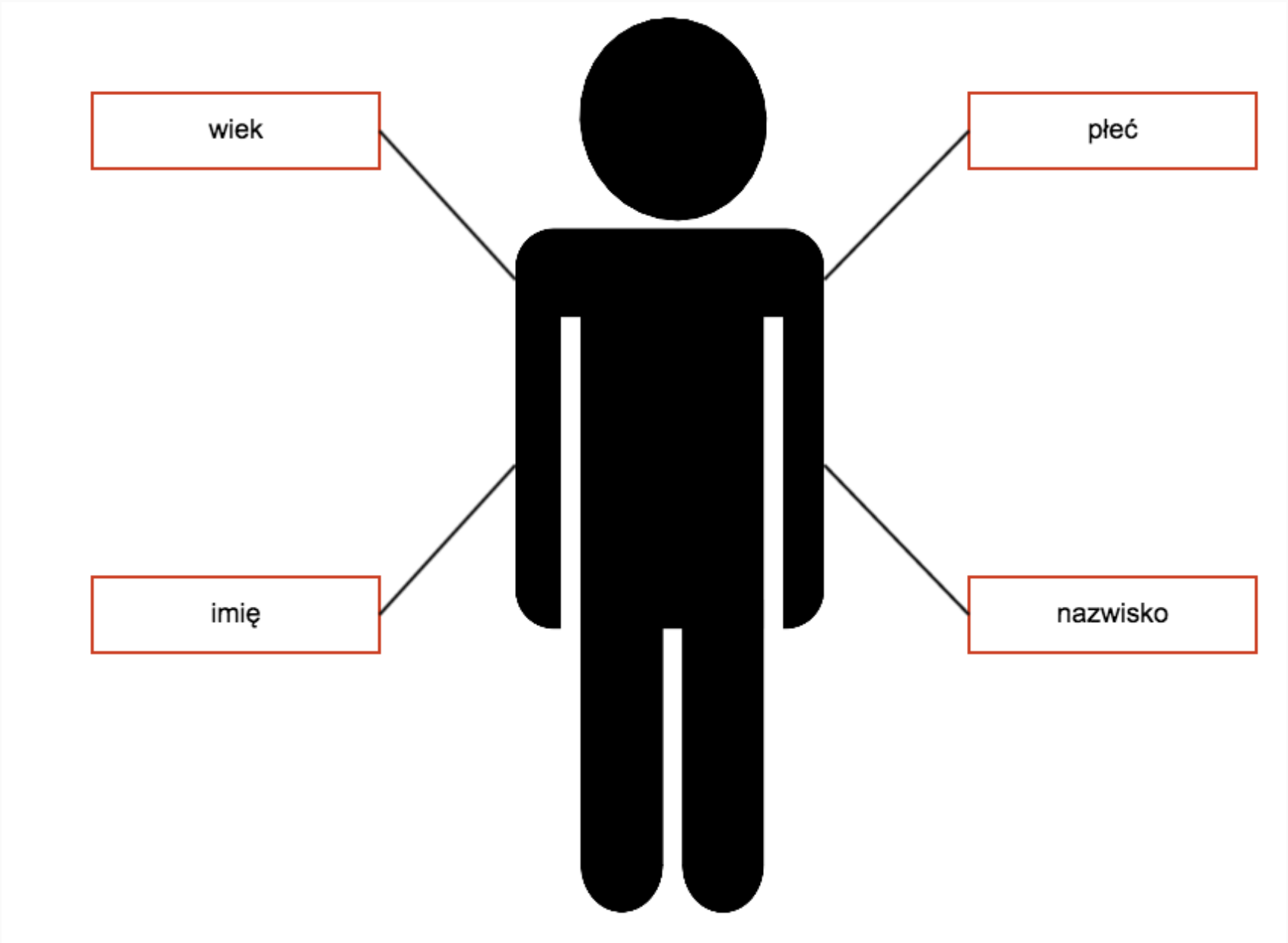
Python - programowanie obiektowe

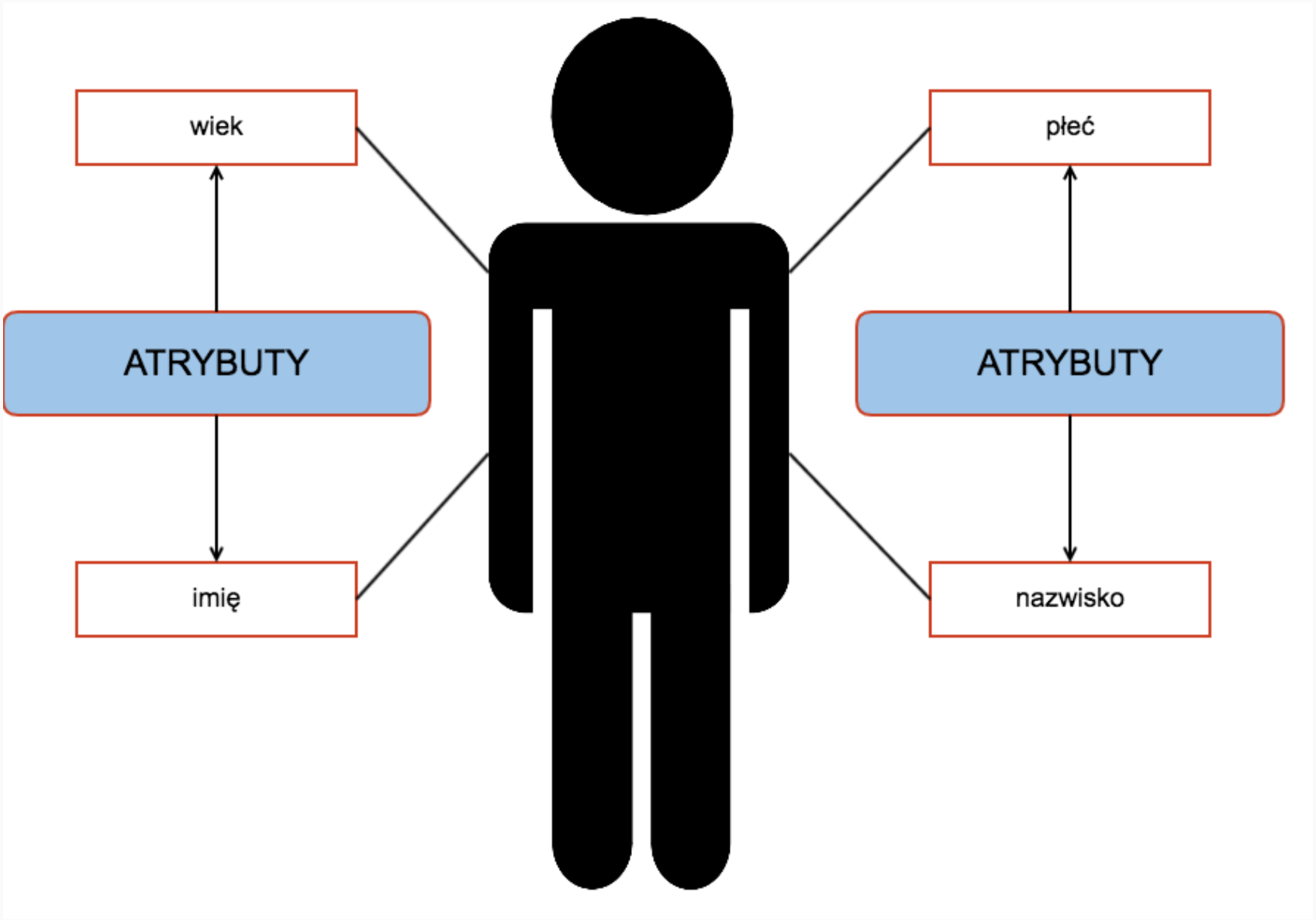
Michał Gałka

Programowanie obiektowe

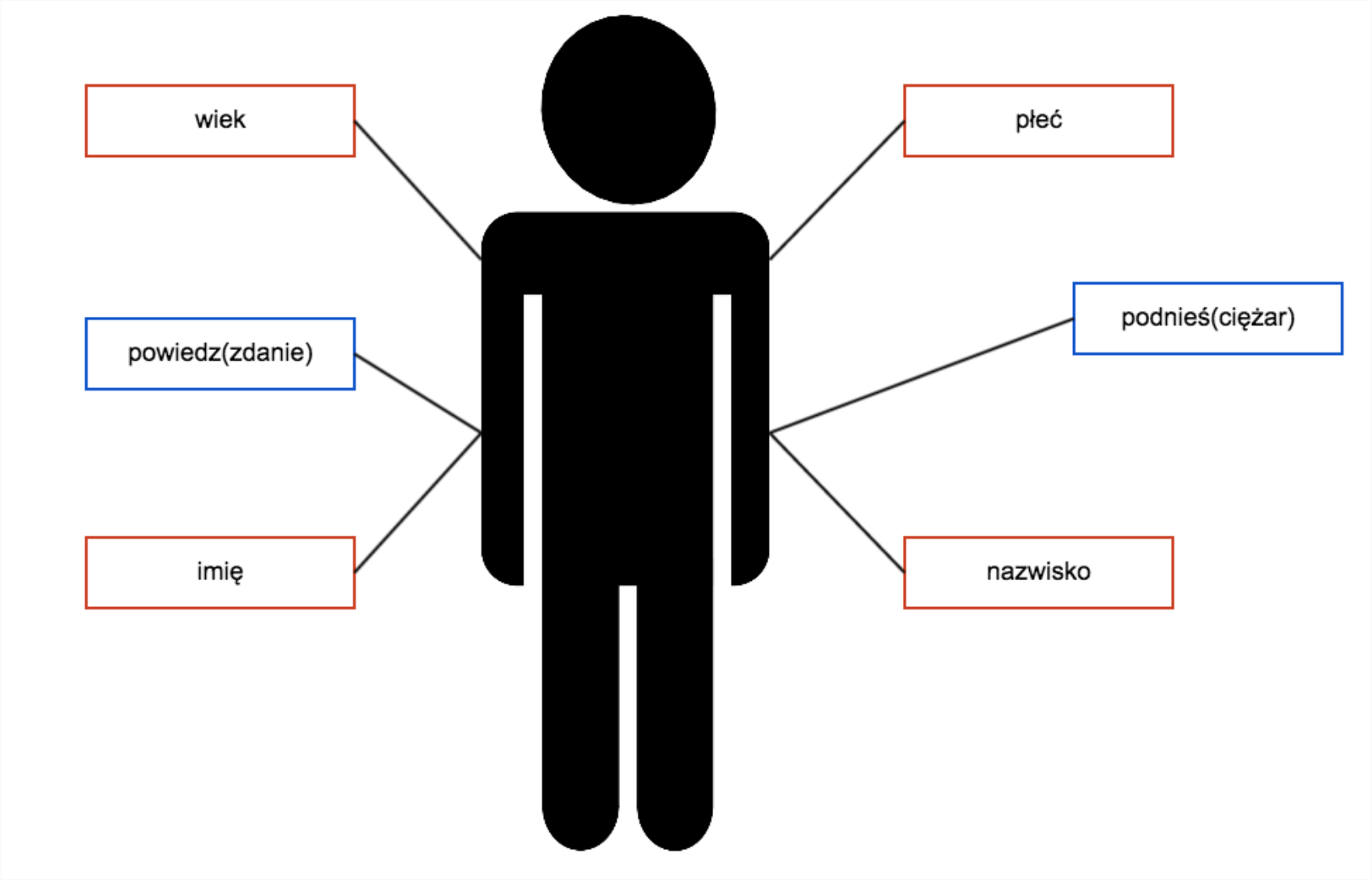
Obiekt



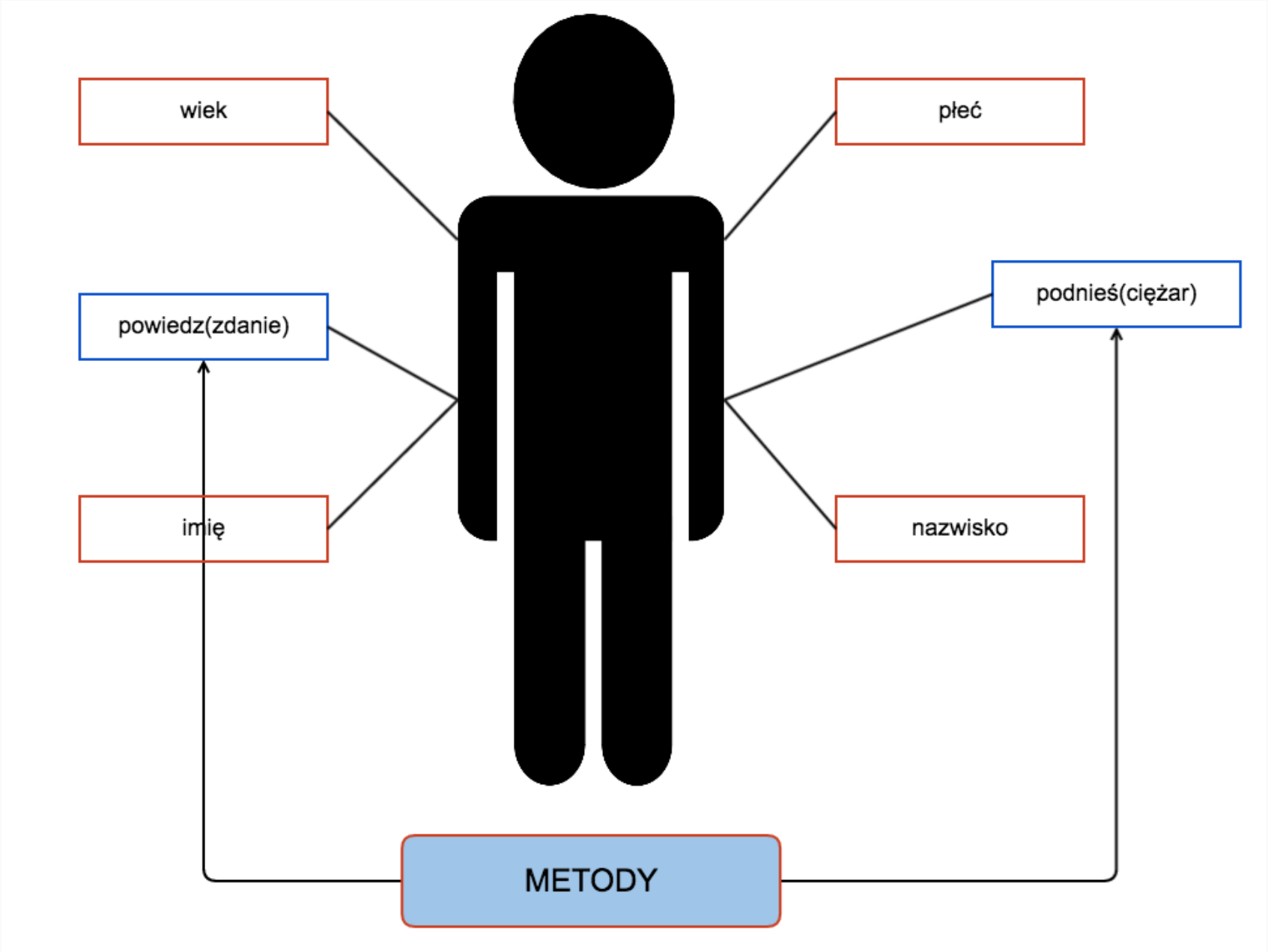
Obiekt



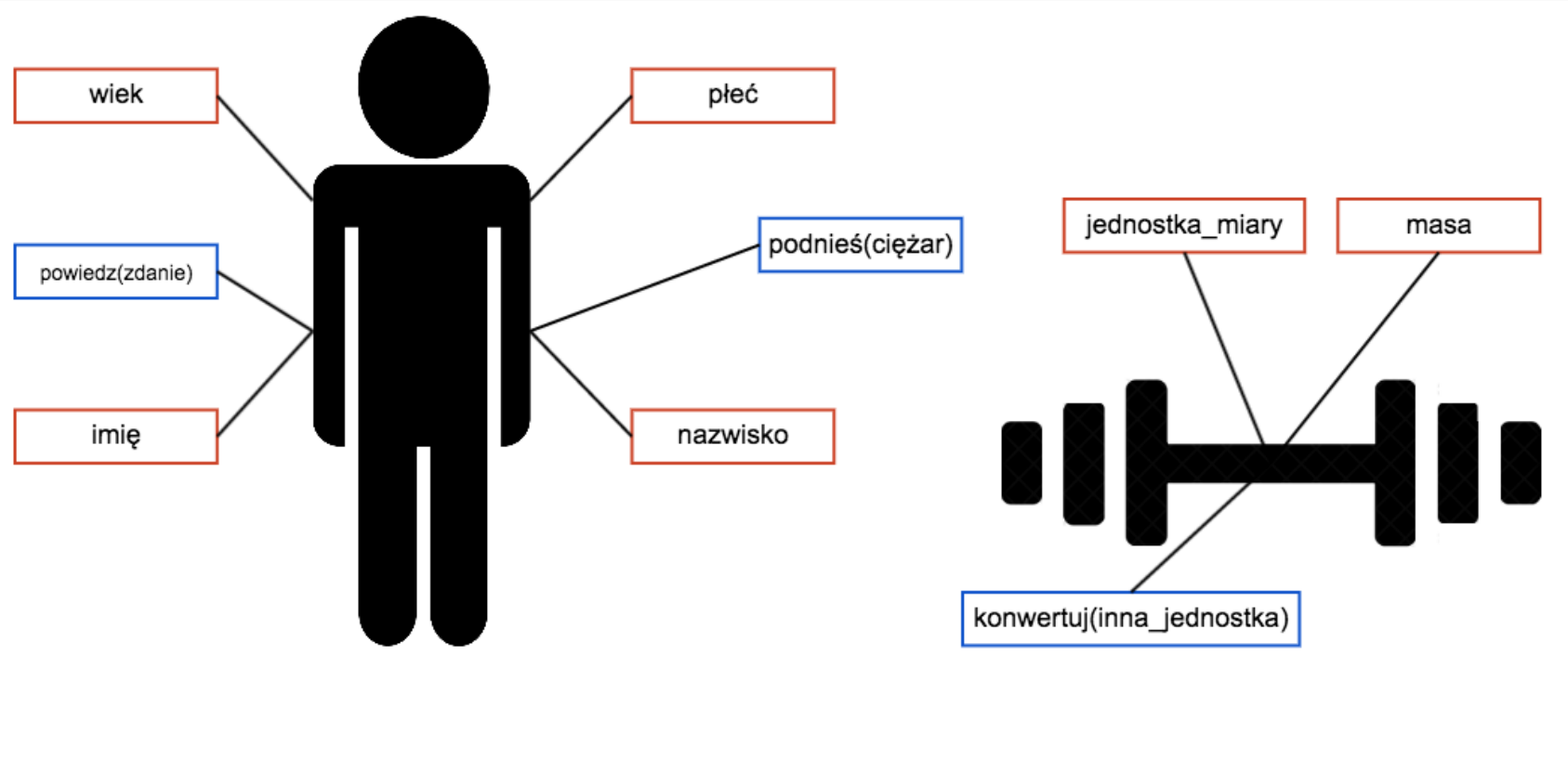
Obiekt



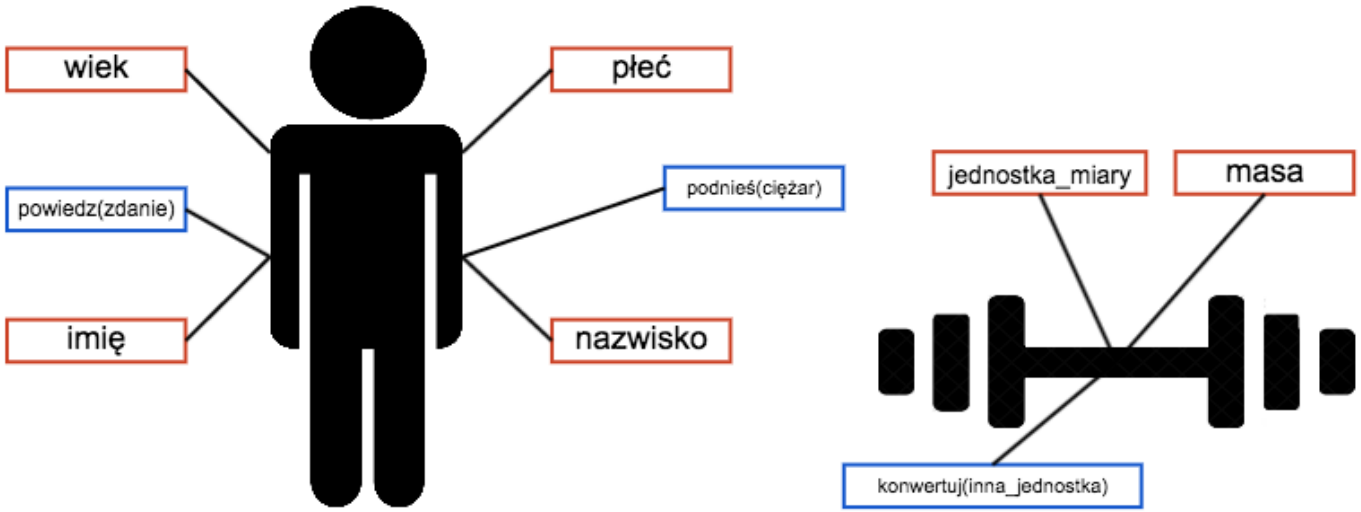
Obiekt


wiek
płeć
powiedz(zdanie)
podnieś(ciężar)
imię
nazwisko
METODY

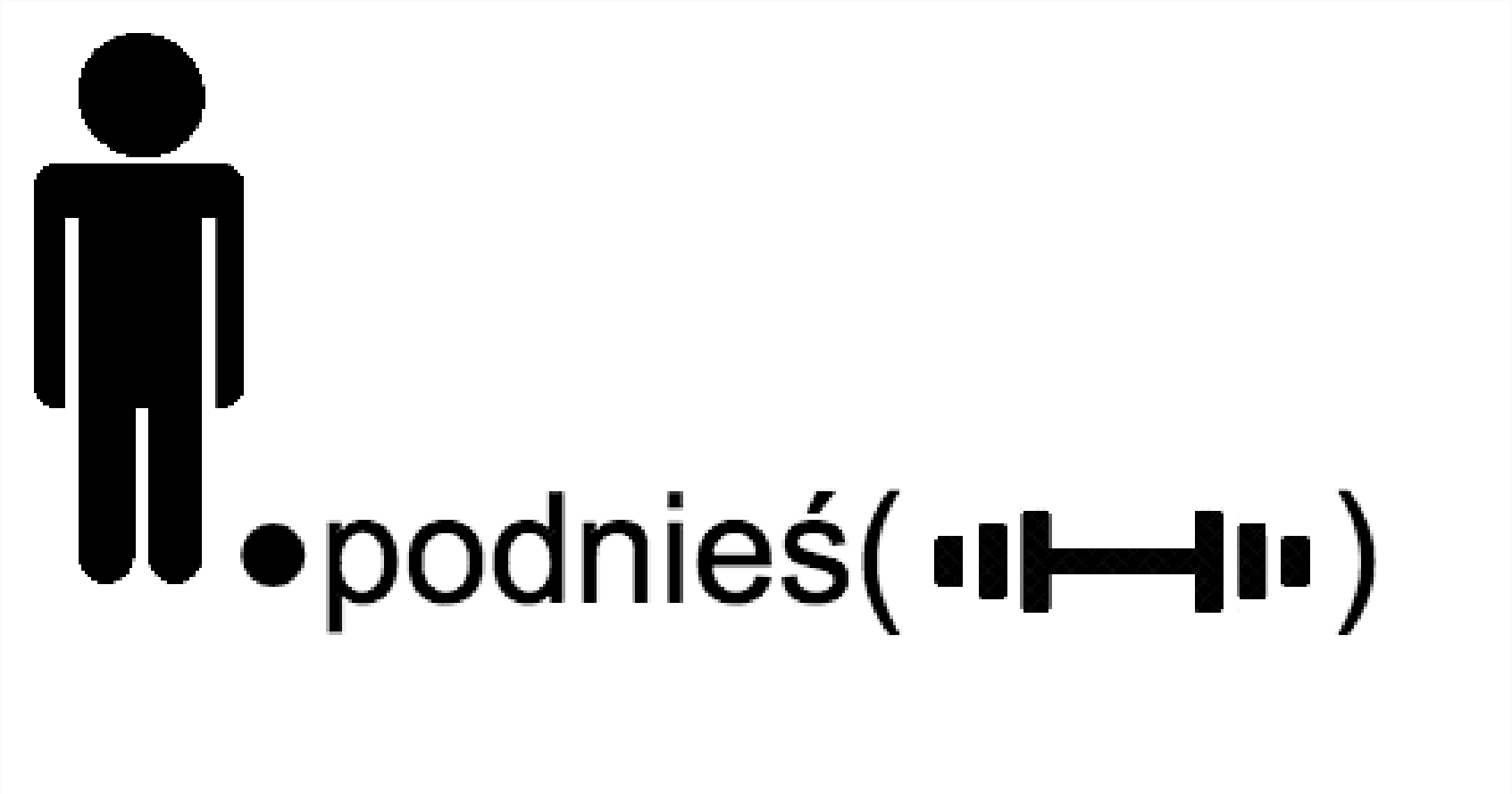
Obiekt


wiek
płeć
powiedz(zdanie)
podnieś(ciężar)
imię
nazwisko
jednostka_miary
masa
konwertuj(inna_jednostka)

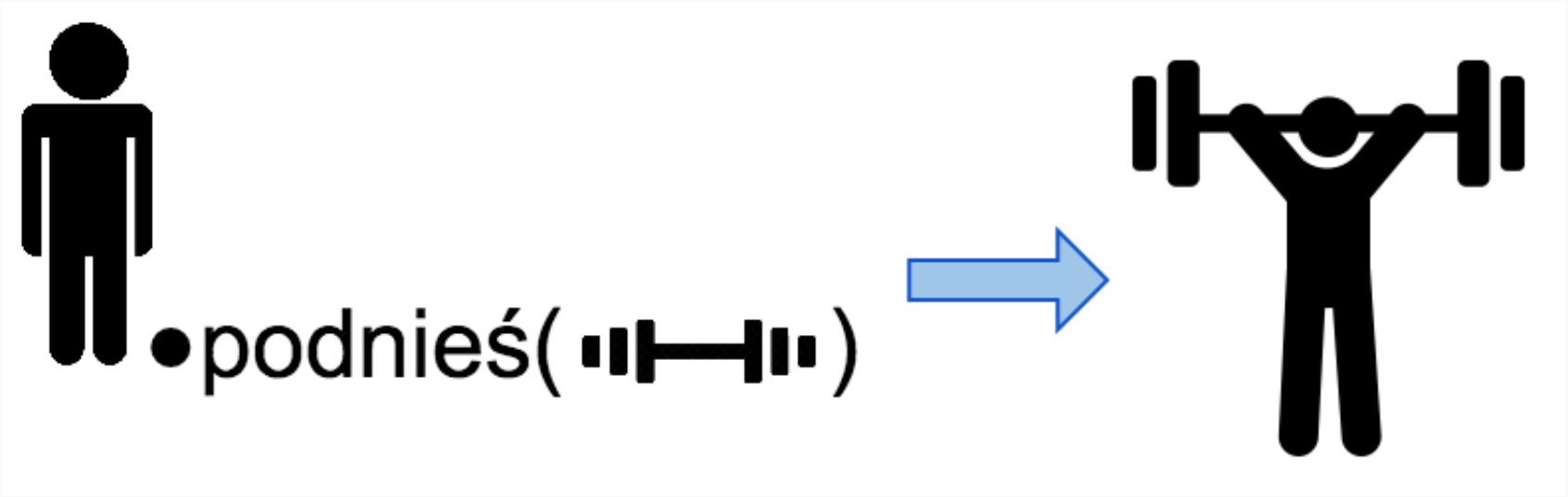
Obiekt


wiek
płeć
powiedz(zdanie)
podnieś(ciężar)
imię
nazwisko
jednostka_miary
masa
konwertuj(inna_jednostka)

Obiekt



Obiekt



Model danych

- W Pythonie wszystko jest obiektem.
- Oznacza to, że każdy element języka Python jest obiektem.
- Każdy obiekt w Pythonie ma toższamość **(ang. identity)**, **typ (ang. type)** i **wartość (ang. value)**.

Tożsamość obiektu

- Tożsamość obiektu nigdy się nie zmienia.
- Wbudowana funkcja `id()` zwraca liczbę całkowitą będącą tożsamością obiektu.
- W interpreterze CPython `id(x)` zwraca adres zmiennej `x`.
- Do porównania tożsamości dwóch obiektów służy operator `is`.

```
>>> class Foo:
...     pass
...
>>> foo = Foo()
>>> bar = Foo()
>>> baz = bar

>>> print(foo is bar)
>>> print(bar is baz)

False
True
```

Typ obiektu

- Typ obiektu jest także niezmienny.
- Typ określa zachowanie obiektu.
- Typ obiektu można sprawdzić przy pomocy funkcji `type()`.
- Typ jest także obiektem.

```
>>> class Foo: pass
...
>>> foo = Foo()
>>> type(foo)
<class '__main__.Foo'>
```

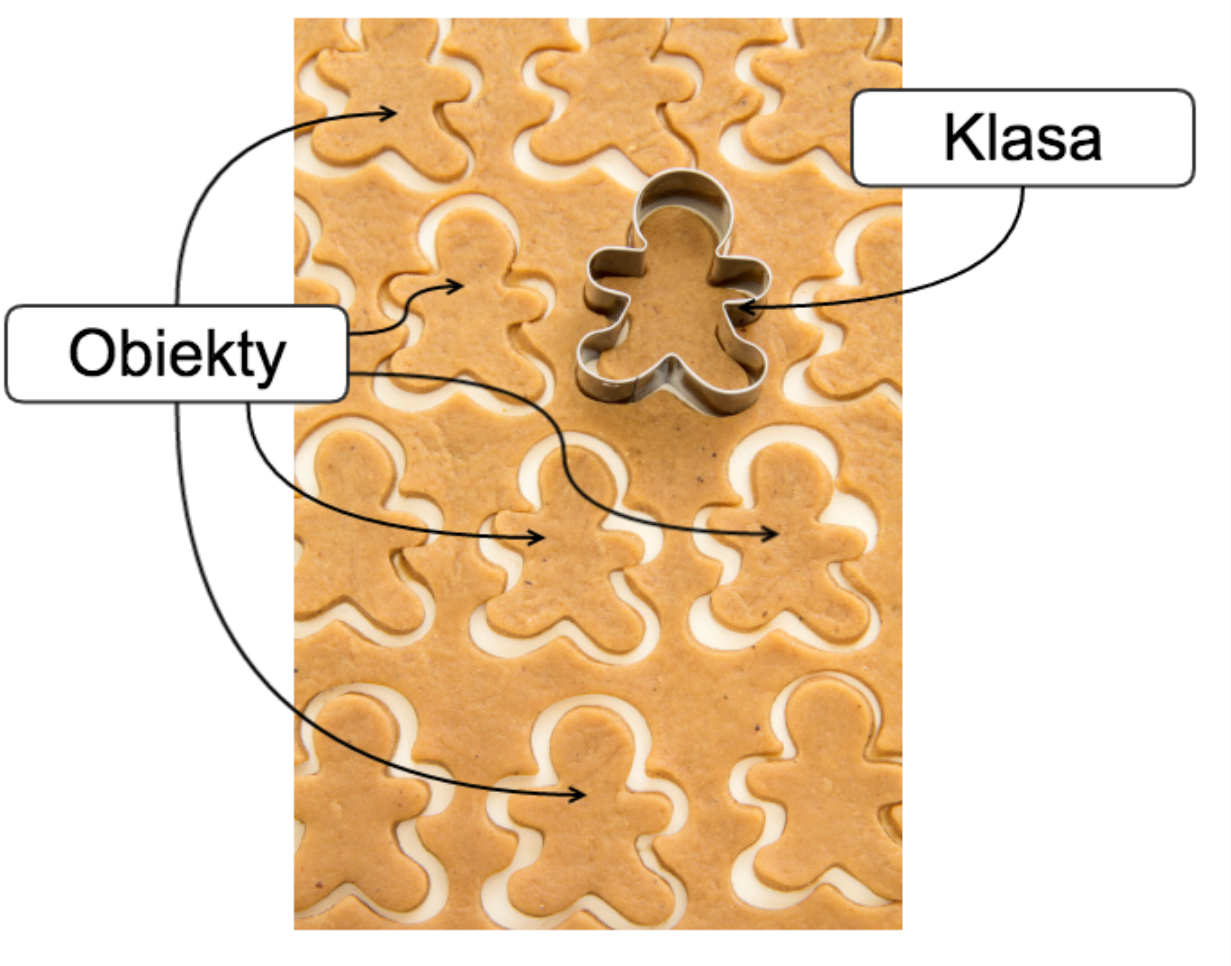
Wartość

- Wartość pewnych obiektów może ulegać zmianie:
  - **Obiekty zmienne** (ang. **mutable**) – ich wartość może być zmieniana
  - **Obiekty niezmienne** (ang. **immutable**) – ich wartość nie może być zmieniana
- Niezmienne kontenery mogą zawierać referencje do zmiennych obiektów.

Obiekt

- Obiekty są zbiorem zmiennych i funkcji opakowanych w jedną encję.
- Zmienne w obiektach nazywamy atrybutami obiektu.
- Funkcje w obiekatach nazywamy metodami obiektu.
- Dostęp do metod i atrybutów obiektu odbywa się poprzez operator `.` (kropka).

Klasa i Obiekt



- Klasy są definicjami obiektów.
- Zawierają metodę tworzącą i inicjalizującą nowe obiekty.
- Obiekty często nazywamy instancjami klasy.

Definicja klasy

```
class Human:
    def __init__(self, first_name, last_name, age, sex):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.sex = sex

    def say(self, sentence):
        print('{} SAYS {}'.format(self.first_name,
                                  sentence))

    def lift(self, weight):
        print('{} LIFTS {} {}'.format(self.first_name,
                                      weight.weight,
                                          weight.unit))
```

```
class Weight:
    conversion_table = {
      'kg': {
                'lbs': 0.45359237
      },
      'lbs': {
         'kg': 2.20462262
      }
    }

    def __init__(self, weight, unit='kg'):
        self.weight = weight
        self.unit = unit

    def convert_to(self, new_unit):
        ratio =
            self.conversion_table[self.unit][new_unit]
        converted_weight = self.weight * ratio
        return converted_weight
```

Atrybuty instancji (obiektów) i klas

- Atrybuty zadeklarowane w metodzie `__init__()` nazywane są atrybutami obiektu.
- Atrybuty zadeklarowane poza metodą `__init__()` (na poziomie klasy) nazywane są atrybutami klasy.
- Atrybuty klasy są wspólne dla wszystkich jej instancji.
  - Zmiana ich wartości widoczna jest w każdej instancji.

`__init__` i `self`

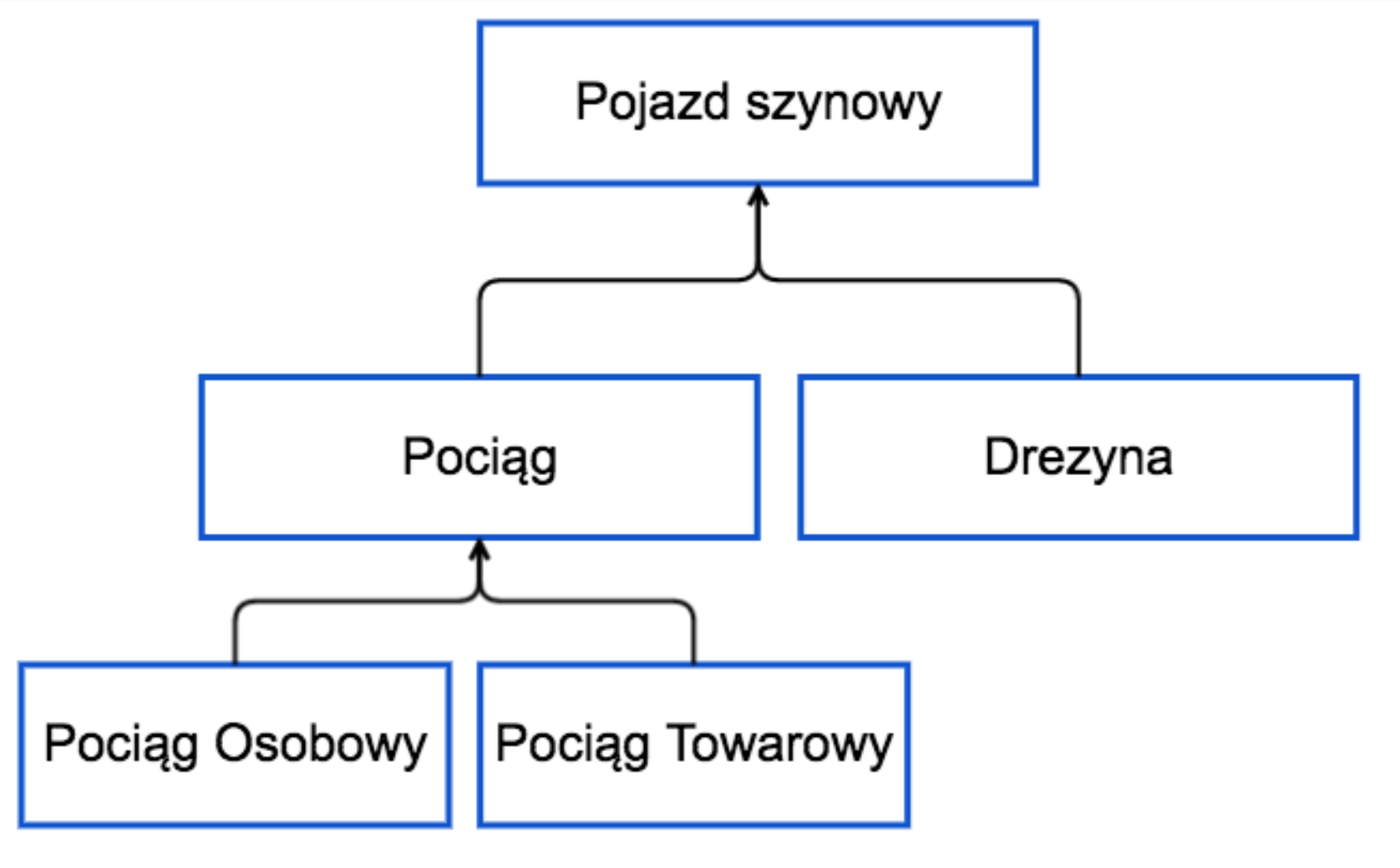
- Metoda `__init__()`
  - Jest wywoływana przy tworzeniu instancji klasy.
  - Jest używana do inicjalizacji instancji.
- Parametr `self` reprezentuje bieżącą instancję klasy
  - Spełnia taką samą funkcję jak `this` w innych językach programowania.
  - Definicja `self` nie jest domyślna w Pythonie.
  - Nazwa “self” jest tylko konwencją.

Praca z klasami i obiektami

```
john = Human('John', 'Doe', 32, 'M')
jane = Human('Jane', 'Doe', 21, 'F')
weight = Weight(16, 'kg')
john.lift(weight)
jane.say('Hello World!')
print('{} {}'.format(weight.weight, weight.unit))
unit = 'lbs'
print('{} {}'.format(weight.convert_to(unit), unit))

John LIFTS 16 kg
Jane SAYS Hello World!
16 kg
7.25747792 lbs
```

Dziedziczenie



Dziedziczenie - przykład

```
class Animal:
    def __init__(self, number_of_legs):
        self.number_of_legs = number_of_legs

    def say(self):
        pass

class Dog(Animal):

    def __init__(self):
        super().__init__(4)

    def say(self):
        print('Woof Woof!')
```

```
class Pig(Animal):
    def __init__(self):
        super().__init__(4)

    def say(self):
        print('Oink Oink!')
```

Dziedziczenie - przykład

```
pig = Pig()
dog = Dog()

dog.say()
print(dog.number_of_legs)
pig.say()
print(pig.number_of_legs)

Woof Woof!
4
Oink Oink!
4
```

Metody specjalne

- Python zawiera specjalne zmienne, funkcje oraz metody, służące różnym celom.
- Ich nazwy tworzone są według wzoru `__nazwa__`

Metody specjalne

- Klasy w języku Python implementują pewne operacje przy pomocy metod specjalnych:
  - Operacje arytmetyczne
  - Dostęp do atrybutów
  - Obsługa indeksów i slice

Metody specjalne

- `object.__repr__(self)`
  - Wywoływana przez wbudowaną funkcję repr().
  - Zwraca reprezentację obiektu.
  - Jeśli to możliwe powinna zwracać wyrażenie w języku Python pozwalające na stworzenie danego obiektu.
  - Jeśli nie jest to możliwe to powinna zwrócić informację w formacie `<opis obiektu>`

Metody specjalne

- `object.__str__(self)`
  - Zwraca “nieoficjalną” reprezentację obiektu jako łańcuch znaków.
  - Zwracana wartość nie musi być poprawnym wyrażeniem w języku Python.
  - Zwracana wartość musi być łańcuchem znaków.

Metody specjalne

- Metody porównań tzw. “rich comparison”
  - `object.__lt__(self, other)` (<)
  - `object.__le__(self, other)` (<=)
  - `object.__eq__(self, other)` (==)
  - `object.__ne__(self, other)` (!=)
  - `object.__gt__(self, other)` (>)
  - `object.__ge__(self, other)` (>=)

Porównania

- Mogą zwracać dowolną wartość.
- Według konwencji zwracają obiekty `True` or `False`.
- Mogą zwrócić obiekt `NotImplemented` jeśli nie można wykonać porównania dla danych typów.
- Funkcja `bool()` jest używana do określenia wartości logicznej wyniku porównania.

Porównania

- Jeśli żadna z metod `__eq__()` i `__ne__()` nie jest zdefiniowana, obiekty są porównywane na podstawie swojej tożsamości.
- Można uprościć tworzenie operatorów porównań używając `functools.total_ordering()`