Wstęp do programowania
semestr zimowy 2025/2026

Dr Anna Muranova
UWM w Olsztynie

Wykład 8

Klasy

Pierwszym krokiem tworzenia własnych obiektów jest stworzenie klasy, która jest czymś w rodzaju projektu, na podstawie którego tworzone są obiekty, które są egzemplarzami klasy. Obiekty mogą przechowywać dane, a także wykonywać kod, który jest umieszczony w metodach. Metody są funkcjami, które są częściami klasy.
Klasa opisuje pewien typ obiektów. Jest więc czymś w rodzaju "projektu" na podstawie którego tworzy się konkretne obiekty, które są osobnymi bytami. W klasie możemy zdefiniować przechowywane przez obiekty rodzaje informacji, a także operacje, które obiekty będą wykonywać. Najpierw więc tworzymy projekt (klasę), a potem na jej podstawie konkretne obiekty.
Klasy można umieszczać w osobnych plikach (modułach), zwłaszcza jeśli są bardzo rozbudowane, ale wiele klas może być także umieszczonych w jednym module.

`https://www.w3schools.com/python/python_classes.asp`
`https://kt.academy/pl/article/py-klasy`
`https://ggoralski.gitlab.io/python-wprowadzenie/czesc_i/15-klasy_i_obiekty/`

Pierwszy przykład

Zacznijmy od najprostszej opcji, czyli pustej klasy. Taka klasa nic nie będzie zawierać, niemniej będzie miała swoją nazwę i będziemy mogli przy jej pomocy stworzyć obiekt

```
class Cookie:
    pass


cookie = Cookie()
```

Przy nazywaniu klas możemy używać tych samych znaków co w przypadku zmiennych i funkcji: małych i dużych liter oraz znaku podkreślenia _. Konwencja nazewnicza jest jednak inna. Dla funkcji i zmiennych używaliśmy `snake_case`. W przypadku klas używamy `PascalCase` (lub `UpperCamelCase`), czyli każde słowo zaczynamy wielką literą, nie używamy spacji ani znaków podkreślenia.

Zmienne obiektu 1

Do obiektu możemy przypisać zmienną z określoną wartością. Taka wartość dotyczyć będzie wyłącznie tego jednego obiektu. Aby odnieść się do zmiennej w obiekcie, musimy wskazać zarówno obiekt, jak i zmienną, a oddzielamy ich nazwy kropką. Dla przykładu, aby odnieść się do zmiennej type w obiekcie cookie1, użyjemy cookie1.type. Zarówno do przypisania wartości, jak i do jej pobrania.

```
class Cookie:
    pass


cookie1 = Cookie()
cookie2 = Cookie()
cookie1.type = "Biscuit"
cookie1.color = "White"
cookie2.type = "Oreo"
print(cookie1.type)  # Biscuit
print(cookie1.color)  # White
print(cookie2.type)  # Oreo
# print(cookie2.color)
```

Konstruktor i inicjalizator 1

Nieczęsto tworzy się zmienne obiektu tak jak w powyższym przykładzie: poza klasą. Często jest to wręcz uznawane za złą praktykę. Częściej tworzy się je w obrębie metod, a zwłaszcza szczególnej metody zwanej inicjalizatorem.
Gdy tworzymy nowy obiekt, stawiamy nawias za nazwą klasy. Ten nawias to wywołanie funkcji tworzącej obiekt, zwanej konstruktorem. Funkcja ta przechodzi przez szereg kroków, niezbędnych do utworzenia obiektu, w tym między innymi woła specjalną metodę o nazwie `__init__` z naszej klasy. Ta metoda zwana jest inicjalizatorem. W jej ciele określamy, co powinno się dziać w czasie tworzenia obiektu. Najczęściej definiujemy w niej atrybuty obiektu.

```
class Cookie:
    def __init__(self, type, color):
        self.type = type
        self.color = color
#pierwszym parametrem jest odniesienie do instancji obiektu,
#na którym tę metodę wywołamy

cookie1 = Cookie("Biscuit","White")
print(cookie1.type)  # Biscuit
print(cookie1.color)  # White
```

Konstruktor i inicjalizator 2

Liczba parametrów funkcji `__init__` określa, ile argumentów powinno się znaleźć w wywołaniu konstruktora (czyli nawiasie, który stawiamy za nazwą klasy, gdy tworzymy obiekt). Jeśli więc w funkcji `__init__` dodamy parametr name, to przy tworzeniu obiektu nie możemy już zostawić pustego nawiasu. Powinniśmy podać tam argument, który posłuży jako imię. Typowym dla funkcji `__init__` jest, że spodziewa się określonych parametrów, po czym przypisuje je do obiektu jako atrybuty o takiej samej nazwie.

```
class Cookie:
    def __init__(self, type, color = None):
        self.type = type
        self.color = color


cookie1 = Cookie("Biscuit","White")
cookie2 = Cookie("Oreo")
print(cookie1.type)  # Biscuit
print(cookie1.color)  # White
print(cookie2.type)  # Oreo
print(cookie2.color) #None
```

Jeszcze przykład

```
class Player:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname
        self.full_name = f"{name} {surname}"
        self.points = 0


player = Player("Michał", "Mazur")
print(player.name)  # Michał
print(player.surname)  # Mazur
print(player.full_name)  # Michał Mazur
print(player.points)  # 0
```

Metody

Wewnątrz klas możemy definiować funkcje. Takie funkcje nazywane są metodami. Definiujemy je w ciele klasy, a ich pierwszym parametrem jest odniesienie do instancji obiektu, na którym tę metodę wywołamy. Parametr ten powinno nazywać się `self`. Gdy wywołujemy metodę, zaczynamy od obiektu, następnie stawiamy kropkę, nazwę metody i nawias z argumentami.

```
class Position:
    def __init__(self, x = 0.0, y = 0.0):
        self.x = x
        self.y = y
    def step_right(self):
        self.x += 1.0
    def move_up(self, value):
        self.y += value

pos = Position()
pos.step_right()
print(pos.x)  # 1.0
pos.move_up(6.0)
print(pos.y)  # 6.0
pos.move_up(3.0)
print(pos.y)  # 9.0
```

Obiekty i zmienne 1

Każdy obiekt jest osobnym bytem. To, że wyglądają podobnie, nie znaczy, że mają na siebie wpływ. Dlatego też w poniższym przykładzie zmiana name w obiekcie user1 nie będzie miała żadnego wpływu na user2.

```
class User:
    def __init__(self, name):
        self.name = name


user1 = User("Rafał")
user2 = User("Rafał")

print(user1.name)  # Rafał
print(user2.name)  # Rafał

user1.name = "Bartek"

print(user1.name)  # Bartek
print(user2.name)  # Rafał
```

Obiekty i zmienne 1

Każdy obiekt jest osobnym bytem. To, że wyglądają podobnie, nie znaczy, że mają na siebie wpływ. Dlatego też w poniższym przykładzie zmiana name w obiekcie user1 nie będzie miała żadnego wpływu na user2.

```
class User:
    def __init__(self, name):
        self.name = name


user1 = User("Rafał")
user2 = User("Rafał")

print(user1.name)  # Rafał
print(user2.name)  # Rafał

user1.name = "Bartek"

print(user1.name)  # Bartek
print(user2.name)  # Rafał
```

Obiekty i zmienne 2

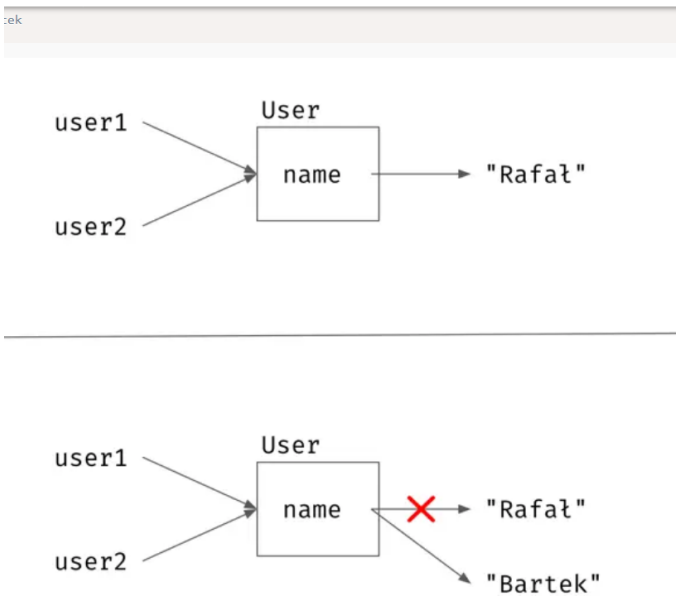
Z drugiej strony, jeśli mamy dwie zmienne wskazujące na jeden obiekt, to możemy go zmienić przy użyciu dowolnej z nich. Po takim zabiegu, wartości dla obu zmiennych ulegną zmianie, bo przecież przekształcone zostało coś, na co obydwie wskazują.

```
user1 = User("Rafał")
user2 = user1

print(user1.name)
# Rafał
print(user2.name)
# Rafał

user1.name = "Bartek"

print(user1.name)
# Bartek
print(user2.name)
# Bartek
```



Rysunek: Źródło: https://kt.academy/pl/article/py-klasy

Obiekty i zmienne 3

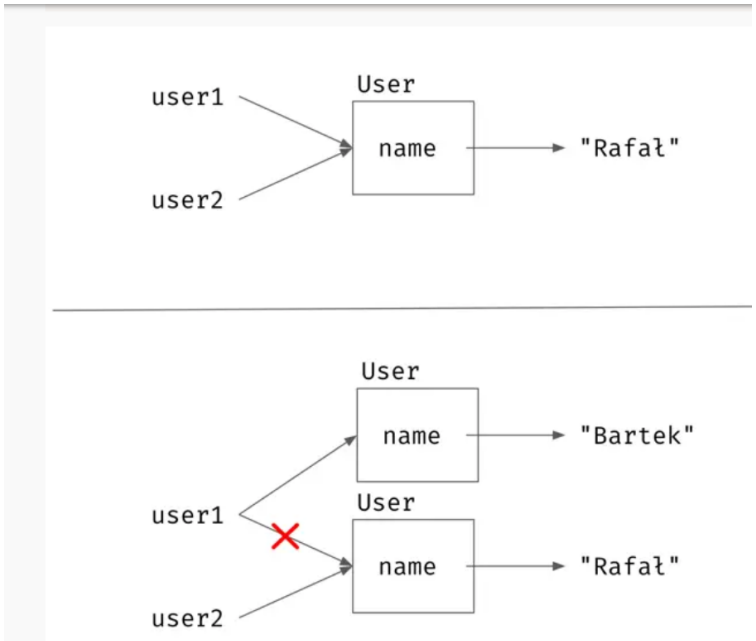
Dwie zmienne wskazują na ten sam obiekt i właściwość tego obiektu zmienia wartość.
Warto porównać to z przykładem, gdy dwa obiekty pokazywały na tą samą wartość, a potem zmieniło się to, na co jedna z tych zmiennych wskazuje.
Wynik będzie inny.

```
user1 = User("Rafał")
user2 = user1

print(user1.name)
# Rafał
print(user2.name)
# Rafał

user1 = User("Bartek")

print(user1.name)
# Bartek
print(user2.name)
# Rafał
```



Rysunek: Źródło: https://kt.academy/pl/article/py-klasy

Metoda copy()

Dwie zmienne wskazują na ten sam obiekt i właściwość tego obiektu zmienia wartość.

```
class User:
    def __init__(self, name):
        self.name = name

    def copy(self):
        return User(self.name)

user1 = User("Rafał")
user2 = user1.copy()
user1.name = "Bartek"
print(user1.name)  # Bartek
print(user2.name)  # Rafał
user2.name = "Marek"
print(user1.name)  # Bartek
print(user2.name)  # Marek
```

Metoda __str__()

```
class User:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f'Username:{self.name}'

user1 = User("Rafał")
print(user1)
```

Przykład: klasa liczb rzymskich

Funkcja pierwsza:

```
def to_arabic(number):
    roman_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100,
                                    'D': 500, 'M': 1000}
    # 4 (IV) and 9 (IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM)
    number = number.replace('IV', 'IIII')
    number = number.replace('IX', 'VIIII')
    number = number.replace('XL', 'XXXX')
    number = number.replace('XC', 'LXXXX')
    number = number.replace('CD', 'CCCC')
    number = number.replace('CM', 'DCCCC')
    return sum(roman_numerals[i] for i in number)

print(to_arabic('MCDLXIV'))
```

Przykład: klasa liczb rzymskich

Funkcja druga:

```
def to_roman(n):
    roman_numerals = {1000:'M', 900: 'CM', 500: 'D', 400: 'CD', 100:
                                'C', 90:'XC',50: 'L', 40:'XL',
                                10:'X', 9:'IX', 5: 'V' ,4:'IV', 1: 'I'}
    s = ''
    for i in roman_numerals:
        div = n // i
        n %= i
        while div:
            s += roman_numerals[i]
            div -= 1
    return s

print(to_roman(1464))
```

Przykład: konstruktor

```
class Roman:
    def __init__(self, n):
        #two constructors are not allowed
        if isinstance(n, str):
            self.roman = n
            roman_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50,
                              'C': 100, 'D': 500, 'M': 1000}

            n = n.replace('IV', 'IIII')
            n = n.replace('IX', 'VIIII')
            n = n.replace('XL', 'XXXX')
            n = n.replace('XC', 'LXXXX')
            n = n.replace('CD', 'CCCC')
            n = n.replace('CM', 'DCCCC')
            self.arabic = sum(roman_numerals[i] for i in n)
```

Przykład: konstruktor dalej

```
        elif isinstance(n, int):
            if n<=0:
                raise ValueError("Roman number should be positive")
            self.arabic = n
            romanian_numerals = {1000: 'M', 900: 'CM', 500: 'D',
                       400: 'CD', 100: 'C', 90: 'XC', 50: 'L', 40: 'XL'
                       10: 'X', 9: 'IX', 5: 'V', 4: 'IV', 1: 'I'}
            s = ''
            for i in romanian_numerals:
                div = n // i
                n %= i
                while div:
                    s += romanian_numerals[i]
                    div -= 1
            self.roman = s


    def __str__(self):
        return self.roman

print(Roman(10))#X
```

Przykład: funkcja

```
class Roman:
    def __init__(self, n):
        ...
    def __str__(self):
        ...

r = Roman("XVI")
print(r.arabic)#16
```

Przykład: dodawanie i mnożenie

```
class Roman:
    def __init__(self, n):
        ...
    def __str__(self):
        ...

    def __add__(self, other):
        return Roman(self.arabic+other.arabic)

    def __mul__(self, other):
        return Roman(self.arabic*other.arabic)
r = Roman("XVI")
p = Roman("X")
print(r+p)#XXVI
print(r*p)#CLX
```

Przykład: odejmowanie i dzielenie

```
class Roman:
    ....

    def __sub__(self, other):
        return Roman(self.arabic-other.arabic)

    def __truediv__(self,other):
        if self.arabic%other.arabic == 0:
            return Roman(self.arabic//other.arabic)
        raise ValueError ("Roman number should be integer")

print(Roman('X')-Roman('IX'))#I
print(Roman('XX')/Roman('X'))#II

print(Roman('X')-Roman('X'))#ValueError
print(Roman('VIII')/Roman('X'))#ValueError
```

Dziedziczenie(inheritance)

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

 class Student(Person):
  pass


x = Person("John", "Doe")
x.printname()

x = Student("Mike", "Olsen")
x.printname()
```

Dziedziczenie(inheritance)

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname, number):
        Person.__init__(self, fname, lname)
        #super().__init__(fname, lname)
        self.number = number


x = Person("John", "Doe")
x.printname()

x = Student("Mike", "Olsen", 1999)
x.printname()
```

Jeszcze przykład

```
class Fruit:

    def __init__(self, name, sugar_content):
        self.name = name
        self.sugar_content = sugar_content

class Apple(Fruit):

    def __init__(self):
        super().__init__("apple", 2)
```