PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 9

AGENDA
DAY 9

- wstęp do obiektowości
- klasy
- instancje – obiekty
- atrybuty i metody klas

OBIEKTY

```
1234    2.343534    'Magdalena'     [1, 3, 5, 7, 9]
{'imie': Andrzej', 'nazwisko': 'kowalski'}
```

dane ww. są instancjami obiektu, każdy obiekt ma:

- typ
- wewnętrzną reprezentację danych (prosta, złożona)
- zestaw procedur do interakcji z obiektem (in. interfejs)

Każda instancja jest konkretnym typem obiektu:

- 1234 jest instancją **int**
- x = 'Natalia' – x jest instancją **string**

OOP – Object Oriented Programming

- w Python – wszystko jest obiektem i posiada typ
- obiekty są abstrakcjami danych, które zawierają:
    - wewnętrzną reprezentację poprzez atrybuty danych
    - interfejs do interakcji z obiektem poprzez metody

- można tworzyć nowe instancje obiektów
- można niszczyć obiekty
    - wyraźnie – używając metody del
    - 'zapomnieć' – **Garbage Collector** usunie niedostępne lub zniszczone obiekty

OOP

[1,2,3,4] ma typ list
Jaka jest wewnętrzna reprezentacja?

L =

Jak można manipulować listami?

```
L[i], L[i:j], L[i,j,k], +

len(), min(), max(), del(L[i])

L.append(),L.extend(),L.count(),L.index(),L.insert(),
L.pop(),L.remove(),L.reverse(), L.sort()
```

OOP

wewnętrzna reprezentacja obiektu powinna być **prywatna**

Właściwe zachowanie obiektu, może być zagrożone, jeśli będziemy manipulować bezpośrednio na wnętrzu obiektu – należy używać zdefiniowanych interfejsów (atrybutów i metod)

KLASA vs. INSTANCJA

KLASA – jest "ideą", "schematem", "wyobrażeniem"
właściwości (zmienne) i interfejs (metody)

INSTANCJA – jest "powołanym do życia" obiektem, który zawiera określone przez klasę właściwości.
Można mieć kilka instancji jednej klasy

KLASA vs INSTANCJA

Do stworzenia klasy potrzebujemy:

- nazwy klasy
- zdefiniować właściwości klasy

Używanie klasy polega na:

- utworzeniu nowej instancji obiektu
- wykonywaniu operacji na instancji

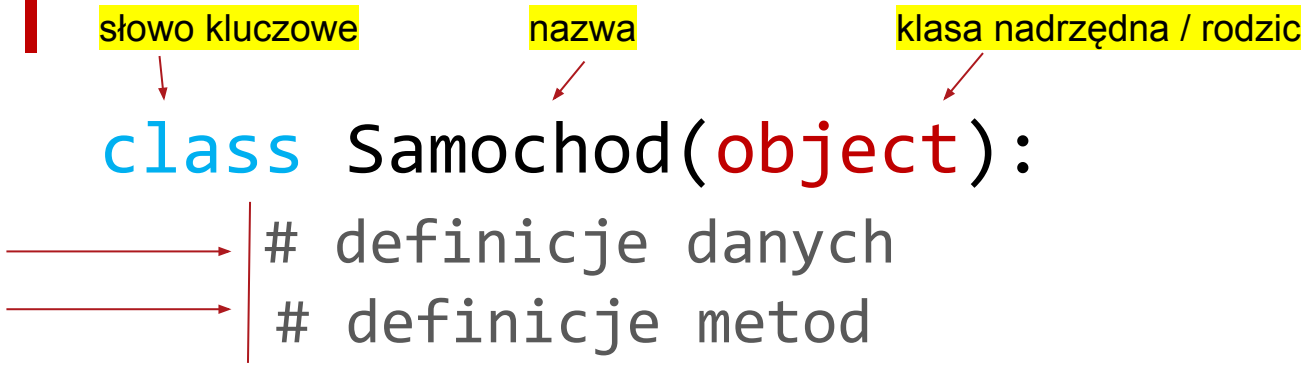
ZALETY OOP

- **tworzenie jednorodnego pakietu**, zawierającego dane oraz sposoby manipulowania nimi

- umożliwiają podejście **– divide and conquer** (dziel i zwyciężaj)
  - można testować zachowanie każdej z klas oddzielnie
  - zwiększa modularność, zmniejsza kompleksowość

- **klasy ułatwiają ponowne użycie kodu**
  - każda z klas tworzy oddzielne "środowisko" – różne klasy mogą mieć takie same nazwy funkcji
  - dziedziczenie pozwala aby podklasa, zredefiniowała lub rozszerzyła wybrane właściwości klasy nadrzędnej

definiowanie klas



- **class** – podobnie jak **def**
- słowo **object** oznacza, że Samochód jest obiektem w Python (object) i **dziedziczy** z niego wszystkie właściwości
  - Samochod jest podklasą object
  - object jest klasą nadrzędną dla Samochod

DEFINIOWANIE KONSTRUKTORA

parametr – referencja instancji

dane inicjalizujące

```
class Samochod(object):
    def __init__(self, marka, model):
        self.marka = marka
        self.model = model
```

specjalna metoda w Python ma 2 podkreślenia double-under-score in. **dunder**

atrybuty każdej instancji obiektu Samochod

DEFINIOWANIE METOD

```
def accelerate(self, value):
    self.speed += value
```

OOP

Paradygmaty OOP:

- enkapsulacja
- dziedziczenie
- polimorfizm

Thanks!!