Programowanie Obiektowe

Mikołaj Storoniak

Czym jest obiekt? Z czego się składa?

- OBIEKT: Zbiór danych, traktowany jako całość na którym można wykonać pewien zbiór operacji
- KLASA: Opisuje jakąś kategorię obiektów (stanowi „szablon”). Obiekt należący do klasy jest nazywany jej instancją. Obiekt tworzymy, wywołując() klasę.

- Tożsamość: każde dwa obiekty są rozróżnialne
- Stan: wartości atrybutów obiektu
- Zachowanie: akcje (funkcje), które może można wykonać na obiekcie. Funkcje przypisane do obiektu nazywamy METODAMI

`objects1.py`

Self

- Jak obiekt odnosi się sam do siebie?
- Zwyczajowe rozwiązanie: słowo kluczowe self
- W Pythonie: jeśli wywołujemy metodę obiektu, to ten automatycznie podaje siebie jako pierwszy argument
- Zwyczajowo używamy słowa self (ale możemy innego)

`self.py`

Konstruktor

- Konstruktor to funkcja służąca do inicjalizacji obiektu
- Wywołuje się sam przy tworzeniu obiektu

- Stworzenie konstruktora: nadpisanie metody `__init__()`

`init.py`

Dziedziczenie

- Klasa może dziedziczyć po innej klasie – wówczas przejmuje wszystkie jej właściwości

`inherit.py`

- Dziedziczenie służy do rozszerzania danej klasy.
- Klasa potomna może nadpisać metodę rodzica.
- Dopuszczalne jest wielokrotne dziedziczenie, należy jednak na nie uważać – może prowadzić do niejednoznaczności.

`multipleInherit.py`

Dziedziczenie

- `isinstance(x,y)` – czy obiekt x należy do klasy y?
- `issubclass(x,y)` – czy klasa x dziedziczy po y?

- `super()` – pozwala wywołać metodę klasy „wyższej”

- Zadanie: sprawdzić

`inherit_2.py`

Kontrola dostępu

- Większość języków obiektowych pozwala na ustalenie, skąd można uzyskać dostęp do zmiennej
- Private, public, protected...
- Jak to działa w Pythonie?

Kontrola dostępu

- Zamiast słów kluczowych, Python używa _podkreśleń
- Bez podkreśleń – zmienna publiczna
- __dwa podkreślenia – zmienna prywatna
- _podkreślenie – zmienna chroniona

- Zadanie: napisać klasę, która zademonstruje działanie modyfikatorów dostępu

Kontrola dostępu

- Python nie obsługuje mechanizmu kontroli dostępu – wszystko jest de facto publiczne
- Obowiązuje jedynie konwencja: `_zmienne` i `_metody` są do użytku wewnętrznego
- Do `__zmiennych` i `__metod` też można uzyskać dostęp, ale Python zmienia ich nazwy na `obj._klasa__zmienna` (name mangling)
- To pozwala uniknąć problemów z kolizją nazw przy dziedziczeniu

`access.py`

Zmienne specjalne

- W Pythonie funkcjonuje zbiór zmiennych „systemowych”
- Oznaczono je jako __zmienna__ (podwójne podkreślenia)

Zmienne i metody specjalne - przykłady

- __name__ – nazwa aktualnego modułu
- __bases__ – krotka z klasami bazowymi danej klasy
- __dict__ – słownik nazw klasy/obiektu
- __class__ – nazwa klasy do której należy obiekt
- __sizeof__ – rozmiar obiektu (w bajtach)

- Istnieje więcej, ale od tego jest dokumentacja :)
- Zadanie: zademonstrować działanie __bases__, __dict__, __class__ i __sizeof__

`special.py`

Nadpisywanie metod

- Jak pokazano wcześniej, metody możemy nadpisywać przy dziedziczeniu
- Istnieje szereg metod specjalnych, których Python używa wewnętrznie i których nadpisanie zmienia zachowanie obiektu

Nadpisywanie metod - operatory

• `__add__(self, other)`	`x+y = x.__add__(y)`
• `__sub__(self,other)`	`x-y = x.__sub__(y)`
• `__mul__(self,other)`	`x*y = x.__mul__(y)`
• `__truediv__(self, other)`	`x/y = x.__truediv__(y)`
• `__pow__(self, other)`	`x**y = x.__pow__(y)`
• `__eq__(self,other)`	`x == y = x.__eq__(y)`

- Zadanie: napisać klasę które reprezentuje funkcję liniową.
- Parametry: a, b
- Możliwość dodania liczby i innej funkcji
- Możliwość mnożenia i dzielenia przez liczbę
- Metoda która zwraca wynik dla danego x
- Możliwość porównania z inną funkcją

Wywołanie obiektu?

- Ostatnio, kiedy próbowaliśmy czegoś takiego, Python wyrzucił błąd
- `__call__(self)` pozwala wywołać obiekt tak, jak funkcję

- Zadanie: Niech funkcja z poprzedniego zadania zwraca wynik po wywołaniu

Wyświetlenie obiektu

- __str__(self) definiuje w jaki sposób obiekt jest konwertowany na stringa
- Konwersja w wielu miejscach następuje automatycznie (np. po wywołaniu funkcji print)

- Zadanie: Zaimplementować wyświetlanie funkcji liniowej

Iteratory

- Iterator to obiekt, przez który możemy przejść przy pomocy pętli for
- Iterator wymaga zdefiniowania dwóch metod:
  - `__iter__(self)`: zwraca samego siebie
  - `__next__(self)`: zwraca wynik kolejnej iteracji. Kończy się wyjątkiem `StopIteration`
  - Oczywiście przydatny jest też `__init__(self, ...)`, do którego możemy podać wstępne dane

`line4(iterator).py`

Źródła

- https://docs.python.org/3
- https://www.geeksforgeeks.org/
- https://www.pythonlikeyoumeanit.com
- htps://www.realpython.com