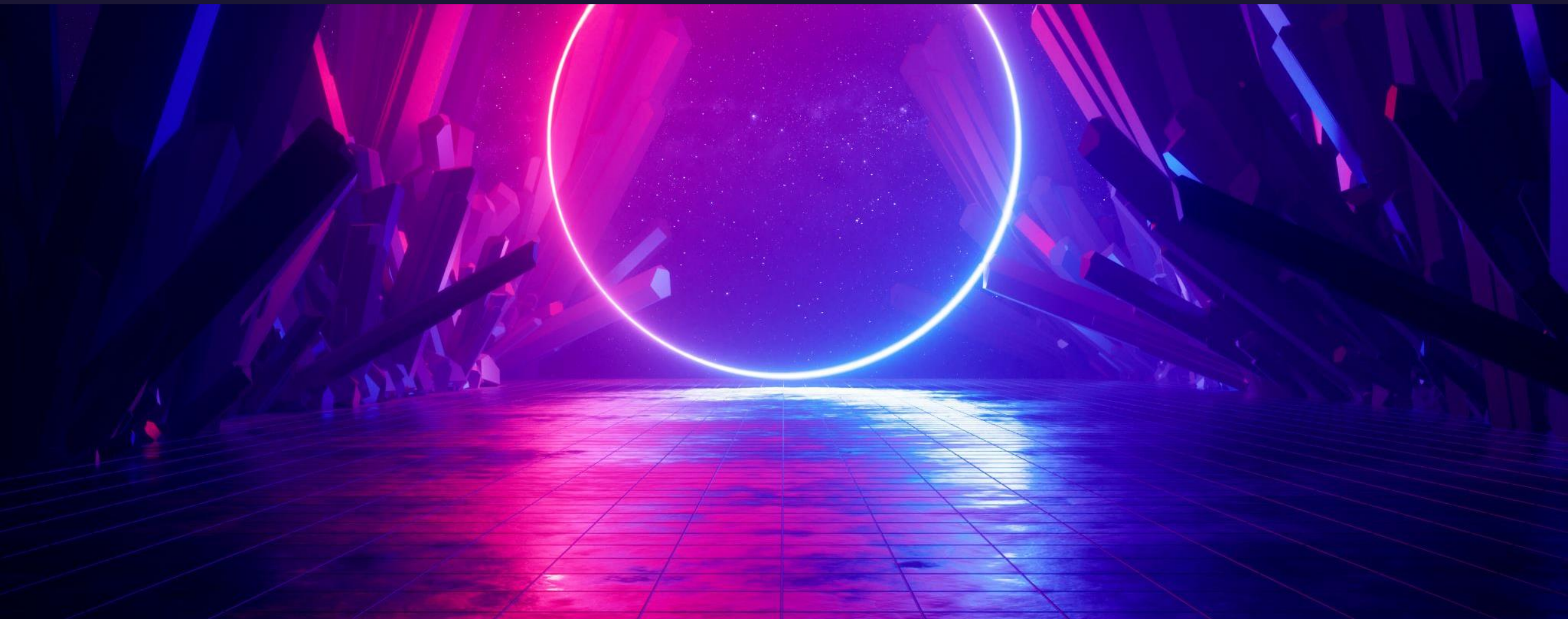
Programowanie w Pythonie

Łukasz Mioduszewski, UKSW 2023

Wreszcie programowanie obiektowe



Klasy

- Wszystko w Pythonie jest obiektem, czyli instancją pewnej klasy
- Klasa zawiera
  zmienne (pola, atrybuty)
  i funkcje (metody):

```
class name:
    statements
```

```
>>> a = 123
>>> type(a)
<class 'int'>
>>> b = "abc"
>>> type(b)
<class 'str'>
>>> c = [1, 2]
>>> type(c)
<class 'list'>
```

Przykład

```
class Point:
    x = 0
    y = 0

# main
p1 = Point()
p1.x = 2
p1.y = -5
```

Point.py

```
1  class Point:
2      x = 0
3      y = 0
```

- Zmienne można deklarować od razu w klasie (jak tutaj) albo w inicjalizatorze

Tworzenie obiektów

- Moduły mogą zawierać klasy, których po zaimportowaniu modułu można użyć
- Do utworzonych obiektów można dodawać nowe pola

point_main.py

```
1  from Point import *
2
3  # main
4  p1 = Point()
5  p1.x = 7
6  p1.y = -3
7  ...
8
9  # Python objects are dynamic (can add fields any time!)
10 p1.name = "Tyler Durden"
```

Metody

Deklarujemy je wewnątrz klasy:

- `def` **`name`**`(self,` **`parameter, ..., parameter`**`):`
  statements
- `self` musi być pierwszym parametrem
- do pól odnosimy się poprzez self.pole

```
class Point:
    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
    ...
```

Porównanie z Javą

- Java: `this`, implicit

```
public void translate(int dx, int dy) {
    x += dx;        // this.x += dx;
    y += dy;        // this.y += dy;
}
```

- Python: `self`, explicit

```
def translate(self, dx, dy):
    self.x += dx
    self.y += dy
```

Przykład

point.py

```
 1  from math import *
 2
 3  class Point:
 4      x = 0
 5      y = 0
 6
 7      def set_location(self, x, y):
 8          self.x = x
 9          self.y = y
10
11      def distance_from_origin(self):
12          return sqrt(self.x * self.x + self.y * self.y)
13
14      def distance(self, other):
15          dx = self.x - other.x
16          dy = self.y - other.y
17          return sqrt(dx * dx + dy * dy)
```

Wywoływanie metod

- Można na dwa sposoby:
  - (wartość parametru `self` może być jawna lub niejawna)

1) **object . method (parameters)**

albo (po co jest ten drugi sposób, okaże się później)

2) **Class . method (object, parameters)**

- Przykład:

```
p = Point(3, -4)
p.translate(1, 5)
Point.translate(p, 1, 5)
```

Inicjalizator

```
def __init__(self, parameter, ..., parameter):
    statements
```

- Inicjalizator to specjalna metoda o nazwie `__init__`
- Przykład:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    ...
```

- Zagadka: jak napisać inicjalizator, gdzie `Point()` bez parametrów daje punkt (0, 0)?

Rzutowanie na string

```
def __str__(self):
    return string
```

- wywołuje się automatycznie kiedy obiekt trafia do funkcji `str` albo `print`
- Przykład:

```
def __str__(self):
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```

Przykład

point.py

```
 1  from math import *
 2
 3  class Point:
 4      def __init__(self, x, y):
 5          self.x = x
 6          self.y = y
 7
 8      def distance_from_origin(self):
 9          return sqrt(self.x * self.x + self.y * self.y)
10
11      def distance(self, other):
12          dx = self.x - other.x
13          dy = self.y - other.y
14          return sqrt(dx * dx + dy * dy)
15
16      def translate(self, dx, dy):
17          self.x += dx
18          self.y += dy
19
20      def __str__(self):
21          return "(" + str(self.x) + ", " + str(self.y) + ")"
```

Inny przykład

point.py

```
class Circle:
  def __init__(self, radius):
    self.radius = radius
  def get_area(self):
    return (self.radius ** 2) * 3.14
  def get_perimeter(self):
    return self.radius * 2 * 3.14
```

Przeładowanie operatorów

- Można zdefiniować jak obiekt ma się zachować dla standardowych operatorów:

Operator	Metoda
-	`__sub__(self, other)`
+	`__add__(self, other)`
*	`__mul__(self, other)`
/	`__truediv__(self, other)`

-	`__neg__(self)`
+	`__pos__(self)`

Operator	Metoda
==	`__eq__(self, other)`
!=	`__ne__(self, other)`
<	`__lt__(self, other)`
>	`__gt__(self, other)`
<=	`__le__(self, other)`
>=	`__ge__(self, other)`

- Więcej operatorów: https://www.geeksforgeeks.org/operator-overloading-in-python/

Przerywnik: zgłaszanie błędów

`raise` **`ExceptionType`**`("`**`message`**`")`

- Kiedy chcesz aby program wyrzucił taki błąd jaki chcesz:
- rodzaje: `ArithmeticError`, `AssertionError`, `IndexError`, `NameError`, `SyntaxError`, `TypeError`, `ValueError`

- przykład:

```
class BankAccount:
    ...
    def deposit(self, amount):
        if amount < 0:
            raise ValueError("negative amount")
        ...
```

Dziedziczenie

```
class name(superclass):
    statements
```

- Przykład:

```
class Point3D(Point):    # Point3D extends Point
    z = 0
    ...
```

- Można dziedziczyć z wielu klas naraz:

```
class name(superclass, ..., superclass):
    statements
```

Dziedziczenie

- Możemy wewnątrz nowej klasy używać metod starej klasy:
- **class . method** (**object, parameters**)
- Przykład:

```
class Point3D(Point):
    z = 0
    def __init__(self, x, y, z):
        Point.__init__(self, x, y)
        self.z = z

    def translate(self, dx, dy, dz):
        Point.translate(self, dx, dy)
        self.z += dz
```

Hermetyzacja

- Dostęp do pól, które powinny być prywatne, może powodować błędy
- Przykład:
  - Chcemy umożliwić sterowanie w taki sposób, aby przy ustawieniach:
    cała wstecz, silniki stop, cała naprzód niemożliwa była zmiana z pozycji
    cała naprzód do cała wstecz i odwrotnie – z pominięciem silniki stop
- Prywatne pola lub metody mają nazwy zaczynające się od **pojedynczego** _

Przykład

silnik.py

```
1  class SilnikOkretowy:
2   stany = ["Cała wstecz", "Silniki STOP", "Cała naprzód"]
3   def __init__(self):
4    self.wskazanie = 1
5    self.stan_silnika = self.stany[self.wskazanie]
6    self.aktualizuj_silniki(0)
7   def aktualizuj_silniki(self, krok):
8    if krok < 0 and self.wskazanie > 0:
9     print("Zmiana silników")
10    self.wskazanie += krok
11   if krok > 0 and self.wskazanie < 2:
12    print("Zmiana silników")
13    self.wskazanie += krok
14   self.stan_silnika = self.stany[self.wskazanie]
15   print(f"Aktualnie silniki: {self.stan_silnika}")
16  def silniki_naprzod(self):
17   self.aktualizuj_silniki(1)
18  def silniki_wstecz(self):
19   self.aktualizuj_silniki(-1)
```

Przykład

Dobre działanie

```
1  # inicjalizacja zmiennej obiektowej
2  batory = SilnikOkretowy()
3  # Aktualnie silniki: Silniki STOP
4  batory.silniki_naprzod()
5  # Zmiana silników
6  # Aktualnie silniki: Cała naprzód
7  batory.silniki_naprzod()
8  # Aktualnie silniki: Cała naprzód
9  batory.silniki_naprzod()
10 # Aktualnie silniki: Cała naprzód
11 batory.silniki_wstecz()
12 # Zmiana silników
13 # Aktualnie silniki: Silniki STOP
14 batory.silniki_wstecz()
15 # Zmiana silników
16 # Aktualnie silniki: Cała wstecz
17 batory.silniki_wstecz()
18 # Aktualnie silniki: Cała wstecz
```

Przykład

Złe działanie

```
print(batory.stan_silnika)
# Cała wstecz
print(batory.wskazanie)
# 0
batory.wskazanie = 2
print(batory.wskazanie)# 2
print(batory.stan_silnika)
# Cała wstecz
batory.aktualizuj_silniki(0)
# Aktualnie silniki: Cała naprzód
# ręczna zmiana może sprawić, że odwoływam się do indeksu
# większego niż rozmiar tablicy i zostanie zwrócony błąd
batory.wskazanie = 5
batory.aktualizuj_silniki(0)
# Traceback (most recent call last):
# File "<pyshell#55>", line 1, in <module>
# batory.aktualizuj_silniki(0)
# File "/home/python/idle-src/0373_silniki_okr_hermet.py", li
# self.stan_silnika = self.stany[self.wskazanie]
# IndexError: list index out of range
```

Hermetyzacja

- Zaczynanie metod i pól od podkreślnika powoduje, że wywoływanie ich poza klasą nie powoduje błędu
- Pomocne w debuggingu i przy dużych projektach, ale dla interpretera nie ma znaczenia

Polimorfizm

- Oznacza wielopostaciowość. Pozwala stosować jedną nazwę (metody, operatora lub obiektu) do reprezentowania różnych rzeczy w różnych klasach
- Przykład wbudowany: operator +
- Przykład: klasa Statek i klasy dziedziczące Bryg oraz Fregata
  Ich konstruktory wykorzystają funkcję super(). Pozwala to na używanie metod z klasy Statek w klasie potomnej. Obie klasy potomne będą miały metody test() oraz info()

Przykład

Statki.py

```
1  class Statek:
2   def __init__(self, rok_wodowania):
3    self._rok_wodowania = rok_wodowania
4    print(f"Utworzono obiekt {self}.")
5   def nowy_rok(self):
6    self._rok_wodowania += 1
7   def rok_wodowania(self):
8    return self._rok_wodowania
9  class Bryg(Statek):
10  def __init__(self, rok_wodowania):
11   super().__init__(rok_wodowania)
12   self._typ = "Bryg/2 maszty"
13  def info(self):
14   print(f"Metoda w klasie {__class__.__name__}")
15   print(f"Obiekt {self} - typ: {self._typ}")
16   print(f"Rok wodowania = {super().rok_wodowania()}")
17  def test(self):
18   print("To jest wywołanie test-Bryg (rok_wodowania + 20)")
19   print(f"wynik = {super().rok_wodowania()+20}")
```

Przykład

Statki.py

```
20 class Fregata(Statek):
21  def __init__(self, rok_wodowania):
22   super().__init__(rok_wodowania)
23   self._typ = "Fregata/3 maszty"
24
25  def info(self):
26   print(f"Metoda w klasie {__class__.__name__}")
27   print(f"Obiekt {self} - typ: {self._typ}")
28   print(f"Rok wodowania = {super().rok_wodowania()}")
29  def test(self):
30   print("To jest wywołanie test-Fregata (rok_wodowania + 40)")
31   print(f"wynik = {super().rok_wodowania()+40}")
32
33
34
35
36
37
38
```

Przykład

Użycie

```
statek01 = Fregata(2001)
statek02 = Fregata(2002)
statek03 = Bryg(2010)
sts = Bryg(1992)
cutty_sark = Fregata(1869)
# Utworzono obiekt <__main__.Fregata object at
0x7f22a2ffac50>.
# Utworzono obiekt <__main__.Fregata object at
0x7f22a410dba8>.
# Utworzono obiekt <__main__.Bryg object at 0x7f22a410dc18>.
# Utworzono obiekt <__main__.Bryg object at 0x7f22a2ffadd8>.
# Utworzono obiekt <__main__.Fregata object at
0x7f22a2ffad30>.
spis_statkow = [statek01, sts, statek02, statek03, cutty_sark]
```

Przykład

Użycie

```
for statek in spis_statkow:
   print("------------------------------")
   statek.test()
   print("-----")
   statek.info()

# ----------------------------
# To jest wywołanie test-Fregata (rok_wodowania + 40)
# wynik = 2041
# -----
# Metoda w klasie Fregata
# Obiekt <__main__.Fregata object at 0x7f22a2ffac50> - typ: Fregata/3 maszty
# Rok wodowania = 2001
# ----------------------------
# To jest wywołanie test-Bryg (rok_wodowania + 20)
# wynik = 2012
# -----
# Metoda w klasie Bryg
# Obiekt <__main__.Bryg object at 0x7f22a2ffada0> - typ: Bryg/2 maszty
# Rok wodowania = 1992
```

Przykład

Użycie

```
for statek in spis_statkow:
   statek.nowy_rok()
for statek in spis_statkow:
   statek.info()

# Metoda w klasie Fregata
# Obiekt <__main__.Fregata object at 0x7f22a2ffac50> - typ: Fregata/3 maszty
# Rok wodowania = 2002
# Metoda w klasie Bryg
# Obiekt <__main__.Bryg object at 0x7f22a2ffada0> - typ: Bryg/2 maszty
# Rok wodowania = 1993
# Metoda w klasie Fregata
# Obiekt <__main__.Fregata object at 0x7f22a410dba8> - typ: Fregata/3 maszty
# Rok wodowania = 2003
# Metoda w klasie Bryg
# Obiekt <__main__.Bryg object at 0x7f22a410dc18> - typ: Bryg/2 maszty
# Rok wodowania = 2011
# Metoda w klasie Fregata
# Obiekt <__main__.Fregata object at 0x7f22a2ffad30> - typ: Fregata/3 maszty
# Rok wodowania = 1870
```

Sprawdzanie czy obiekt jest danej klasy

Użycie

```
for statek in spis_statkow:
   print(f"isinstance({statek}, Bryg) = {isinstance(statek, Bryg)}")
   print(f"isinstance({statek}, Fregata) = {isinstance(statek, Fregata)}")
   print(f"isinstance({statek}, Statek) = {isinstance(statek, Statek)}")
   print("----------")


# isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, Bryg) = False
# isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, Fregata) = True
# isinstance(<__main__.Fregata object at 0x7f22a2ffac50>, Statek) = True
# ----------
# isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Bryg) = True
# isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Fregata) = False
# isinstance(<__main__.Bryg object at 0x7f22a2ffada0>, Statek) = True
# ----------
# isinstance(<__main__.Fregata object at 0x7f22a410dba8>, Bryg) = False
# isinstance(<__main__.Fregata object at 0x7f22a410dba8>, Fregata) = True
# isinstance(<__main__.Fregata object at 0x7f22a410dba8>, Statek) = True
# ----------
# isinstance(<__main__.Bryg object at 0x7f22a410dc18>, Bryg) = True
```