Klasy i obiekty — część 1

1. Prosta klasa

1 pkt za wykonanie zadań z tej sekcji

1. Przeczytaj artykuł "Wyzwanie Python #4: Programowanie obiektowe"
2. Utwórz skrypt o nazwie 'klasa.py' i następującej zawartości:

```
import sys
##################################
def methodBody(self, name):
    return "Wywołano metodę \033[{color}m{name:^17}\033[0m obiektu \033[
            name     = name,
            color    = "38:5:{}".format(id(self) % 13 + 1),
            objectId = id(self)
        )
##################################
class Klasa(object):
    def __init__(self):
        print(methodBody(self, sys._getframe().f_code.co_name))

    def __del__(self):
        print(methodBody(self, sys._getframe().f_code.co_name))

    def __str__(self):
        return methodBody(self, sys._getframe().f_code.co_name)

    def __repr__(self):
        return methodBody(self, sys._getframe().f_code.co_name)

    def metodaInstancyjna(self):
        print(methodBody(self, sys._getframe().f_code.co_name))

    @classmethod
    def metodaKlasowa(cls):
        print("Wywołano metodę \033[1m{name:^17}\033[0m klasy   \033[1m{
            name = sys._getframe().f_code.co_name,
            cls  = cls.__name__)
        )

    @staticmethod
    def metodaStatyczna():
        print("Wywołano metodę \033[1m{name:^17}\033[0m klasy   \033[1m{
            name = sys._getframe().f_code.co_name,
```

```
            cls  = __class__.__name__)
        )
##################################
print("Załadowano zawartość pliku '{name}'".format(name=__file__))
```

3. Wykonaj komendę `python3 -i klasa.py`
4. Sprawdź jakie metody są wywoływane dla każdej z poniższych linii kodu:

> Nie kopiuj i nie wklejaj wszystkich linii naraz, ale po kolei wklejaj je do konsoli Python

```
obiekt = Klasa()
obiekt = None
obiekt = Klasa()
obiekt = Klasa()
obiekt
print(obiekt)
obiekt.metodaInstancyjna()
Klasa.metodaKlasowa()
Klasa.metodaStatyczna()
# Naciśnij Ctrl+D lub wpisz 'exit()'
```

5. Na początku klasy, przed kodem źródłowym pierwszej metody, zdefiniuj zmienną (własność) o nazwie *tab*, a następnie zainicjuj ją wartością/tablicą: `[1,2,3]`
6. Utwórz skrypt o nazwie 'main.py' zawierający poniższy kod:

```
from klasa import Klasa

obiekt1 = Klasa()
obiekt2 = Klasa()
print('*' * 30)
print("Po utworzeniu obiektów")
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('-' * 10)
Klasa.tab = [4, 5, 6]
print("Po wykonaniu instrukcji \u001b[31mKlasa.tab = [4, 5, 6]\u001b[0m'
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('-' * 10)
print("Po wykonaniu instrukcji \u001b[31mobiekt1.tab = [7, 8, 9]\u001b[0
obiekt1.tab = [7, 8, 9]
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
```

```
print('-' * 10)
print(
    "Po wykonaniu instrukcji '\u001b[31mobiekt2.tab = [-3, -2, -1]\u001b
obiekt2.tab = [-3, -2, -1]
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('*' * 30)
```

7. Uruchom skrypt 'main.py' i sprawdź co jest wypisywane na ekranie — wyjaśnij przyczynę otrzymania takich, a nie innych wyników
8. Spowoduj aby kod `obiekt = Klasa(['a', 'b', 'c'])` powodował utworzenie zmiennej instancyjnej o nazwie *tab* i przypisanie jej wartości tablicy wyspecyfikowanej na liście argumentów konstruktora

   > Jak możesz zauważyć, nazwa proponowanej zmiennej instancyjnej pokrywa się z ... nazwą zmiennej klasowej (statycznej)

9. Zastąp treść skryptu 'main.py' następującą zawartością:

```
from klasa import Klasa

obiekt1 = Klasa(['a', 'b', 'c'])
obiekt2 = Klasa(['x', 'y', 'z'])
print('*' * 30)
print("Po utworzeniu obiektów")
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('-' * 10)
Klasa.tab = [4, 5, 6]
print("Po wykonaniu instrukcji \u001b[31mKlasa.tab = [4, 5, 6]\u001b[0m'
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('-' * 10)
print("Po wykonaniu instrukcji \u001b[31mobiekt1.tab = [7, 8, 9]\u001b[0
obiekt1.tab = [7, 8, 9]
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
print('\tobiekt2.tab ->', obiekt2.tab)
print('-' * 10)
print(
    "Po wykonaniu instrukcji '\u001b[31mobiekt2.tab = [-3, -2, -1]\u001b
obiekt2.tab = [-3, -2, -1]
print('\tKlasa.tab   ->', Klasa.tab)
print('\tobiekt1.tab ->', obiekt1.tab)
```

```
print('\tobiekt2.tab ->', obiekt2.tab)
print('*' * 30)
```

10. Ponownie uruchom skrypt 'main.py' i sprawdź co jest wypisywane na ekranie — wyjaśnij przyczynę otrzymania takich, a nie innych wyników
11. Zmodyfikuj treść klasy *Klasa* zawartą w pliku 'klasa.py':
    - Metoda `metodaInstancyjna()` ma wypisywać wartość obydwu zmiennych *tab* — zarówno instancyjnej, jak i klasowej
    - Zdefiniuj dwie zmienne instancyjne o nazwach: *_zmienna1* oraz *__zmienna2*, a następnie przypisz im (w treści metody `__init__()`) wartości określne w drugim i trzecim argumencie wywołania konstruktora

    ```
    obiekt = Klasa([4, 5, 6], 10, 20)
    ##################################################
    # Zmiennej 'tab' jest przypisywana tablica [4, 5, 6]
    # Zmiennej '_zmienna1' jest przypisywana wartość 10
    # Zmiennej '__zmienna2' jest przypisywana wartość 20
    ##################################################
    ```

12. Wykonaj komendę `python3 -i klasa.py`
13. Wpisz poniższy kod i sprawdź, czy wykonuje się on poprawnie:

```
obiekt = Klasa([4, 5, 6], 10, 20)

print(obiekt.tab)
print(obiekt._zmienna1)
print(obiekt.__zmienna2)
```

14. Jak można zauważyć wykonanie kodu `print(obiekt.__zmienna2)` kończy się błędem. Korzystając z informacji przedstawionych na wykładzie lub w artykule wypisz wartość własności *__zmienna2*

2. Typ wyliczeniowy

1 pkt za wykonanie zadań z tej sekcji

1. Utwórz skrypt o nazwie 'day.py', który zawiera:
    - Klasę o nazwie *Day* z siedmioma wartościami: `MON`, `TUE`, `WED`, `THU`, `FRI`, `SAT` oraz `SUN` — użyj modułu enum
    - Funkcję `nthDayFrom(n, day)` zwracającą dzień tygodnia przesunięty o *n* w stosunku do *day*
    - Metodę instancyjną, dla klasy *Day*, o nazwie `difference(day)` zwracają ilość dni, jaka dzieli dwa dni: dzień reprezentowany przez aktualny obiekt (*self*) oraz dzień reprezentowany przez obiekt *day*
2. Sprawdź poprawność implementacji korzystając z następującego kodu testującego:

```
import unittest
from day import Day, nthDayFrom
```

```
class Test_TestDay(unittest.TestCase):

    def test_nth(self):
        self.assertEqual(nthDayFrom(1, Day.SAT), Day.SUN)
        self.assertEqual(nthDayFrom(2, Day.SAT), Day.MON)
        self.assertEqual(nthDayFrom(-1, Day.TUE), Day.MON)
        self.assertEqual(nthDayFrom(-2, Day.TUE), Day.SUN)

    def test_difference(self):
        self.assertEqual(Day.MON.difference(Day.TUE), 1)
        self.assertEqual(Day.MON.difference(Day.SUN), -1)
        self.assertEqual(Day.SUN.difference(Day.MON), 1)
        self.assertEqual(Day.SUN.difference(Day.SAT), -1)


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

3. Własna klasa

1. (**1 pkt**) Wykonaj poniższe czynności:
    1. Utwórz pakiet *DeaneriySystem* zawierający dwie klasy:
        1. Klasę *Day* — przenieś plik 'day.py' do katalogu pakietu
        2. Klasę *Term* (w pliku 'term.py') w skład której wchodzą:
            - trzy publiczne pola:
                1. *hour* — godzina rozpoczęcia zajęć
                2. *minute* — minuta rozpoczęcia zajęć
                3. *duration* — czas trwania zajęć (w minutach)
            - konstruktor, akceptujący dwa parametry: *hour*, *minute*, przypisujący ww. polom podane wartości, a polu *duration* wartość 90
            - metodę `__str__()`, która zamienia termin na napis postaci: **"godzina:minuta [czas trwania]"**, przykładowo dla: *hour = 9* oraz *minute = 45*, napis powinien mieć postać `"9:45 [90]"`
            - metoda `earlierThan(termin)`, akceptującą inny obiekt tej klasy i zwracającą wartość `True`, jeżeli bieżący termin jest wcześniejszy niż ten podany
            - metoda `laterThan(termin)`, akceptującą inny obiekt tej klasy i zwracającą wartość `True`, jeżeli bieżący termin jest późniejszy niż ten podany
            - metodę `equals(termin)`, zwracającą wartość logiczną `True`, jeżeli obydwa terminy są sobie równe (godzina rozpoczęcia oraz czas trwania zajęć).
    2. Zmodyfikuj klasę *Term*:

- dodaj ,,prywatne" pole *day* typu `Day`
- zmodyfikuj konstruktor tak, aby uwzględniał możliwość przypisania powyższemu polu określonej wartości, tj. dnia tygodnia
- zmodyfikuj metody — mają uwzględniać dni tygodnia

3. Zaimportuj klasę *Term* i sprawdź działanie ww. metod za pomocą poniższego kodu

```
term1 = Term(Day.TUE, 9, 45)
print(term1)                      # Ma się wypisać: "Wtorek 9:45 [90]
term2 = Term(Day.WED, 10, 15)
print(term2)                      # Ma się wypisać: "Środa 10:15 [90]
print(term1.earlierThan(term2)); # Ma się wypisać: "True"
print(term1.laterThan(term2));   # Ma się wypisać: "False"
print(term1.equals(term2));      # Ma się wypisać: "False"
```

4. Stwórz test jednostkowy "MiniTest" sprawdzający poprawność implementacji ww. metod

2. (**2 pkt.**) Rozbuduj klasę o nowe metody — zostaną one wyspecyfikowane na początku zajęć