

# PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 11



# AGENDA

## DAY 11

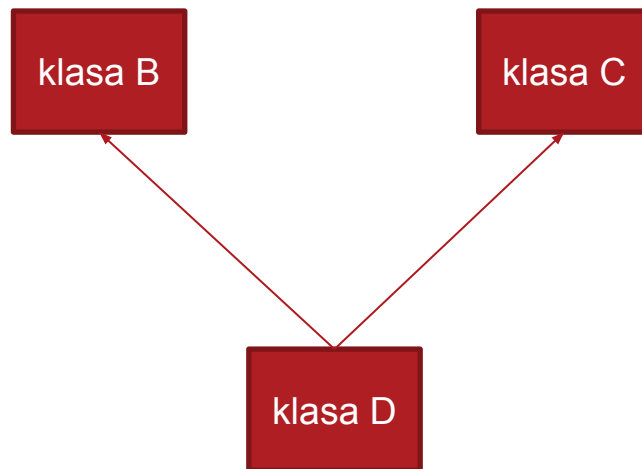
- dziedziczenie diamentowe
- pola klasy
- metody klas
- metody statyczne

# dziedziczenie diamentowe



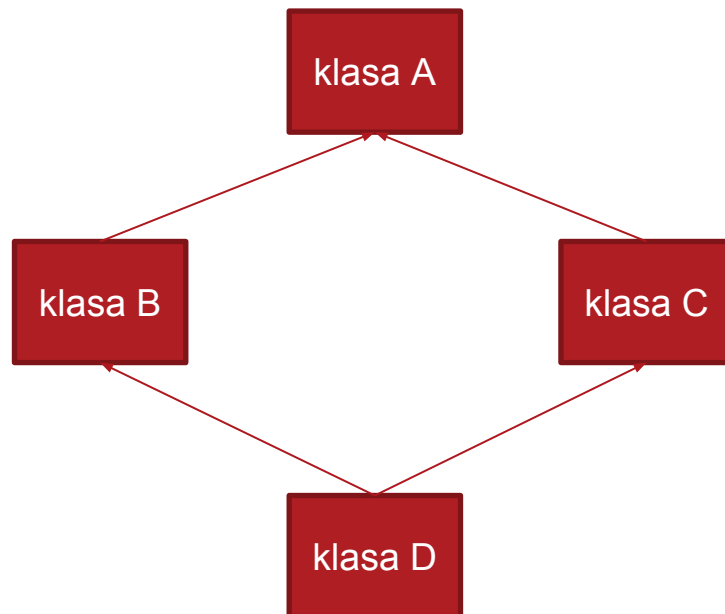
# DZIEDZICZENIE OD WIELU RODZICÓW

Klasa może dziedziczyć z wielu klas

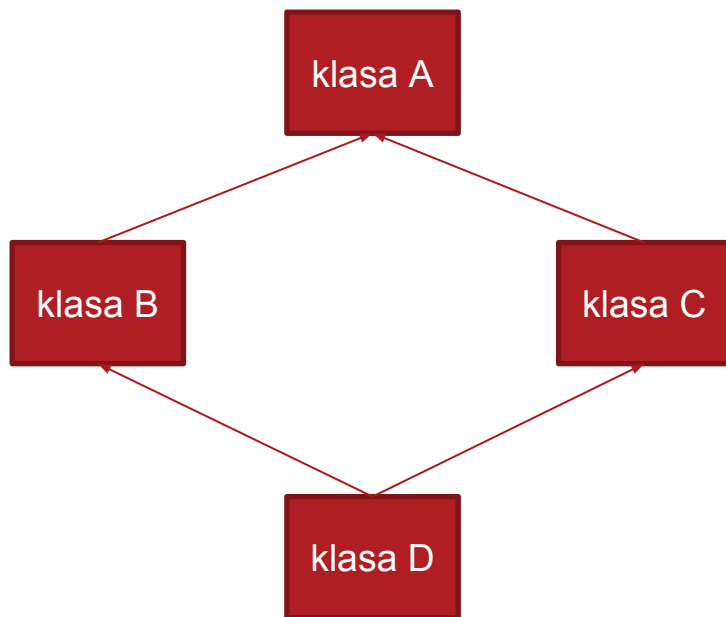


# DZIEDZICZENIE DIAMENTOWE

Ale co w przypadku dziedziczenia diamentowego?



# DZIEDZICZENIE DIAMENTOWE



Klasa dziecka będzie szukać atrybuty w kolejności od lewej do prawej, z dołu w górę.

W poniższym przykładzie, najpierw poszuka w klasie Horse, a następnie w Donkey

```
class Mule(Horse, Donkey):  
    pass
```



# super()

*nie jest taki super*

Musimy uważać jeśli dziedziczymy używając **super()** jako odwołanie do klasy nadrzędnej.

**pola klasy, metody klasy,  
metody statyczne**



# POLA KLASY

Zmienne definiowane na poziomie klasy.

**Nie używamy** słówka **self**

Służą do przechowywania danych niezależnych od instancji (wspólne dla wszystkich instancji)

# METODY KLASY

Metody, które jako pierwszy argument przyjmują klasę zamiast instancji.

Używamy **dekoratora** `@classmethod` nad definicją metody.

Pierwszy argument to słowo kluczowe `cls`

Możemy używać jako alternatywne konstruktory

```
@classmethod  
def my_class_method(cls):  
    pass
```

# METODY statyczne

Metody, które nie przyjmują ani instancji ani klasy jako argument. Wyglądają jak normalne metody

Używamy **dekoratora** `@staticmethod` nad definicją metody.

Używamy je gdy przekazanie jakiejś informacji nie wymaga tworzenia instancji klasy. (matematyczne)

```
@staticmethod  
def my_static_method():  
    pass
```

```
MyClass.my_static_method()
```

# | pola i metody | pseudo-prywatne

**enkapsulacja**

# pola i metody pseudo-prywatne

Python daje możliwość stworzenia pseudo-prywatnych pól i metod.

Do nazwy (pola, metody) dodajemy dwa podkreślniki tylko z przodu. Można je użyć wewnątrz klasy, ale poza nią są niewidoczne – ale można i tak ich użyć!!!

```
self.__moje_pole_prywatne
```

```
def __metoda_prywatna(self, arg1):  
    self.__moje_pole_prywatne = arg1
```

# namespace

Namespace jest obszarem nazw, które są dostępne dla klasy.

```
print(MojaKlasa.__dict__)  
print(instancja.__dict__)
```

W ten sposób możemy znaleźć pseudo-prywatny atrybut

# | properties: setter & | getter

# PROPERTIES

Properties – właściwości

definiujemy jak metody z dekoratorem, z nazwą identyczną jak zmienna, służą do manipulowania zmiennymi w kontrolowany przez nas sposób.

**Wywołujemy bez nawiasów !!!!**



# PROPERTIES - GETTER

Getter – służy do zwrócenia wartości ze zmiennej

```
self.__name
```

```
@property
```

```
def name(self):  
    return str(self.__name).capitalize()
```

# PROPERTIES - SETTER

setter – służy do zapisania wartości do zmiennej – daje możliwość do kontrolowania tego co zapisujemy

`self.__name`

`@name.setter`

```
def name(self, name):  
    self.__name = name
```

# PROPERTIES - DELETER

setter – służy do usuwania zawartości zmiennej w kontrolowany sposób

`self.__name`

`@name.deleter`

`def name(self):`

`self.__name = None`

■ [bit.ly\ankieta2pyt](https://bit.ly/ankieta2pyt)



# Thanks!!