



Wprowadzenie do OOP w Pythonie

Programowanie zorientowane obiektowo to potężne narzędzie w rękach programisty. Zastosowanie klas zbliża program do naturalnego opisu obiektów, ich właściwości i zachowań. To sprawia, że nasz kod może opisywać i rozwiązywać problemy w sposób czytelny, zrozumiały. Klasy więc, choć początkowo mogą wydawać się trudne, znacznie ułatwiają programiście realizację jego zadań.

12+

DOWIESZ SIĘ

-  Czym jest klasa, czym są jej instancje, argumenty i metody.
-  Jak tworzyć własne typy obiektów oraz poznasz przeznaczenie specjalnych metod takich jak „__init__” i „__str__”.

POTRZEBNA WIEDZA

-  Znajomość podstawowych struktur, typów danych, składni Pythona.
-  Umiejętność definiowania funkcji w Pythonie.

KLASA I OBIEKT KLASY

Zanim przejdziemy do samego Pythona, zastanówmy się przez chwilę, czym są pojęcia takie jak klasa i obiekt. Obiekt to zazwyczaj jakaś rzecz, miejsce czy postać. Otaczają nas zewsząd obiekty różnego typu. Śpimy zazwyczaj na łóżku, na prześcieradle, przykryci kołdrą, z głową na poduszce. Śniadanie jemy przy stole, siedząc na krześle. Korzystamy z talerza, kubka, noża, widelca. Nasz posiłek również składa się z różnych składników. Każda z tych rzeczy jest osobnego typu, ale możemy je łączyć w bardziej ogólne typy – na przykład prześcieradło, kołdra i poduszki to pościel. Łóżko, stół i krzesła to meble. Nóż i widelec to sztućce. Kubek i talerz to naczynia. Widzimy więc, że rzeczy możemy grupować w różne typy – mniej lub bardziej szczegółowo określające dany obiekt.

Weźmy na przykład taki obiekt jak łyżka. Mogą to być na przykład **stalowe łyżeczki do herbaty**. Mogą to być też **aluminiowe łyżki do zupy**. Obie grupy możemy ogólniej opisać po prostu jako łyżki, a nawet zaliczyć do bardziej ogólnego typu, jakim będą **sztućce**. Bez względu

na to, jaką nazwę wybraliśmy, nadal opisuje ona bardzo dużą grupę rzeczy.

Nazwę **stalowa łyżeczka do herbaty** możemy zastosować do każdej stalowej łyżeczki do herbaty na świecie – bez względu na jej kształt, pochodzenie czy rozmiar. Może to być też któraś z tych łyżeczek do herbaty w twojej kuchni. Każda z nich jest obiektem tego typu. W świecie programowania te ogólne typy nazywamy często **klasami**. Konkretnie obiekty danego typu nazywamy natomiast **instancjami** danej klasy.

Proces uogólnienia, w którym przechodzimy od opisu konkretnych rzeczy do bardziej ogólnego i bardziej pojemnego pojęcia, nazywamy **abstrahowaniem**. Tak więc typ czy klasa pewnych obiektów jest pewną **abstrakcją**, czyli uogólnieniem konkretnych obiektów.

Możemy tworzyć o naszych klasach i obiektach pewne wyobrażenia. Na przykład łyżkę możemy opisać lub narysować. O pomoc w wizualizacji poprosiłem moją córkę Gabrysię (pozdrawiamy przy okazji całą klasę 3b ze Szkoły Podstawowej im. Olimpijczyków w Baczynie).



Ilustracja 1. łyżka – szkło Gabrieli Korzeniewskiej

Skoro jesteśmy przy szkole, to weźmy teraz jakiś bardziej szkolny przykład. Może to być dobrze znana z zajęć matematyki figura – kwadrat.

Istnieje bardzo dużo różnych kwadratów. Dobrze wiemy, że ich cechą wspólną jest to, że wszystkie boki kwadratu są równe i parami równoległe do siebie, zaś kąty wewnętrzne mają po 90°. Wynikają stąd mniej znane fakty, pozwalające zaliczyć kwadraty do bardziej ogólnych typów. Każdy kwadrat jest więc zarazem prostokątem, czworokątem, równoległobokiem i rombem. Z drugiej strony nie każdy prostokąt czy romb jest kwadratem.

Sytuację, w której jakieś obiekty mają cechy innych obiektów, możemy opisać mechanizmem **dziedziczenia**. To trochę podobnie jak z dziećmi, które mogą dziedziczyć jakieś cechy po swoich rodzicach czy innych przodkach.

TYPY W PYTHONIE

Możesz się spotkać czasem z określeniem, że Python w pełni wspiera paradygmat programowania obiektowego. Paradygmat to sposób, w jaki patrzymy na pewne zagadnienie, pewien model, styl, wzorzec postępowania. W programowaniu mamy różne takie podejścia. Programowanie obiektowe oparte jest o obiekty, które łączą ze sobą stan i zachowanie. Stan to jakieś dane opisujące dany obiekt, nazywamy je też często atrybutami. Zachowanie to na przykład jakaś procedura, funkcja powiązana z obiektem. Takie powiązane z obiektem funkcje nazywamy metodami. Metody mogą na przykład zwracać jakieś informacje w oparciu o stan obiektu, mogą też ten stan modyfikować. Na razie wydaje się to skomplikowane, ale w miarę poznawania nowego materiału stanie się to bardziej zrozumiałe.

W Pythonie wszystko jest obiektem. Są oczywiście różne elementy składni, słowa kluczowe języka, opera-

tory różnego rodzaju. Ale poza tym wszystko, co da się przypisać do jakiejś zmiennej, w Pythonie jest obiektem. Oznacza to, że wszystkie te nasze pythonowe obiekty są jakiegoś typu. Jaki to jest typ, możemy sprawdzić przy pomocy instrukcji `type`.

Listing 1. Sprawdzenie typów wartości i zmiennych

```
c = "Kwadraty"
d = [1, 2, 3]

def mojafunkcja():
    pass

print(type(1))
print(type(1.1))
print(type(c))
print(type(d))
print(type(print))
print(type(mojafunkcja))
print(type(mojafunkcja()))
print(type(type))
```

W Listingu 1 przy pomocy funkcji `type` sprawdzamy, jakiego typu są obiekty podane jako argumenty tej funkcji. Możemy je podać bezpośrednio – jako wartości – lub posługując się referencją, czyli zmienną. Ciekawym przykładem jest podanie jako argumentu zdefiniowanej przez nas funkcji oraz jej wywołania. Jak widzimy, wynik wywołania funkcji jest zupełnie innego typu niż sama funkcja. Na koniec sprawdzamy, jakiego typu jest sam `type`. Poniżej zobaczymy wynik działania kodu z Listingu 1.

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'builtin_function_or_method'>
<class 'function'>
<class 'NoneType'>
<class 'type'>
```

Możemy też dokonać sprawdzenia w drugą stronę. To znaczy sprawdzić, czy dany obiekt jest instancją jakiejś klasy. Inaczej mówiąc, czy jest to obiekt danego typu. Służy do tego metoda `isinstance`.

Listing 2. Sprawdzenie, czy liczba 1 jest typu „int” i „str”

```
print(isinstance(1, int))
print(isinstance(1, str))
```

Wynikiem będą wypisane odpowiednio True i False. Liczba 1 jest typu int, ale nie jest typu str. Czasem jednak może być tak, że dany obiekt będzie zarówno jednego, jak i drugiego typu. Na przykład wartość True:

Listing 3. „True” jest zarówno instancją „bool”, jak i „int”. Nie jest jednak instancją „float”

```
print(isinstance(True, bool))
print(isinstance(True, int))
print(isinstance(True, float))
```

Wartości wypisane przez ten kod to True, True i False. Typ bool jest więc pewnym podtypem typu int. I rzeczywiście, jeśli wywołamy pomoc dla typu bool, to zobaczymy:

Listing 4. Pomoc dla typu „bool”

```
>>> help(bool)
Help on class bool in module builtins:
class bool(int)
|   bool(x) -> bool
|
```

Zapis class bool(int) oznacza, że tworzona klasa bool dziedziczy po int. Jak to dokładniej działa, zobaczymy, gdy będziemy opisywać mechanizm dziedziczenia, ale już teraz pomoc Pythona potwierdziła nasze obserwacje dotyczące zależności między tymi typami.

Zerknijmy jeszcze z ciekawości do pomocy dla int:

Listing 5. Pomoc dla „int”

```
Help on class int in module builtins:
class int(object)
|   int([x]) -> integer
|   int(x, base=10) -> integer
```

Jak widzimy, nasz int też wydaje się po czymś dziedziczyć. Tym razem jest to object. Object to taki podstawowy typ, po którym dziedziczą wszystkie inne

w Pythonie. Tak, tak – możesz samodzielnie sprawdzić, że jeśli za x w poniższym wyrażeniu wstawisz dowolną wartość, to wynikiem zawsze będzie True:

```
isinstance(x, object)
```

Warto przypomnieć, że kiedy wcześniej omawialiśmy typy wbudowane, mówiliśmy o int, bool, float, complex, str, tuple, list, dict, set jako o funkcjach. Pomoc Pythona wskazuje, że to jednak nie są funkcje, a klasy określające typy. Dodając do nich nawiasy () oznaczające wywołanie, można je jednak potraktować tak, jakby to były funkcje.

TWORZENIE WŁASNYCH KLAS

W pomocy widzieliśmy już, jak możemy zdefiniować klasę. Zobaczmy jednak, jak to wygląda w praktyce:

Listing 6. Zdefiniowanie klasy

```
>>> class Kwadrat:
>>>     pass
```

Klasę tworzymy więc, używając słowa kluczowego class. Po nim następuje nazwa klasy. Z reguły nazwa ta jest pisana wielką literą. Jeśli składa się z większej liczby słów, to słowa są ze sobą bezpośrednio połączone i każde zaczyna się wielką literą, na przykład: Stalowałyżka.

Jest to przyjęta w Pythonie konwencja nazewnicza znana jako PascalCase. Zastanawiasz się, dlaczego takie typy jak int czy bool są zapisane jednak małymi literami? Zapewne dlatego, że powstały dużo wcześniej, zanim zdecydowano się na stosowanie tej konwencji.

Jeśli zechcemy wypisać typ naszej klasy, to dowiemy się, że jest to type:

Listing 7. Sprawdzenie typu klasy

```
>>> print(type(Kwadrat))
<class 'type'>
```

Taki sam wynik dostaniemy dla innych znanych już typów (int, float itd.).

Zobaczmy teraz, jak możemy utworzyć obiekt naszej klasy i jaki będzie on miał typ. Obiekty tworzymy poprzez użycie nazwy klasy i dodanie do niej nawiasów oznaczających wywołanie (). Jeśli zechcemy wywołać

coś z argumentami, to między tymi nawiasami podamy jakieś argumenty:

Listing 8. Utworzenie instancji klasy „Kwadrat” i sprawdzenie typu tej instancji

```
>>> kw = Kwadrat()
>>> print(type(kw))
<class '__main__.Kwadrat'>
```

Warto zapamiętać: jeśli utworzymy klasę, to sama utworzona przez nas klasa jest obiektem typu `type`. Zaś obiekty nowo utworzonej klasy są po prostu obiektami tej klasy.

CECHA – CZYLI ATRYBUT

Wróćmy na chwilę do naszych rozważań teoretycznych. Kiedy opisujemy jakieś obiekty, to przeważnie opisujemy jakieś ich cechy. Powiedzmy, że chciałbym opisać konkretnego człowieka. Mogę wtedy powiedzieć, że na przykład jest wysoki albo niski. Mogę nawet dokładnie określić jego wzrost w centymetrach. Inną cechą może być kolor oczu, włosów. Gdybym miał powiedzieć, ile serc czy mózgów ma człowiek, to śmiało mógłbym powiedzieć, że po jednym. Jest to cecha wspólna dla wszystkich obiektów typu człowiek. Nie trzeba więc jej wpisywać do dokumentów opisujących konkretną osobę.

ATRYBUT KLASY W PYTHONIE

Atrybuty klasowe to po prostu zmienne zdefiniowane w ciele klasy. W ciele klasy, to znaczy, że kod przesunięty



CIEKAWOSTKA

Programowanie obiektowe po raz pierwszy pojawiło się w świecie informatyki w 1962 roku wraz z językiem SIMULA 1. Koncepcja ta rozwinięta została w SIMULA 67 wydanym w 1967 roku.

Na dobre jednak programowanie obiektowe pojawiło się w informatyce w latach 90. XX wieku, kiedy to zaczął rozwijać się język C++, który nadal jest ważnym i popularnym językiem programowania.

jest o jeden poziom względem słowa kluczowego `class` w prawo. Zupełnie tak jak wcześniej w funkcji czy w pętlach. Skoro to nazwy zmiennych, to mają zastosowanie do nich wszystkie reguły, które obowiązują dla zwykłych zmiennych.

Po atrybut sięgamy poprzez kropkę następującą po obiekcie. Po atrybut klasowy możemy sięgnąć poprzez nazwę klasy oraz poprzez instancję danej klasy.

Atrybuty klasowe dzielone są przez wszystkie instancje danej klasy. Oznacza to, że jeśli zmienimy wartość dla jakiegoś atrybutu, to zmieni się ona także dla wszystkich instancji klasy. Wszystkie te operacje widać w Listingu 9.

Listing 9. Definicja klasy „Czlowiek”. Atrybuty klasowe są wspólne dla wszystkich instancji

```
class Czlowiek:
    liczba_serc = 1
    liczba_mozgow = 1

print(Czlowiek.liczba_mozgow)

wojtek = Czlowiek()
natalia = Czlowiek()

print(wojtek.liczba_mozgow)
print(natalia.liczba_serc)

# wynik to
# 1
# 1
# 1

Czlowiek.liczba_mozgow = 2

print(wojtek.liczba_mozgow)
print(natalia.liczba_mozgow)

# wynik to
# 2
# 2
```

ATRYBUT INSTANCJI W PYTHONIE

Kiedy chcemy ustawić jakiś atrybut, który charakteryzuje konkretną instancję, to możemy to także zrobić poprzez kropkę dla istniejącego obiektu:

```
obiekt.atrybut = wartosc
```

Jeśli użyjemy nazwy, która jest już atrybutem klasowym, to utworzymy atrybut instancji o tej nazwie. Przesłoni on atrybut klasowy. Od teraz, sięgając po tę cechę w zmienionym obiekcie, będziemy pobierać atrybut instancji. Atrybut klasowy nie będzie jednak zmieniony i nadal będzie obowiązywał dla pozostałych instancji – tak jak w Listingu 10.

Listing 10. Ustawienie atrybutu dla instancji i próba wybrania nieistniejącego atrybutu

```
wojtek.rok_ur = 2012
wojtek.liczba_mozgow = 1
print(natalia.liczba_mozgow)
print(natalia.rok_ur)
```

Odwołanie się do nieistniejącego atrybutu da nam błąd `AttributeError`. Widzimy to w wyniku wywołania kodu z Listingu 10:

```
Traceback (most recent call last):
  File "/PRJ11/main.py", line 64, in <module>
    natalia.rok_ur
AttributeError: 'Czlowiek' object has no
attribute 'rok_ur'
```

METODA

O ile zmienne umieszczone w ciele klasy nazywamy atrybutami, o tyle funkcje, które tam umieścimy, nazywamy metodami. Mamy trzy rodzaje metod. Pierwszy rodzaj to metoda instancji. Dwa pozostałe to metoda klasowa i metoda statyczna. Nimi nie będziemy się dziś zajmować. Metoda instancji to funkcja wewnątrz klasy, której pierwszym i wymaganym parametrem jest zawsze instancja. Zwyczajowo nazywamy ten parametr `self`. Spójrzmy w Listing 11. Tworzymy tam klasę, która ma atrybut klasowy o nazwie `atrybut`. Definiujemy w niej też metodę `moja_metoda`. Tworzymy dalej instancję i wywołujemy metodę. Wywołujemy, czyli wybieramy z obiektu poprzez kropkę i dodajemy nawiasy.

Listing 11. Metoda instancji w klasie

```
class MojaKlasa:

    atrybut = "PRJ"
```

```
def moja_metoda(self):
    print("Moja metoda")
    print(self.atribut)
```

```
instancja = MojaKlasa()
instancja.moja_metoda()
```

Zwróćmy uwagę, że przy wywoływaniu metody `moja_metoda` z instancji naszej klasy nie przekazujemy w nawiasach argumentu, chociaż według definicji jest on potrzebny. Przy takim wywołaniu Python zadba o to, by przekazać do tej funkcji pierwszy argument. Innymi słowy obiekt, na który wskazuje instancja, staje się tym `self` w funkcji.

Możemy skorzystać z metody instancji także poprzez klasę. Ale wtedy już musimy jawnie podać w wywołaniu instancję jako argument. Tak jak w poniższym kodzie:

```
MojaKlasa.moja_metoda(instancja)
```

W obu przypadkach wynikiem będzie:

```
Moja metoda
PRJ
```

METODA `__INIT__`

Ustawianie atrybutów instancji sposobem przedstawionym w Listingu 10 może być bardzo niewygodne. Łatwo o pomyłkę. Często też chcemy mieć pewność, że instancja będzie mieć potrzebne dla innych metod atrybuty. Wróćmy do przykładu z kwadratami. Zdefiniujemy klasę `Kwadrat` w taki sposób, by konieczne było podanie długości boku w czasie tworzenia instancji. Do tych zadań służy metoda `__init__`. Jest ona wywoływana przez mechanizmy wbudowane w interpreter języka jednorożkowo, w momencie tworzenia nowego obiektu danego typu. Metoda ta, jak każda inna, może przyjmować argumenty, które możemy wykorzystać m.in. do ustawienia atrybutów instancji. Ogólnie dobrą praktyką jest inicjowanie tutaj wszelkich atrybutów, do których odwołują się metody zdefiniowane w naszej klasie. W Listingu 12 jest to `self.bok`.

Listing 12. Metoda „__init__” pozwala na ustawienie atrybutów tworzonego obiektu

```
class Kwadrat:

    def __init__(self, bok):
        self.bok = bok

    def obwod(self):
        return self.bok * 4

kw_a = Kwadrat(1)
kw_b = Kwadrat(8)

print(kw_a.obwod())
print(kw_b.obwod())
```

METODA __STR__

Co się stanie, gdy spróbuję wyprintować instancję klasy Kwadrat?

print(kw_a) da nam wynik podobny do tego:

```
<__main__.Kwadrat object at 0x10eb8ff70>
```

Z tego zapisu dowiadujemy się jedynie, że mamy do czynienia z obiektem klasy Kwadrat z modułu __main__.

Trudno byłoby jednak odróżnić ten kwadrat od innego. Możemy poprawić trochę sytuację, dodając do naszej klasy definicję metody __str__. Powinna ona zwracać napis, który będzie reprezentował naszą klasę (Listing 13).

Listing 13. Metoda „__str__” odpowiada za zachowanie instancji w sytuacji zamieniania jej na napis

```
class Kwadrat:

    def __init__(self, bok):
        self.bok = bok

    def __str__(self):
        return f"<Kwadrat, bok: {self.bok}>"

    def obwod(self):
        return self.bok * 4
```

Tym razem print(kw_a) da nam napis <Kwadrat, bok: 1>.

W kolejnych artykułach poznamy klasy dokładniej. Nauczmy się stosować dziedziczenie. Powiemy instancjom naszych klas, w jaki sposób mogą oddziaływać ze sobą przy pomocy różnych operatorów.



Rafał Korzeniewski

Z wykształcenia muzyk-puzonista i fizyk.
Z zamiłowania i zawodu Pythonista. Trener Pythona,
współorganizator PyWaw (<http://pywaw.org>)
– warszawskiego meetupu poświęconego Pythonowi.
W wolnych chwilach uczy się gry na nowych instrumentach, udziela się społecznie i dużo czyta.





KORZENIEWSKI@GMAIL.COM



ZAPAMIĘTAJ

-  Python jest językiem, który w pełni wspiera paradygmat programowania obiektowego. Możemy tworzyć w nim własne typy, nazywane klasami. Możemy później wykorzystywać te klasy do tworzenia instancji (obiektów). Klasy i instancje mogą mieć swoje atrybuty oraz metody.
-  Dzięki klasom możemy więc tworzyć obiekty, które nie tylko oferują jakieś zachowania, czyli metody, ale które potrafią też przechowywać i modyfikować stan obiektu

ĆWICZ W DOMU

-  Zdefiniuj klasy dla innych figur geometrycznych.
-  Dodaj do nich metody, które policzą ich obwody i pola powierzchni.
-  Utwórz instancje tych klas i sprawdź, czy twoje metody działają poprawnie.
-  Spróbuj zdefiniować klasy, które będą opisywały inne rzeczy wokół ciebie.