Obiekty w Pythonie

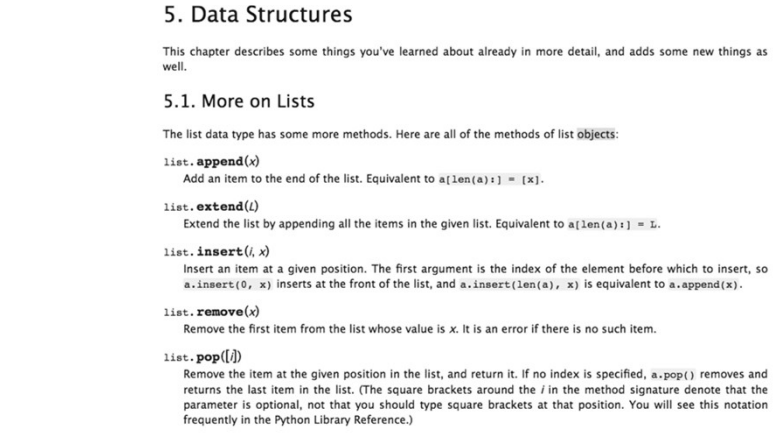
Rozdział 14
Charles Severance

UMSI

Python dla wszystkich
www.py4e.pl

Uwaga

- Ten wykład dotyczy definicji i mechaniki obiektów
- Ten wykład mówi o wiele więcej o tym, "jak to działa" a nie, "jak tego użyć"
- Nie będziesz mieć pełnego obrazu, dopóki nie przyjrzysz się temu w kontekście prawdziwych problemów
- Więc uzbrój się w cierpliwość i na następne czterdzieści kilka slajdów zanurz się w nauce techniki



5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.append(x)`
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

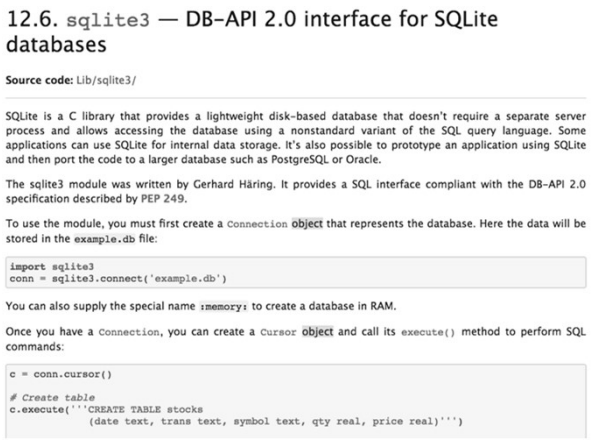
`list.extend(L)`
Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

`list.insert(i, x)`
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.remove(x)`
Remove the first item from the list whose value is *x*. It is an error if there is no such item.

`list.pop([i])`
Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

https://docs.python.org/3/tutorial/datastructures.html



12.6. `sqlite3` — DB-API 2.0 interface for SQLite databases

Source code: Lib/sqlite3/

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249.

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')
```

https://docs.python.org/3/library/sqlite3.html

Zacznijmy od programów

```
inp = input('Europejskie piętro? ')
usf = int(inp) + 1
print('Amerykańskie piętro:', usf)
```

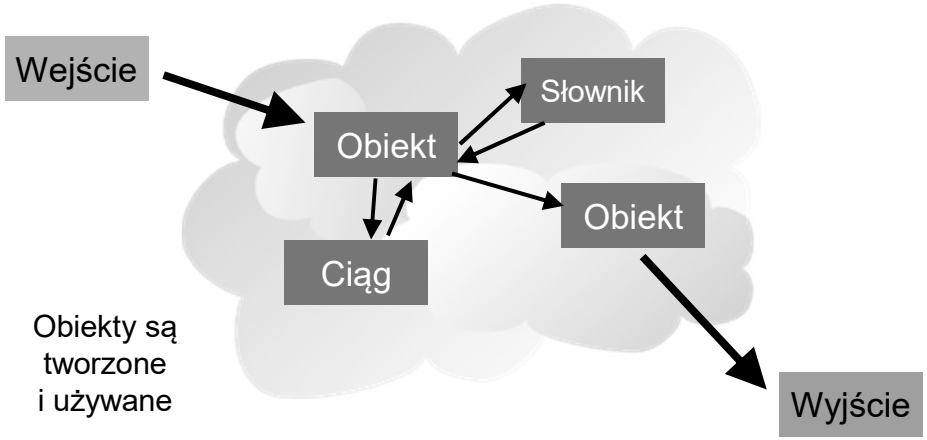
```
Europejskie piętro? 0
Amerykańskie piętro: 1
```

Obiektowość

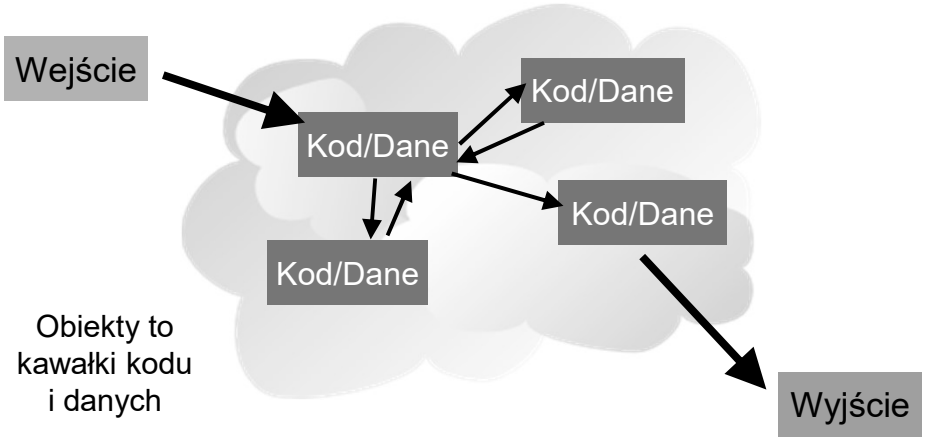
- Programy złożone są z wielu współpracujących obiektów
- Zamiast być "całym programem", każdy obiekt jest małą "wyspą" w programie i współdziała z innymi obiektami
- Program jest złożony z wielu obiektów pracujących razem – obiekty wykorzystują możliwości innych obiektów

Obiekt

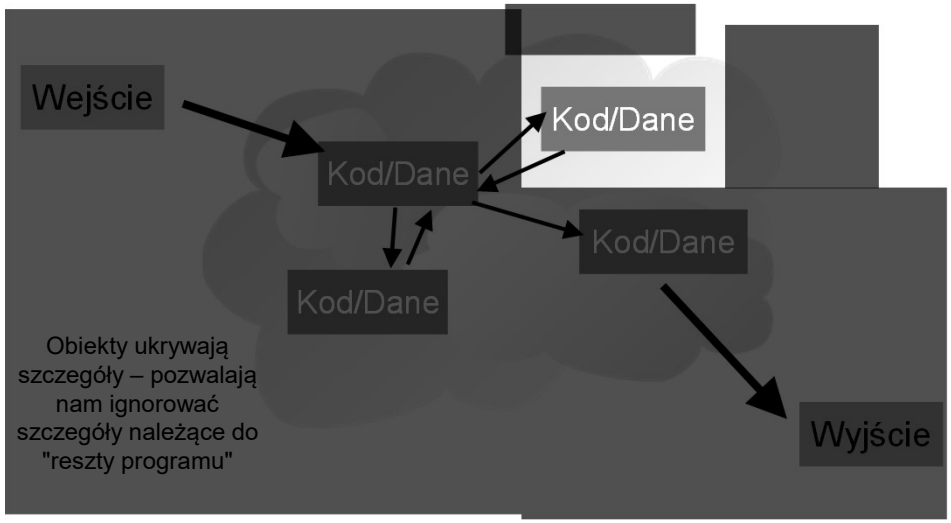
- Obiekt to niezależny fragment Kodu i Danych
- Kluczowym aspektem podejścia obiektowego jest dzielenie problemu na mniejsze, zrozumiałe części (dziel i rządź)
- Obiekty mają granice, dzięki którym możemy ignorować niepotrzebne szczegóły
- Od początku korzystaliśmy z obiektów: Obiekty ciągów, Obiekty liczb całkowitych, Obiekty słowników, Obiekty list



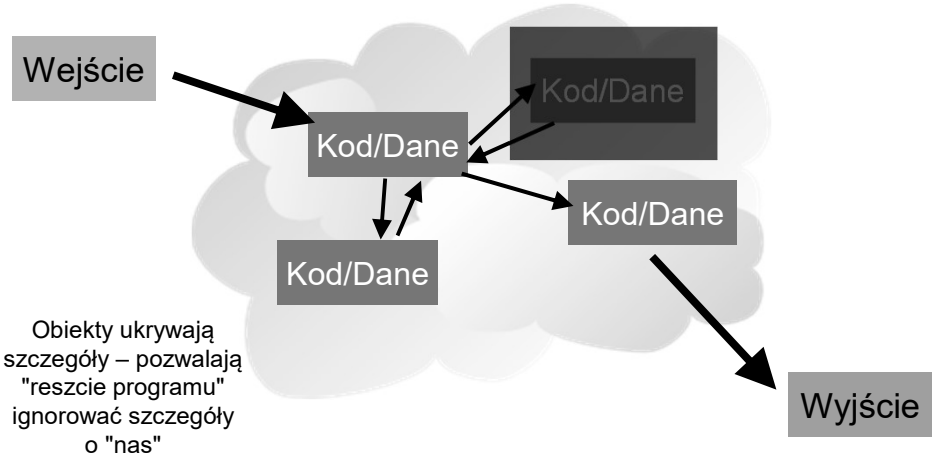
Obiekty są tworzone i używane



Obiekty to kawałki kodu i danych



Obiekty ukrywają szczegóły – pozwalają nam ignorować szczegóły należące do "reszty programu"



Obiekty ukrywają szczegóły – pozwalają "reszcie programu" ignorować szczegóły o "nas"

Definicje

- Klasa – szablon
- Metoda lub wiadomość – zdefiniowana zdolność klasy
- Pole lub atrybut – fragment danych w klasie
- Obiekt lub instancja – konkretna instancja klasy

Terminologia: Klasa

Określa abstrakcyjne cechy rzeczy (obiektu) włącznie z jego charakterystyką (atrybutami, polami, czyli właściwościami) oraz jego zachowaniami (tym, co może zrobić, czyli metodami, operacjami lub możliwościami). Można powiedzieć, że klasa jest projektem lub fabryką opisującą naturę rzeczy. Na przykład, klasa Pies zawierałaby cechy współdzielone przez wszystkie psy, takie jak rasa i kolor sierści (charakterystyka) oraz zdolność szczekania i warowania (zachowania).

https://pl.wikipedia.org/wiki/Programowanie_obiektowe

Terminologia: Instancja

Możemy mieć instancję klasy lub konkretnego obiektu. Instancja to obiekt tworzony podczas wykonania programu. W żargonie programistów obiekt Lassie jest instancją klasy Pies. Zbiór wartości atrybutów konkretnego obiektu nazywamy jego stanem. Na obiekt składają się jego stan i zachowanie określone w klasie obiektu.

Terminów obiekt i instancja często używa się wymiennie.

https://pl.wikipedia.org/wiki/Programowanie_obiektowe

Terminologia: Metoda

Zdolność obiektu. W językach programowania metody to czasowniki. Lassie to Pies, więc może szczekać (bark). Dlatego bark() to jedna z metod Lassie. Może też mieć inne metody, na przykład siad – sit(), jeść – eat(), iść – walk() albo ratować Timmy'ego – save_timmy(). W programie użycie metody zazwyczaj ma wpływ tylko na jeden obiekt; wszystkie Psy potrafią szczekać, ale każdy pies może szczekać samodzielnie.

https://pl.wikipedia.org/wiki/Programowanie_obiektowe

Niektóre obiekty w Pythonie

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
>>> type(2.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> y = list()
>>> type(y)
<class 'list'>
>>> z = dict()
>>> type(z)
<class 'dict'>
```

```
>>> dir(x)
[(...), 'capitalize', 'casefold', 'center',
'count', 'encode', 'endswith', 'expandtabs',
'find', 'format', … 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> dir(y)
[(...), 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
>>> dir(z)
[(...), 'clear', 'copy', 'fromkeys', 'get',
'items', 'keys', 'pop', 'popitem', 'setdefault',
'update', 'values']
```

Przykładowa klasa

class jest słowem zastrzeżonym

Każdy obiekt PartyAnimal zawiera trochę kodu

Powiedz obiektowi, aby wykonał swój kod party()

```
class PartyAnimal:
   x = 0

   def party(self) :
      self.x = self.x + 1
      print("Jak na razie", self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

To jest szablon do tworzenia obiektów PartyAnimal

Każdy obiekt PartyAnimal zawiera trochę danych

Stwórz obiekt PartyAnimal i zapisz go w an

PartyAnimal.party(an)

```
class PartyAnimal:
   x = 0

   def party(self) :
      self.x = self.x + 1
      print("Jak na razie", self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

$ python3 party0.py

```
class PartyAnimal:
   x = 0

   def party(self) :
      self.x = self.x + 1
      print("Jak na razie", self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

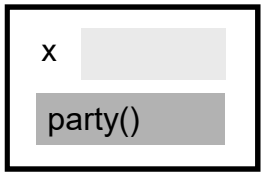
```
$ python3 party0.py
```

an


```
class PartyAnimal:
   x = 0

   def party(self) :
      self.x = self.x + 1
      print("Jak na razie", self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python3 party0.py
Jak na razie 1
Jak na razie 2
Jak na razie 3
```

an
self



PartyAnimal.party(an)

Zabawa z dir() i type()

Sprawdzanie możliwości dla kujonów

- dir() wyświetla listę możliwości
- Zignoruj te z podkreślnikami, używa ich tylko Python
- Cała reszta to działania, jakie może wykonać obiekt
- Podobnie jak type() mówi nam *coś o* zmiennej

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(y)
[(...), '__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', … '__setitem__',
'__setslice__', '__str__',
'append', 'clear', 'copy',
'count', 'extend', 'index',
'insert', 'pop', 'remove',
'reverse', 'sort']
>>>
```

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("Jak na razie", self.x)

an = PartyAnimal()

print("Type", type(an))
print("Dir ", dir(an))
print("Type", type(an.x))
print("Type", type(an.party))
```

Możemy użyć dir(), żeby sprawdzić wszystkie "możliwości" naszej nowo utworzonej klasy.

```
$ python party3.py
Type <class '__main__.PartyAnimal'>
Dir  ['__class__', ... 'party', 'x']
Type <class 'int'>
Type <class 'method'>
```

Sprawdź dir() z ciągiem znaków

```
>>> x = 'Hej tam'
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

Cykl życia obiektu

https://pl.wikipedia.org/wiki/Konstruktor_(programowanie_obiektowe)

Cykl życia obiektu

- Obiekty są tworzone, używane i porzucane
- Mamy specjalne bloki kodu (metody), które wywołujemy:
  - w momencie tworzenia (konstruktor)
  - w momencie niszczenia (destruktor)
- Konstruktory są używane bardzo często
- Destruktorów używa się rzadko

Konstruktor

Głównym celem konstruktora jest poprawne ustawienie wartości początkowych niektórych zmiennych w chwili tworzenia obiektu.

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('Jestem tworzony')

   def party(self) :
     self.x = self.x + 1
     print('Jak na razie', self.x)

   def __del__(self):
     print('Jestem niszczony', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an zawiera', an)
```

```
$ python3 party4.py
Jestem tworzony
Jak na razie 1
Jak na razie 2
Jestem niszczony 2
an zawiera 42
```

Konstruktor i destruktor są opcjonalne. Konstruktora zazwyczaj używa się do ustawienia wartości zmiennych. Destruktora używa się rzadko.

Konstruktor

W programowaniu obiektowym konstuktor to specjalny blok kodu w klasie, wywoływany przy tworzeniu obiektu

https://pl.wikipedia.org/wiki/Konstruktor_(programowanie_obiektowe)

Wiele instancji

- Możemy tworzyć wiele obiektów – klasa jest szablonem dla obiektu
- Możemy zapisać każdy obiekt w jego własnej zmiennej
- Nazywamy to wieloma instancjami tej samej klasy
- Każda instancja ma swoją własną kopię zmiennych instancji

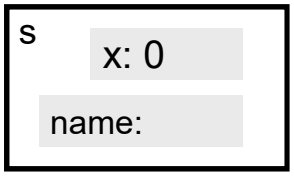
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name, '- utworzenie')

   def party(self) :
     self.x = self.x + 1
     print(self.name, '- zliczenie imprezek -', self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

Konstruktory mogą mieć dodatkowe parametry. Można ich użyć do ustawienia zmiennych dla konkretnej instancji klasy (czyli konkretnego obiektu).

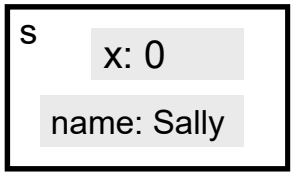
party5.py

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name, '- utworzenie')

   def party(self) :
     self.x = self.x + 1
     print(self.name, '- zliczenie imprezek -', self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

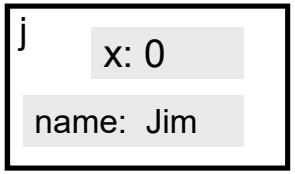
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name, '- utworzenie')

   def party(self) :
     self.x = self.x + 1
     print(self.name, '- zliczenie imprezek -', self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```




```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name, '- utworzenie')

   def party(self) :
     self.x = self.x + 1
     print(self.name, '- zliczenie imprezek -', self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```



Mamy dwie niezależne instancje



```
class PartyAnimal:
   x = 0                                        Sally - utworzenie
   name = ""                                    Jim - utworzenie
   def __init__(self, nam):                     Sally - zliczenie imprezek - 1
     self.name = nam                            Jim - zliczenie imprezek - 1
     print(self.name, '- utworzenie')           Sally - zliczenie imprezek - 2

   def party(self) :
     self.x = self.x + 1
     print(self.name, '- zliczenie imprezek -', self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

Dziedziczenie

https://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

Dziedziczenie

- Tworząc nową klasę, możemy ponownie wykorzystać istniejącą klasę, dziedzicząc wszystkie jej możliwości, a następnie dodając kilka nowych
- To inna forma zapisania i ponownego użycia
- Napisz raz – wykorzystaj wiele razy
- Nowa klasa (potomna) ma wszystkie możliwości starej klasy (źródłowej) – a może mieć nawet więcej

Terminologia: Dziedziczenie

‘Klasy pochodne’ są bardziej wyspecjalizowane od bazowych, po których dziedziczą atrybuty i zachowania. Mogą też otrzymywać własne.

https://pl.wikipedia.org/wiki/Programowanie_obiektowe

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
        self.name = nam
        print(self.name, "- utworzenie")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "- zliczenie imprezek -", self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "punkty", self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan to klasa rozszerzająca PartyAnimal. Ma wszystkie możliwości PartyAnimal i dodatkowo własne.


```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
        self.name = nam
        print(self.name, "- utworzenie")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "- zliczenie imprezek -", self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "punkty", self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```


s

x:
name: Sally


```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
        self.name = nam
        print(self.name, "- utworzenie")

    def party(self) :
        self.x = self.x + 1
        print(self.name, "- zliczenie imprezek -", self.x)

class FootballFan(PartyAnimal):
    points = 0
    def touchdown(self):
        self.points = self.points + 7
        self.party()
        print(self.name, "punkty", self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```


j

x:
name: Jim
points:

Definicje

- Klasa – szablon
- Atrybut – zmienna wewnątrz klasy
- Metoda – funkcja wewnątrz klasy
- Obiekt – konkretna instancja klasy
- Konstruktor – kod wykonywany przy tworzeniu obiektu
- Dziedziczenie – możliwość rozszerzenia klasy, aby stworzyć nową

Podsumowanie

- Programowanie obiektowe to bardzo ustrukturyzowane podejście do ponownego użycia kodu
- Możemy grupować dane i funkcjonalności i tworzyć wiele niezależnych instancji klas

Dodatkowe informacje o źródłach

- "Snowman Cookie Cutter" autorstwa Didriks na licencji CC BY https://www.flickr.com/photos/dinnerseries/23570475099
- Zdjęcie z programu telewizyjnego *Lassie*. Lassie watches as Jeff (Tommy Rettig) works on his bike jest w Domenie publicznej https://en.wikipedia.org/wiki/Lassie#/media/File:Lassie_and_Tommy_Rettig_1956.JPG

Podziękowania dla współpracowników

Autorstwo pierwszej wersji: Charles Severance, University of Michigan School of Information

Polska wersja powstała z inicjatywy Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu

Tłumaczenie: Agata i Krzysztof Wierzbiccy, EnglishT.eu

... wstaw tu nowych współpracowników i tłumaczy