|  |  |  |  |
| --- | --- | --- | --- |
| Obraz zawierający symbol, design  Opis wygenerowany automatycznie | Politechnika Bydgoska im. J. J. Śniadeckich  **Wydział Telekomunikacji,**  **Informatyki i Elektrotechniki** | | Obraz zawierający symbol, logo, Grafika, Czcionka  Opis wygenerowany automatycznie |
| **Przedmiot** | Skryptowe języki programowania | | |
| **Prowadzący** | mgr inż. Martyna Tarczewska | | |
| **Temat** | *Python - wprowadzenie* | | |
| **Student** | Ivan Fomin | | |
| **Nr ćw.** | 2 | **Data wykonania** | 15.10.2023 |
| **Ocena** |  | **Data oddania spr.** | 15.10.2023 |

1. **Cel ćwiczenia:**

Celem ćwiczenia jest poznanie koncepcji programowania obiektowego w języku Python oraz wykonanie zadań utrwalających przyswojoną wiedzę.

1. **Przygotowanie do ćwiczeń:**

Czytałem dokumentację pliku oraz dokumentację dotyczącą klas w Pythonie.

1. **Zadania/przebieg ćwiczenia:**

Zadanie 1)   
from typing import List

class Student:

    def \_\_init\_\_ (self):

        self.name = ""

        self.last\_name = ""

        self.marks = []

        self.avg = 0

        self.index = 0

    def give\_name(self, name: str, last\_name: str) -> None:

        self.name = name

        self.last\_name = last\_name

    def give\_mark(self, mark: int) -> None:

        self.marks.append(mark)

    def get\_marks(self) -> List[int]:

        return self.marks

    def say\_hello(self) -> None:

        print("Hello! I'm " + self.name + " " + self.last\_name)

    def get\_avg\_marks(self):

        try:

            self.avg = sum(self.marks)/len(self.marks)

            return "AVG: " + str(self.avg)

        except ZeroDivisionError:

            return "No marks!"

    def give\_index(self, index:int)->None:

        self.index = index

    def get\_index(self) -> str:

        return "Index: " + str(self.index)

s = Student()

s.give\_name("Jane", "Doe")

s.give\_mark(5) *# wywołanie sposób 1*

Student.give\_mark(s, 3) *# wywołanie sposób 2*

print(s.get\_marks())

print(s.get\_avg\_marks())

s.say\_hello()

s.give\_index(121191)

print(s.get\_index())

Wyniki:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie

Zadanie 2)

class Vehicle():

    def \_\_init\_\_(self, owner: str, table: str):

        self.owner = owner

        self.table = table

    def get\_sound(self) -> None:

        print("vehicle's brum brum")

    def get\_owner(self) -> str:

        return self.owner

class Car(Vehicle):

    def get\_sound(self) -> None:

        print("car's brum brum")

car = Car("John", "A3")

vehicle = Vehicle("Alice", "X6")

car.get\_sound()

print(car.get\_owner())

vehicle.get\_sound()

print(vehicle.get\_owner())

tak wywołanie get\_owner() dla instancji klasy Vehicle powodowało błąd , bo metoda klasy podrzędnej nie jest widoczna dla instancji klasy nadrzędnej .Dlatego najpierw get\_owner zrobiłem w klasie nadrzędnej.

Wyniki:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie

Zadanie 3)

class Student:

    quantity = 0

    def \_\_init\_\_ (self):

        self.name = ""

        self.last\_name = ""

        self.marks = []

        Student.quantity += 1

s1 = Student()

print(s1.quantity)

s2 = Student()

print(s2.quantity)

s3 = Student()

print(s3.quantity)

s4 = Student()

print(s4.quantity)

print(s1.quantity)

print(s2.quantity)

print(Student.quantity)

"""

 ta zmienna quantity jest zmienną klasy a dla każdego jej objektu jest taka sama

 (jakby była self.quantity w funkcji init to dla każdej instancji była by swoja)

"""

Wyniki:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Opis wygenerowany automatycznie

Zadanie 4)

from typing import List

class Student:

    def \_\_init\_\_(self):

        self.name = ""

        self.last\_name = ""

        self.marks = []

    def \_check\_name(self, name: str, last\_name: str) -> bool:

        return bool(name.strip() and last\_name.strip())

*# strip()jest używana do usunięcia białych znaków z początku i końca stringa*

*# w tym kodzie sprawdza czy jest to pusty string(czyli składa się tylko z białych znaków lub są całkowicie pusty)*

    def give\_name(self, name: str, last\_name: str) -> None:

        self.name = name

        self.last\_name = last\_name

    def give\_mark(self, mark: int) -> None:

        self.marks.append(mark)

    def get\_marks(self) -> List[int]:

        return self.marks

    def say\_hello(self) -> None:

        if self.\_check\_name(self.name, self.last\_name):

            print("Hello! I'm " + self.name + " " + self.last\_name)

        else:print("You have not name OR last name")

s = Student()

s.give\_name("Jane", "Doe")

s.say\_hello()

s.give\_name("", "Smith")

s.say\_hello()

Wyniki:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie  
  
Zadanie 5)

class Item:

    def get\_sound(self) -> None:

        print("item's sound")

class Element:

    def get\_sound(self) -> None:

        print("element's sound")

class Thing(Element,Item):

    def say\_hello(self) -> None:

        print("hello!")

"""

class Thing(Item,Element):

    def say\_hello(self) -> None:

        print("hello!")

"""

i = Item()

i.get\_sound()

e = Element()

e.get\_sound()

t = Thing()

t.get\_sound()

gdy item jest pierwszy w klasach nadrzędnych to „t” używa metodę klasy Item ,  
gdy Element jest pierwszy , to metodę klasy Element  
W przypadku metody get\_sound, która jest zdefiniowana zarówno w klasie Item, jak i w klasie Element, Python poszukuje metody w kolejności dziedziczenia. Oznacza to, że jeśli metoda jest obecna w klasie Thing, zostanie ona użyta. Jeśli nie, Python przeszuka klasy dziedziczone przez Thing w kolejności, w jakiej zostały podane (najpierw Item, potem Element), aby znaleźć odpowiednią metodę.

Wyniki:  
Element pierwszy:  
Obraz zawierający tekst, Czcionka, typografia, design

Opis wygenerowany automatycznie

Item pierwszy:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie

Zadanie 6)

class Student:

    def \_\_init\_\_(self, name:str, last\_name:str, index:int):

        self.name = name

        self.last\_name = last\_name

        self.index = index

    def \_\_lt\_\_(self, other)->bool:

        return self.index < other.index

    def \_\_gt\_\_(self, other)->bool:

        return self.index > other.index

    def \_\_eq\_\_(self, other)->bool:

        return self.index == other.index

    def \_\_le\_\_(self, other)->bool:

        return self.index <= other.index

    def \_\_ge\_\_(self, other)->bool:

        return self.index >= other.index

    def \_\_ne\_\_(self, other)->bool:

        return self.index != other.index

s1 = Student("Jan", "Kowalski", 121345)

s2 = Student("Bob", "Marley", 122890)

print("s1 < s2:", s1 < s2)

print("s1 > s2:", s1 > s2)

print("s1 == s2:", s1 == s2)

print("s1 <= s2:", s1 <= s2)

print("s1 >= s2:", s1 >= s2)

print("s1 != s2:", s1 != s2)

Wynik:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie

Zadanie 7)

class NewClass:

    def \_\_init\_\_(self, value):

        self.value = value

*# te metody  pozwalają na zdefiniowanie reprezentacji tekstowej obiektów*

    def \_\_repr\_\_(self):

        return f"NewClass({self.value})"

    def \_\_str\_\_(self):

        return f"Value: {self.value}"

object1 = NewClass(42)

print(repr(object1))

print(str(object1))

print(object1)

"""

Jeśli klasa implementuje zarówno \_\_repr\_\_ jak i \_\_str\_\_

print() korzysta z \_\_str\_\_ ale jeśli \_\_str\_\_ nie jest dostępne, używane jest \_\_repr\_\_.

W przypadku braku obu metod, używana jest domyślna reprezentacja obiektu, która obejmuje jego adres w pamięci.

"""

Wyniki:

Z:  
Obraz zawierający tekst, Czcionka, typografia, Grafika

Opis wygenerowany automatycznie

Bez:  
Obraz zawierający tekst, Czcionka, zrzut ekranu, typografia

Opis wygenerowany automatycznie

Zadanie 8)

class NewClass:

    def \_\_init\_\_(self, x:int, y:int):

        self.x = x

        self.y = y

    def sum(self)->int:

        return self.x + self.y

o = NewClass(1, 2)

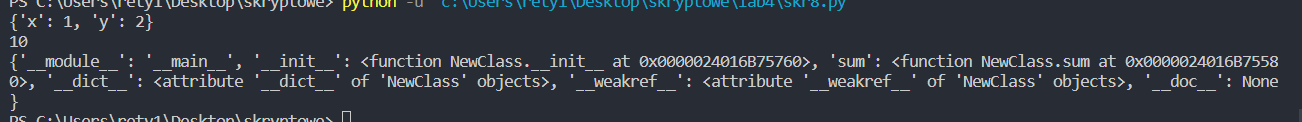
print(o.\_\_dict\_\_)

o.\_\_dict\_\_['x'] = 10

print(o.x)

print(NewClass.\_\_dict\_\_)

\_\_dict\_\_ przechowuje słownik zawierający przestrzeń nazw (czyli zmienne i metody) obiektu. Działa jako słownik, który mapuje nazwy atrybutów (klucz) na ich wartości (wartość). Można uzyskać dostęp do \_\_dict\_\_ obiektu, aby odczytać lub modyfikować jego atrybuty dynamicznie.

Wyniki:  


Zadanie 9)

class IceCream():

    def \_\_init\_\_(self, packaged:bool, mainingridient:str):

        self.packeged = packaged

        self.mainingridient = mainingridient

class Popsicle(IceCream):

    def \_\_init\_\_(self, packaged: bool, mainingridient: str,rate:float):

        self.rate = rate

        super().\_\_init\_\_(packaged, mainingridient)

product = Popsicle(True,"Ice Juice",6.7)

print(product.\_\_dict\_\_)

*# najpierw jest wykonywana inicjalizacja w inicjalizacja w klasie dziedziczącej(self.rate) a potem po wykonywaniu super()*

*# z klasy  nadrzędnej (self.packeged oraz self.mainingridient)*

Wyniki:  


Zadanie Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie10)Testy:

W I teście sprawdziłem czy instancja o jest ma typ obiekt

W II teście sprawdziłem czy metody sum() dwóch instancji z takimi samymi x,y zwracają taki samy wynik

W III teście porównanie dwóch instancji z równymi x,y

**4. Wnioski:**W tym labie zapoznałem się z klasami w Python; poznawałem koncepcję programowania obiektowego w języku Python oraz wykonałem zadania utrwalając przyswojoną wiedzę