Jak można zauważyć metoda zapisująca informacje do pliku zwraca ilość zapisanych bajtów do pliku, jeśli zapis się powiedzie.

Dobrą praktyką jest upewnienie się, że pliki zostaną zamknięte po ich użyciu. Jednym ze sposobów na to jest użycie bloków `try` i `finally`.

```
try:
    f = open("fileName.txt")
    print(f.read())
finally:
    f.close()
```

> To zapewnia, że plik jest zawsze zamknięty, nawet jeśli wystąpi błąd:exclamation:

Alternatywą w pracy z plikiem jest użycie operatora `with`. Tworzy on tymczasową zmienną, która jest dostępna wyłącznie na poziomie wciętego bloku instrukcji `with`.

```
with open("filename.txt") as f:
    print(f.read())
```

> Plik zostanie automatycznie zamknięty po wyjściu z bloku `with` nawet jeśli błędy pojawią się w trakcie jego wykonywania:exclamation:

Zadania do wykonania

:one: Wypróbuj kod z *listingów* znajdujących się w instrukcji i sprawdź ich działanie.

:two: Napisz funkcję *lotto*, która do zwracanej listy wpisze 6 losowych i nie powtarzających się liczb z zakresu od 1 do 49.

:three: Stworzony na poprzednich zajęciach kalkulator edytuj tak aby wykorzystywał on funkcję do obliczania wybranych przez użytkownika działań (funkcje przyjmują po 2 parametry i zwracają wynik obliczeń).

:four: Do kalkulatora dopisz możliwość obliczania pola/obwodu koła. (do wzoru wykorzystaj zaimportowaną wartość liczby Pi).

:five: Napisz funkcję, która sumuje liczby z listy podanej jako parametr i podaj ją (tą funkcję) jako parametr do innej funkcji, która odczyta liczby znajdujące się w pliku każda w nowej linii.

:six: Do kalkulatora dopisz funkcję, która umożliwi użytkownikowi zapis działań oraz wyniku dla obliczenia pola/obwodu okręgu do pliku tekstowego.

:seven: Napisz funkcję wyznaczającą rozwiązania równania kwadratowego. Funkcja przyjmuje 3 parametry *a, b, c* i rozwiązania zapisuje do pliku *result.txt*

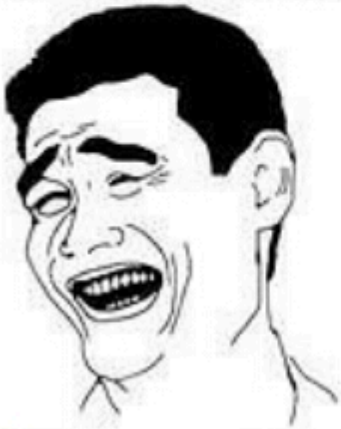
:exclamation: zadania 2-7 mają zostać dodane na GitHuba :exclamation:

Swap integers without additional variable?

CHALLENGE ACCEPTED


```
a = a + b;
b = a - b;
a = a - b;
```

BITCH PLEASE



python

```
a,b = b,a
```