Składnia języka Python oraz podstawowe typy danych

Dokonaj konfiguracji edytora *Visual Studio Code*

1. Pierwszy skrypt

1 pkt za wykonanie wszystkich zadań z tej sekcji

1. Utwórz skrypt 'main.py' zawierający definicję funkcji `sum(arg1, arg2)`, która oblicza, a następnie zwraca, wartość `arg1 + arg2`
2. Wywołaj tę funkcję (w skrypcie) z dwoma argumentami będącymi liczbami całkowitymi oraz wypisz, na ekranie, wynik jej działania następująco: `suma = <wynikSumowania>`
3. Wykonaj skrypt — `python3 main.py`
4. Uruchom konsolę Python — `python3`
5. Sprawdź wynik ewaluacji poniższych wyrażeń:

```
2 + 2
2 + 2.0
2 + '2'
'2' + '2'
zmienna = 2
type(zmienna)
zmienna = '2'
type(zmienna)
```

   Na podstawie otrzymanych wyników wywnioskuj jakie rodzaje typowania oferuje Python:
   - silne
   - słabe
   - statyczne
   - dynamiczne
6. Spowoduj, aby skrypt 'main.py' wypisywał wartość zmiennej *__name__* w postaci następującego łańcucha: `__name__ = <wartośćZmiennej>`
7. Wywołaj komendę `python3 main.py` i zobacz jaka jest wartość tej zmiennej
8. Uruchom konsolę Python, a następnie wykonaj komendę `import main` i ponownie sprawdź, jaka jest wartość ww. zmiennej
   Zobacz czy napis `suma = <wynikSumowania>` pojawia się przy kolejnych wywołaniach `import main`
9. Korzystając z informacji zawartych w artykule, zmodyfikuj skrypt tak, aby nie był

"gadatliwy" w środowisku testowym:
    - jeżeli skrypt jest uruchamiany bezpośrednio, tzn. z linii komend, to funkcja `sum(arg1,arg2)` **ma być wywoływana**, czyli skrypt ma wypisywać wynik sumowania
    - jeżeli skrypt jest ładowany jako moduł, tj. za pomocą `import`, to funkcja **nie ma być wywoływana**

2. Testy jednostkowe "MiniTest"

1 pkt za wykonanie wszystkich zadań z tej sekcji

1. Utwórz skrypt 'test.py', z testami, o następującej zawartości:

```
import main
import unittest


class Test_TestSum(unittest.TestCase):
    def test_sum_integer_integer(self):
        self.assertEqual(main.sum(2, 2), 4)

    def test_sum_integer_float(self):
        self.assertEqual(main.sum(2, 1.5), 3.5)

    # def test_sum_integer_string(self):
    #    self.assertEqual(main.sum(2, '2'), 4)

    # def test_sum_string_string(self):
    #     self.assertEqual(main.sum('2.1', '2.0'), 4.1)

    # def test_sum_integer_wrong_number_in_string(self):
    #     self.assertEqual(main.sum(2, 'Ala ma kota123'), 2)


if __name__ == '__main__':
    unittest.main()
```

2. Uruchom skrypt 'test.py' (`python3 test.py`) i zobacz jakie wyniki zwraca
3. Odkomentuj dodatkowe dodatkowe testy zawarte w liniach 12-19 pliku 'test.rb'
4. Zmodyfikuj definicję (treść) funkcji `sum(arg1, arg2)` tak, aby dodatkowe testy kończyły się powodzeniem — funkcja ma również sumować liczby będące napisami
5. Zmodyfikuj testy (a w razie potrzeby również i skrypt) — oprócz sprawdzania poprawności sumowania liczb całkowitych oraz rzeczywistych, powinny także sprawdzać poprawność sumowania liczb wymiernych oraz zespolonych
6. Zbadaj czy skrypt lub testy wykrywają, że argumentem funkcji jest napis, którego nie da się skonwertować na liczbę: `sum(2, 'Ala ma kota123')`
7. Zmodyfikuj testy — test ma dodatkowo, za pomocą assertRaises(), sprawdzać czy w przypadku podania napisu, którego nie da się skonwertować na liczbę, generowany

jest wyjątek
8. Sprawdź czy testy wykrywają fakt wywołania funkcji z argumentami niebędącym ani liczbą, ani napisem, np. `sum(1, [2, 3])`

3. Typy mnogościowe oraz obsługa linii komend

1. Uruchom konsolę 'python3'
2. Utwórz N-elementową tablicę (klasa list), a następnie wypisz jej elementy w następującej postaci:

```
Element_0
Element_1
...
Element_N-1
```

3. Wypisz jej elementy w następującej formie:

```
Tab[0] = Element_0
Tab[1] = Element_1
...
Tab[N-1] = Element_N-1
```

4. Utwórz N-elementową tablicę asocjacyjną (klasa dict) i wypisz jej elementy następująco:

```
Hash[Klucz_1] = Element_1
Hash[Klucz_2] = Element_2
...
Hash[Klucz_N] = Element_N
```

5. (**1 pkt**) Napisz skrypt, który dla napisów zawartych w linii komend, sprawdza, który z nich jest liczbą pierwszą. Napisy, które nie tworzą prawidłowej liczby mają być ignorowane

```
$ python3 skrypt.py 1 10 13
1
13
```

6. (**2 pkt.**) Napisz:
   - Zestaw funkcji, które przetwarzają dane zawarte w linii komend, w oparciu o metody klasy list, dict oraz set — szczegóły zostaną podane na początku zajęć
   - Test weryfikujący poprawność działania tych funkcji