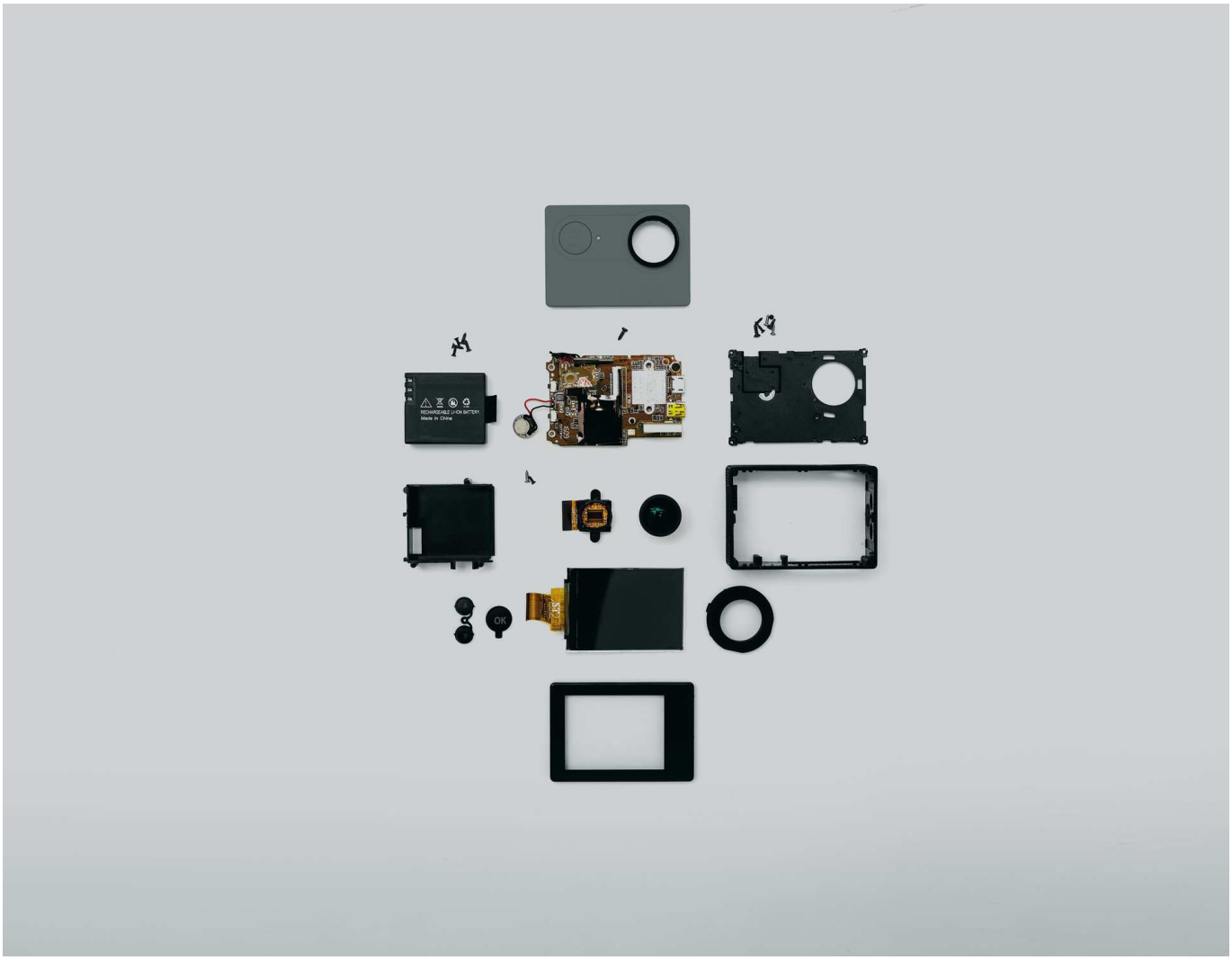

Wstęp do programowania obiektowego w języku Python

- Wprowadzenie
- Przeczytaj
- Film samouczek
- Sprawdź się
- Dla nauczyciela

Źródło: Alexander Andrews, domena publiczna.

Programowanie obiektowe różni się od programowania strukturalnego tym, że podstawową strukturą danych, z którą pracuje programista, jest obiekt. W omawianym podejściu obiekty posiadają własne metody oraz atrybuty, z których mogą korzystać. Programowanie obiektowe ma za zadanie ułatwić pisanie kodu, sprawić, by był czytelniejszy oraz usprawnić jego edycję.

Podstawowe informacje na ten temat znajdziesz w e-materiale Wstęp do programowania obiektowego.

W tym e-materiale poznamy specyfikę programowania zorientowanego obiektowo w języku Python.

Jeśli chcesz dowiedzieć się, jak to zagadnienie wygląda w przypadku innych języków programowania, sięgnij do e-materiałów:

- Wstęp do programowania obiektowego w języku C++,
- Wstęp do programowania obiektowego w języku Java.

Twoje cele

- Prześledzisz, czym są klasy i obiekty oraz na czym polega dziedziczenie.
- Opiszesz właściwości podstawowej klasy.

- Przeanalizujesz i zdefiniujesz klasy zgodne z zasadami programowania w języku Python.
- Użyjesz elementów programowania obiektowego w praktyce.

Przeczytaj

W języku Python programowanie zorientowane obiektowo jest opcjonalne. Pozwala ono efektywnie zarządzać kodem programu.

Obiekt, w potocznym rozumieniu, to najczęściej realna rzecz, osoba, zwierzę. Wyobraźmy sobie zatem psa. Ma kolor, wielkość, wagę. Potrafi szczekać, spać, jeść. Często uczymy go różnych komend: „podaj łapę", „zostaw", „aport". Spróbujemy przenieść przykład psa w świat programowania.

Zaczniemy od utworzenia klasy, czyli od opisu ogólnego psa oraz jego możliwych zachowań. W języku Python zapisujemy to w następujący sposób:

```
class Pies:
    pass
```

W tej definicji możemy wyodrębnić:

- słowo kluczowe `class` – oznacza definicję nowej klasy,
- nazwę klasy `Pies` – według dokumentu PEP 8 nazwy klas zapisujemy wielką literą,
- słowo kluczowe `pass` oznaczające instrukcję: nic nie robimy, jej wykonanie nie powoduje żadnych skutków; używamy go, gdy składnia wymaga instrukcji, a my nie chcemy, aby program cokolwiek wykonywał.

Przykład 1

Utworzymy obiekt klasy `Pies` o nazwie `pierwszy_psiak`. Za pomocą funkcji `type()` sprawdzimy, jakiego typu jest utworzony obiekt oraz jakie elementy zawiera.

```
class Pies:
    pass

pierwszy_psiak = Pies()

print(type(pierwszy_psiak))
# <class '__main__.Pies'>
```

Typ obiektu `pierwszy_psiak` to klasa `__main__.Pies`.

Ważne!

> Pola klasy mają tą samą wartość dla wszystkich obiektów takiej klasy, a pola obiektu – a więc instancji klasy – są unikalne dla każdego obiektu.

W następnym kroku stworzymy podstawowy opis psa. Dla tego obiektu będziemy chcieli zapamiętać następujące pola:

- rasa psa,
- imię psa.

Dla klasy `Pies` ustalimy pole `typ` o wartości `czworonóg`. Ponadto zdefiniujemy funkcję o nazwie: `podaj_lape()`, która będzie wypisywać na ekranie komunikat, korzystając z danych zawartych w klasie, oraz funkcję `opis_zwierzaka()` informującą o typie i rasie.

Zdefiniujmy podstawową klasę, używając wskazówek typów do wskazania typów danych. Pola obiektów inicjalizowane będą przez funkcję **`init`**`()`. Omówienie tej funkcji znajdziesz w e-materiale Konstruktory i destruktory w języku Python

```
1  class Pies:
2      """klasa opisująca psa"""
3
4      # zmienna klasowa współdzielona przez wszystkie instancje kla
5      typ: str = "czworonóg"
6
7      def __init__(self):
8          self.imie: str = ""
9          self.rasa: str = ""
10
11     def podaj_lape(self):
12         print(f"Zwierzak {self.imie} podaje łapę.")
13
14     def opis_zwierzaka(self):
15         print(f"Zwierzak typu {self.typ} - {self.imie}, rasa: {se
```

> **Ważne!**
>
> Bardzo ważnym elementem jest słowo kluczowe `self` – reprezentuje ono instancję klasy. Gdy go użyjemy, obiekt będzie wiedział, że odnosimy się do jego pola bądź metody.
> **Przykład 2**
>
> Przykładowe utworzenie obiektu klasy `Pies`:
>
> ```
> 1 pies = Pies()
> ```

```
2
3 # przypisujemy konkretne wartości do pól instancji klasy
4 pies.rasa = 'kundelek'
5 pies.imie = 'Reksio'
```

Przykład 3

Możemy zdefiniować inne obiekty klasy `Pies` i przypisać ich polom różne wartości.

```
1  # definicja nowych obiektów klasy Pies
2  pies_a = Pies()
3  pies_a.rasa = "kundelek"
4  pies_a.imie = "Reksio"
5
6  pies_b = Pies()
7  pies_b.rasa = "kundelek"
8  pies_b.imie = "Azor"
9
10 pies_c = Pies()
11 pies_c.rasa = "wilczur"
12 pies_c.imie = "Brutus"
```

Każdy z obiektów klasy ma pola klasy o różnych wartościach, a metody `podaj_lape()` oraz `opis_zwierzaka()` działają dla każdego obiektu w następujący sposób, tj. poszczególne obiekty w wyświetlanych komunikatach będą wykorzystywały swoje zmienne wewnętrzne, podane im wcześniej przy inicjalizacji obiektów.

```
1  pies_a.opis_zwierzaka()
2  # Zwierzak typu czworonóg - Reksio, rasa: kundelek
3  pies_a.podaj_lape()
4  # Zwierzak Reksio podaje łapę.
5
6  pies_b.opis_zwierzaka()
7  # Zwierzak typu czworonóg - Azor, rasa: kundelek
8  pies_b.podaj_lape()
9  # Zwierzak Azor podaje łapę.
10
11 pies_c.opis_zwierzaka()
12 # Zwierzak typu czworonóg - Brutus, rasa: wilczur
13 pies_c.podaj_lape()
14 # Zwierzak Brutus podaje łapę.
```

Łatwo również wykazać, że obiekty klasy są różnymi obiektami. Python ma funkcję `id()`, pozwalającą pokazać unikatowy identyfikator zmiennej.

```
# przykładowe wyniki wywołania funkcji id()
# jeśli spróbujesz wywołać kod samodzielnie,
# najprawdopodbniej otrzymasz inne wyniki,
# ponieważ Python przyporządkuje obiektom inne
# adresy w pamięci
print(id(pies_a))
# 140279558901432

print(id(pies_b))
# 140279558899752

print(id(pies_c))
# 140279558901712
```

Ważne!

W programowaniu obiektowym dziedziczenie jest mechanizmem pozwalającym przekazać do klas, że cześć ich cech oraz metod jest zdefiniowana w innej klasie, a one same są potomkami tej klasy. Dzięki dziedziczeniu, tworząc nową klasę opisującą obiekty podobne do opisywanych przez klasę `Pies`, nie będziemy musieli ponownie definiować funkcji `podaj_lape()` i `opis_zwierzaka()`, ani pola `typ`. Będą one automatycznie przeniesione do nowej klasy.

Przyjmijmy, że oprócz ogólnych obiektów typu `Pies` chcemy też opisywać specyficzne obiekty `Hiena`. Muszą one zawierać dodatkowe pola:

- `rasa: str` – musi mieć zawsze wartość `hiena`,
- `rejon_występowania: str`.

Przykład 4

Zdefiniujmy klasę `Hiena`, która będzie zawierała pola i funkcje klasy `Pies` wzbogacone o pewien nowy element. Pozwoli ona na przechowywanie informacji specyficznych dla hieny, przy czym niektóre elementy będą wspólne – dokładnie te, które klasa `Hiena` odziedziczyła po klasie `Pies`.

```
class Hiena(Pies):
    """klasa opisująca hienę - bazuje na klasie Pies"""
    # przypisujemy wszystkim instancjom klasy Hiena wartość zmi
```

```
4
5       def __init__(self):
6           self.imie: str = ""
7           self.rasa: str = "hiena"
8           self.rejon_wystepowania: str = ""
9
10      # nadpisujemy metodę, aby wyswietlała też rejon występowani
11      def opis_zwierzaka(self):
12          print(f"Zwierzak typu {self.typ} - {self.imie}, rasa: {
```

Możemy zdefiniować kilka obiektów i sprawdzić, jak działają ich metody.

```
1  # definicja nowych obiektów klasy Hiena
2  hiena_a = Hiena()
3  hiena_a.imie = "Kunegunda"
4  hiena_a.rejon_wystepowania = "Polska"
5
6  hiena_b = Hiena()
7  hiena_b.imie = "Topaz"
8  hiena_b.rejon_wystepowania = "Hiszpania"
9
10 # próba wywołania metod zdefiniowanych w klasie Pies
11 hiena_a.opis_zwierzaka()
12 # Zwierzak typu czworonóg - Kunegunda, rasa: hiena, rejon wyste
13 hiena_a.podaj_lape()
14 # Zwierzak Kunegunda podaje łapę.
15
16 hiena_b.opis_zwierzaka()
17 # Zwierzak typu czworonóg - Topaz, rasa: hiena, rejon wystepowa
18 hiena_b.podaj_lape()
19 # Zwierzak Topaz podaje łapę.
```

Ważne!

Metoda `podaj_lape()` nie była jawnie deklarowana w klasie `Hiena`. Została **odziedziczona** z klasy `Pies`.

Polecenie 1

Spróbuj zdefiniować klasę o nazwie `Uczen`, która będzie miała następujące pola instancji klasy, inicjalizowane przez funkcję `__init__()`. Omówienie tej funkcji znajdziesz w e-materiale Konstruktory i destruktory w języku Python:

- imię – łańcuch znaków,
- nazwisko – łańcuch znaków,
- email – łańcuch znaków,
- oceny z informatyki – lista,
- oceny z plastyki – lista,
- oceny z języka polskiego – lista.

Dla zainteresowanych

Pola klasy (nie mylić z polami instancji klasy) powinny być niezmienne (immmutable). Użycie typów zmiennych (mutable) może spowodować nieoczekiwane działanie obiektów klas. Oto definicja klasy ze zmiennymi i niezmiennymi polami:

```
class Samochod:
    # Produkujemy tylko samochody jednej marki i nie potrzebuje
    marka: str = "Polonez"

    # Niepoprawne zdefiniowanie pola klasy, ponieważ lista jest
    # każda instancja klasy będzie mogła ją modyfikować
    wyposazenie: list = ["Kierownica", "Lusterka", "Radio"]

    # W takim przypadku lepiej by było użyć krotki, która jest
    wyposazenie: tuple = ("Kierownica", "Lusterka", "Radio")
```

Już wiesz

- W języku Python wszystko jest obiektem.

- Klasy w języku Python mogą dziedziczyć pola klasy oraz metody po swojej klasie bazowej, jeżeli jest taka podana w definicji klasy.
- Programowanie obiektowe pozwala efektywnie zarządzać kodem programu.
- Obiekt to konkretny element klasy.
- Każdy obiekt zawiera wszystkie pola klasy i jej metody.

Słownik

identyfikator

(ang. *identity*) – liczba naturalna, która jest unikatowa i stała dla każdego obiektu w przestrzeni nazw języka Python w trakcie swojego istnienia; w implementacji CPython jest to adres pamięci RAM, w którym przechowywany jest obiekt

klasa

opis (definicja) tworzonych na jej podstawie obiektów; składa się z pól i metod; pola (atrybuty) przechowują wartości, które są cechami obiektu, z kolei metody pozwalają na manipulacje tymi wartościami

niezmienna

(ang. *immutable*) sekwencja, która jest niemodyfikowalna „w miejscu", a więc nie można zmienić żadnego z jej elementów wewnątrz, trzeba stworzyć nowy obiekt, który zawiera zmienione elementy; w języku Python większość wbudowanych typów jest niezmienna, np. liczby, krotki, łańcuch znaków

obiekt

instancja klasy; powstaje w wyniku utworzenia zmiennej typu obiektowego (danej klasy); obiekt jest zbudowany z pól zadeklarowanych w klasie; do zmian wartości pól możemy wykorzystać zaimplementowane w klasie metody; obiekty możemy jednoznacznie identyfikować

PEP 8

Style Guide for Python Code – dokument stworzony przez Guido van Rossum, opisujący konwencje kodowania dla języka Python, które powinny stanowić drogowskaz dla programistów

programowanie zorientowane obiektowo

(ang. *OOP – Object Oriented Programming*) – paradygmat programowania; konwencja wytwarzania oprogramowania, zalecająca traktowanie programu komputerowego jako współpracujących ze sobą abstrakcyjnych obiektów, posiadających pola/atrybuty (określające stan) oraz metody (definiujące zachowania); obiekty stworzone według klas współpracują ze sobą w celu wykonywania zadań

wskazówki typów

(ang. *type hints*) informacja o typie danego obiektu zapisana literalnie w kodzie programu (te podpowiedzi w żaden sposób nie wpływają na działanie programu; z uwagi na dynamiczną naturę języka Python), pomaga ona w zrozumieniu kodu programu i pozwala na sprawdzanie kodu przez różne programy IDE, np. PyCharm czy SublimeText lub Atom; przykład użycia takiego zapisu: `zmienna: int = 10;` *type hints* są dokładnie opisane w dokumencie PEP 484, inną nazwą są *annotations* opisane w dokumencie PEP 3107

zmienna

(ang. *mutable*) sekwencja, która jest modyfikowalna „w miejscu", a więc można zmienić dowolny z jej elementów wewnątrz, bez konieczności przypisywania nowych wartości do kolejnego obiektu; w języku Python zmienne są np. listy, słowniki, zbiory

Film samouczek

Problem 1

Zaimplementuj klasy `Pracownik` i `Szef`. Klasa, w której będziemy przechowywać dane o pracownikach, powinna zawierać pola `imie` i `wyplata` oraz metodę `wyswietl_wyplate_pracownika(imie: str)`, która wyświetli, ile zarabia pracownik o imieniu `imie`. Klasa `Szef` powinna dziedziczyć po klasie `Pracownik` i zawierać pole `premia` i metodę `policz_wynagrodzenie()`, która oblicza i wyświetla, ile zarabia szef po dodaniu premii.

Specyfikacja problemu:

Dane:

- pola klasy `Pracownik`:
    - `imie` – łańcuch znaków
    - `wyplata` – liczba rzeczywista
- pola klasy `Szef`:
    - `imie` – łańcuch znaków
    - `wyplata` – liczba rzeczywista
    - `premia` – liczba rzeczywista

Wynik:

- utworzone obiekty klas posiadające zadane wartości

Swój program przetestuj dla przedstawionych danych:

- pracownik Janek – 1500 zł wypłaty,
- pracownik Paweł,

- szef Kuba – wypłata 3000 zł, premia 1500 zł.

```
1
```

```
1
```

Polecenie 1

Porównaj swoje rozwiązanie z tym przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/Rgu2GYMQfhlq9

Film przedstawia elementy projektowania obiektowego w języku Python.

Pobierz plik z kodem źródłowym:

Plik o rozmiarze 643.00 B w języku polskim

> **Polecenie 2**
>
> Przeanalizuj, jak tworzona jest klasa i jej obiekt. Zwróć uwagę na działanie wbudowanej funkcji `id()`.

1

Będziemy korzystać z klasy `Komputer` wraz z metodą `informacje()`.

Klasa `Komputer` jest klasą reprezentującą komputer. Ma metody i atrybuty, które opisują parametry danego komputera.

Metoda `informacje()` wyświetla informacje na temat konkretnego obiektu klasy `Komputer`, na którym została wywołana. W wyświetlanych informacjach są między innymi: identyfikator

obiektu (`id(self)`), nazwa (`self.nazwa`), procesor (`self.procesor`), system operacyjny (`self.os`), moc obliczeniowa (`self.gigaflops`) i wydajność (`self.bogomips`).

Klasa `Komputer` nie ma konstruktora, więc w momencie tworzenia obiektu nie ma możliwości przypisania wartości początkowych do jego atrybutów. W przypadku chęci zmiany wartości atrybutów, można to zrobić po stworzeniu obiektu, korzystając z notacji kropkowej: `obiekt.atrybut = nowa_wartosc`.

```
1 class Komputer:
2     """Klasa opisująca
  komputer z parametrami"""
3
4     def informacje(self):
5         """funkcja
  działająca na danych
  obiektu"""
6         print(f"Komputer
  id={id(self)} nazywa się:
  {self.nazwa}.")
7         print(f"Ma
  procesor {self.procesor} i
  system operacyjny
  {self.os}.")
8         print(f"Jego moc
  obliczeniowa to {self.
  Gigaflops} GFLOPS.")
9         print(f"Jego
  wydajność to
  {self.bogomips}
  bogomips.")
```

2

Tworzymy dwa nowe obiekty klasy `Komputer`.

Nowe obiekty klasy `Komputer` tworzymy poprzez wywołanie konstruktora klasy.

Konstruktor ten, który w klasie `Komputer` nie został zdefiniowany jawnie, jest dziedziczony z klasy bazowej `object` i nie przyjmuje żadnych argumentów. W wyniku wywołania konstruktora tworzony jest nowy obiekt, którego stan początkowy jest ustalany na podstawie atrybutów klasy, które są zainicjowane wartościami domyślnymi, gdy takie zostały zdefiniowane, lub pustymi. Nowe obiekty są zapisywane w zmiennych `komp_1` i `komp_2`, dzięki czemu można się do nich odwoływać.

```
komp_1 = Komputer()
komp_2 = Komputer()
```

3

Przypisujemy różne wartości do pól instancji klasy.

Do pól instancji klasy zostały przypisane wartości poprzez odwołanie się do obiektu i wywołanie pola, np. `komp_1.nazwa = "Sharp MZ-700"`. W ten sposób można ustawić wartość dla każdego pola instancji klasy.

```
komp_1.nazwa = "Sharp MZ-700"
komp_1.procesor = "Fujitsu MB8843"
komp_1.os = "None - ROM"
komp_1.bogomips = 0
komp_1.gigaflops = 0

komp_2.nazwa = "Standard PC"
komp_2.procesor = "Xeon E3"
komp_2.os = "Linux Mint 19.3"
komp_2.bogomips = 5180
```

```
11 komp_2.gigaflops = 170
```

4

Sprawdzamy teraz wskazanie funkcji `id()` dla pierwszego obiektu (w różnych komputerach i systemach operacyjnych wartość ta będzie różna).

Wywołujemy funkcję `id()` z argumentem `komp_1`, który jest pierwszym obiektem klasy `Komputer`, i wyświetlamy zwróconą wartość za pomocą funkcji `print()`.

```
1 print(id(komp_1))
2 # 139797347314376
```

5

Sprawdzamy teraz wskazanie funkcji `id()` dla drugiego obiektu (w różnych komputerach i systemach operacyjnych wartość ta będzie różna).

Wynikiem wywołania funkcji `id(komp_1)` i `id(komp_2)` są różne liczby, co oznacza, że obiekty `komp_1` i `komp_2` są dwoma osobnymi instancjami klasy `Komputer` i zajmują one różne obszary pamięci w programie. W języku Python każdy obiekt ma unikalny identyfikator, który jest przypisany do niego w momencie jego utworzenia, a funkcja `id()` zwraca ten identyfikator. Możemy zatem stwierdzić, że każdy obiekt klasy `Komputer` jest osobnym, niezależnym od pozostałych obiektów bytem w pamięci, a dostęp do jego pól i metod jest możliwy poprzez nazwę zmiennej, w której jest on przechowywany.

```
1 print(id(komp_2))
```

```
# 139797347314768
```

6

Możemy sprawdzić, co zawierają pola instancji klasy. By to zrobić, wykonujemy instrukcję `print`, przekazując jej jako argument wartości poszczególnych atrybutów obiektów `komp_1` oraz `komp_2`.

```
print(komp_1.nazwa)
# Sharp MZ-700
print(komp_2.nazwa)
# Standard PC
print(komp_1.os)
# None - ROM
print(komp_2.os)
# Linux Mint 19.3
```

7

Wykonujemy metodę `informacje()` dla obiektu `komp_1`.

```
komp_1.informacje()
```

Efekt wywołania:

```
komp_1.informacje()
# Komputer id=139797347314376 nazywa się: Sharp MZ-700.
# Ma procesor Fujitsu MB8843 i system operacyjny None - ROM.
# Jego moc obliczeniowa to 0 GFLOPS.
# Jego wydajność to 0 bogomips.
```

8

Metodę `informacje()` wywołujemy dla drugiego obiektu.

```
komp_2.informacje()
```

Efekt wywołania:

```
komp_2.informacje()
# Komputer id=139797347314768 nazywa się: Standard PC.
# Ma procesor Xeon E3 i system operacyjny Linux Mint 19.3.
# Jego moc obliczeniowa to 170 GFLOPS.
# Jego wydajność to 5180 bogomips.
```

9

Do atrybutów obydwu obiektów przypisaliśmy wartości, a te zostały poprawnie wyświetlone.

Efekt wywołania wbudowanej funkcji `id()` dla każdego stworzonego obiektu jest inny, co oznacza, że obiekty są różnymi instancjami klasy `Komputer`.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Ćwiczenie 2

Ćwiczenie 3

Napisz program, w którym utworzysz klasę o nazwie Samochod, zawierającą metodę `wypisz_dane()`, wypisującą wartości pól: `marka`, `model`, `liczba drzwi` oraz `rokProdukcji`.

Następnie stwórz w programie klasę `Maluch`, która będzie dziedziczyła z klasy Samochod. Polom klasowym powinny być przypisane wartości: `marka = "Samochodex"`, `model = "maly"`, `liczba drzwi = 3`.

Swoje rozwiązanie przetestuj dla obiektu typu Samochod z następującymi wartościami: `Autopol` (marka), `rodzinny` (model), `2010` (rok produkcji), 5 (liczba drzwi) i instancji klasy `Maluch` z wartością 1978 (rok produkcji).

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Wstęp do programowania obiektowego w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

> II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.
>
> > Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:
> >
> > > 2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;
> > >
> > > 3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Prześledzisz, czym są klasy i obiekty oraz na czym polega dziedziczenie.
- Opiszesz właściwości podstawowej klasy.
- Przeanalizujesz i zdefiniujesz klasy zgodne z zasadami programowania w języku Python.

- Użyjesz elementów programowania obiektowego w praktyce.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wstęp do programowania obiektowego w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Przedstawienie tematu zajęć oraz wspólne z uczniami ustalenie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Na forum klasy uczniowie analizują przedstawione w niej rozwiązania przykładów 1, 2, 3 i 4. Następnie zapisują programy na swoich komputerach i testują je.

2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie analizują prezentację. Następnie rozwiązują problem 1 i porównują swoją odpowiedź z filmem.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 3, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".

Praca domowa:

1. Uczniowie wykonują polecenie 1 z sekcji „Przeczytaj".
2. Uczniowie wykonują ćwiczenia nr 1–2 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.