Wstęp

Grzegorz Kowzan

Ogólnie (1)

Python to interpretowany[1] wieloparadygmatowy język wysokiego poziomu. Wspiera programowanie:

- funkcyjne,
- obiektowe,
- proceduralne,

nie podchodząc do żadnego dogmatycznie. Najczęściej wymieniane zalety to:

- łatwość w prototypowaniu, wygodna praca w powłoce interaktywnej,
- bogactwo biblioteki standardowej i bibliotek zewnętrznych,
- łatwość integracji z C/C++ i Fortranem[2],
- przejrzystość kodu.

---

[1] W zależności od implementacji: `https://wiki.python.org/moin/PythonImplementations`

[2] `ctypes` z biblioteki standardowej albo `cython`

Ogólnie (2)

Python jest językiem *dynamicznie typowanym*, tj. typ zmiennej jest określany dopiero w momencie jej użycia. Często określany mianem *duck typing* (*jeśli chodzi jak kaczka i kwacze jak kaczka, to musi być kaczką*), tj. niezalecane jest sprawdzanie typu przekazywanych obiektów, tylko czy zachowują się tak jak się spodziewamy.

Python posiada obecnie dwie aktywne gałęzie: 2 – starszą i 3 – nowszą. Starsza jest obecnie bardziej popularna, lecz wiele projektów jest kompatybilna z obydwoma i w najbliższej przyszłości dalej obydwie będą używane. Na zajęciach używamy wersji 3.4.

Istotne różnice między wersjami to między innymi:

- duża zmiana w obsłudze łańcuchów znaków - ta w 3 jest uważana za lepszą,
- `print` jako funkcja,
- dzielenie operatorem / zwraca zawsze `floata`,
- upowszechnienie iteratorów.

Szczęśliwie możliwe jest wczytanie tych zmian do wersji 2 i pisanie kodu kompatybilnego między wersjami.

Kto używa Pythona

Poniższe oprogramowanie w dużej części jest napisanie w Pythonie:

- Dropbox
- Civilization IV
- Spotify
- Instagram
- Quora
- Django

Dzięki projektowi Django i innym często jest używania do tworzenia serwisów internetowych. Jednym z najbardziej znanych orędowników Pythona jest Google. Dzięki dostępności bibliotek NumPy, SciPy, AstroPy, Matplotlib jest co raz powszechniej używany zamiast Matlaba do analizy danych i obliczeń numerycznych.

Instalacja

Linux Już jest zainstalowany.

Windows Można czystego Pythona ze strony `python.org`. Polecam dystrybucje takie jak: Anaconda[3], WinPython[4].

Mac Os Zainstalować `homebrew` i potem Pythona.

W ogólności polecam *The Hitchhiker's Guide to Python* [5] jako przewodnik po ekosystemie Pythona.

W przypadku Linuksa wiele dodatkowych bibliotek znajduje się w repozytoriach używanej dystrybucji, w przypadku systemu Windows Anaconda oferuje wiele paczek z dodatkowych oprogramowaniem.

PyPI - the Python Package Index

Pod każdym systemem można używać programu pip[6] do pobierania praktycznie całego otwartego oprogramowania napisanego w Pythonie.

---

[3] `https://www.continuum.io/downloads`
[4] `https://winpython.github.io/`
[5] `http://docs.python-guide.org/en/latest/`
[6] `https://pypi.python.org/pypi`

Materiały

1. https://docs.python.org/3.4/library/index.html - biblioteka standardowa
2. https://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie - kurs podstawowy
3. https://learnpythonthehardway.org/book/ - od zera do programisty
4. http://www.scipy-lectures.org/ - kurs programowania naukowego
5. http://fizyka.umk.pl/~gkowzan/python/slajdy - slajdy z zajęć
6. http://www.pythonweekly.com/ - ciekawe artykuły co tydzień
7. http://www.google.com

Przykład

Kod

```
1  import codecs
2
3  WORD = 'tadeusz'
4
5  with codecs.open('pan-tadeusz.txt', 'r', 'utf-8') as f:
6      for i, line in enumerate(f, start=1):
7          line_split = [w.lower() for w in line.split()]
8          if any([WORD in w for w in line_split]):
9              print("{0:d}: {1:s}".format(i, line), end='')
```

Wynik

```
3: Pan Tadeusz czyli ostatni zajazd na Litwie
193:     «Dobrze mój Tadeuszu, (bo tak nazywano
196: Dobrze mój Tadeuszu, żeś się dziś nagodził
...
```

Typy

Typy proste

- `logiczne` `True`, `False`,
- `int` liczby całkowite o nieograniczonej precyzji,
- `float` `double` z C, szczegóły w `sys.float_info`,
- `complex` para `floatów`.

Typy sekwencyjne

- `list` lista,
- `tuple` krotka,
- `range` zakres liczbowy półotwarty,
- `str` łańcuch znaków jako sekwencja znaków z Unikodu,
- `bytes` ciąg bajtów, np. łańcuch znaków zakodowanych w standardzie UTF-8.

Inne

- `set` zbiór teoriomnogościowy (unikalne elementy),
- `dict` słownik (hash table).

Przykłady

```
    int 5412, -55, int(25.7), int("30")
  float .2, 1.0, 1.0e4, 1e7, float(10), float('7e13')
complex 1.0j, 2+7j, complex('4+5.2j')
   list [0, 7.0, 'kaczka', 1.0j],
        list(range(1, 100, 5))
  tuple (1, 8.0, 'pomidor'), (1, ), tuple([0.0, 1.0])
  range range(<start>, <end>, <step>)
    str ='tekst'=, ="tekst"=, ="""tekst wielolinijkowy"""=
  bytes b'Hello world'=, \\ =bytes([0x7f, 0x45]),
        b'\x7f\x45\'=, \\ ='Zażółć gęślą
        jaźń'.encode('utf-8')
    set {54.  25.1, (1, 'k')}, set([54.  25.1, (1,
        'k')]), set(range(10))
   dict {'jajecznica':  ['jajka', 'cebula', 'masło']},
        dict(jajecznica=['jajka', 'cebula', 'masło'])
```

Podstawy (1)

Deklaracja zmiennych

```
1  x = 4
2  print(type(x)) # można w każdej chwili sprawdzić typ
3  x = []
4  print(type(x))
5  x = range(0, 10, 2)
6  print(x, type(x)) # range to nie jest lista, ale można po tym iterować
7  print(list(x)) # trzeba rzutować na listę, krotkę, żeby dostać się do
    ↪  elementów
```

```
1  <class 'int'>
2  <class 'list'>
3  range(0, 10, 2) <class 'range'>
4  [0, 2, 4, 6, 8]
```

Wywoływanie funkcji

- najpierw argumenty pozycyjne,
- potem argumenty nazywane (są opcjonalne).

```
1  nazwa_funkcji(arg1, ..., argN, kw1=kwarg1, ..., kwN=kwargN)
```

Podstawy (2)

while

```
1  i = 0; s = "zażółć gęślą jaźń"
2  while s[i] != "ź":
3      i += 1
4  if i == s.index("ź"):
5      print("Znalazłem!")
```

```
1  Znalazłem!
```

print

- dowolna ilość argumentów pozycyjnych, spacje pomiędzy
  - separator między elementami można zmienić argumentem `sep`

```
1  print(1, 2, 3, 4, sep=', ')
```

```
1  1, 2, 3, 4
```

- domyślne „wygląd” elementu, który nie jest typu `str` to łańcuch znaków zwracany przez funkcję `str`, tj.

```
1  print(str(['kaczka', 7.0]))
2  print(['kaczka', 7.0])
```

da to samo

- znak wyświetlany po ostatnim argumencie określany jest przez argument `end`, domyślna wartość to `\n`, czyli znak nowej linii

Formatowanie łańcuchów znaków

docs.python.org/3.4/library/string.html#formatstrings

Kod

```
1  print("{1} {0} być po kolei".format(
2      "musi", "Nie") )
3  print("Masz na imię {name:s} i lat {age:d}".format(
4      name='Joffrey', age=13))
5  print("Liczba urojona: {0.real} + i{0.imag}".format(1.0+3.14j))
6  print("Współrzędne: x={punkt[0]:.2f}, y={punkt[1]:.2f}".format(
7      punkt=(1.1, 2.5)))
```

Wyniki

```
Nie musi być po kolei
Masz na imię Joffrey i lat 13
Liczba urojona: 1.0 + i3.14
Współrzędne: x=1.10, y=2.50
```

Instrukcje sterujące (1)

Instrukcja warunkowa

```
1  if <warunek>:
2      instrukcje
3  elif <warunek>:
4      instrukcje
5  else:
6      instrukcje
```

Pętla `while`

```
1  while <warunek>:
2      # instrukcje
3
4      # następna iteracja
5      continue
6
7      # wyjdź z pętli
8      break
```

Poniższe są równoważne `False`:

- `[]`, `( )`, `{}`, `=''=`, `0`, `0.0` - tj. puste: lista, krotka, słownik/zbiór, zero(`int`), zero(`float`)

Poza tym przydają się:

op. logiczne `or`, `and`, `not`,

op. porównania `<`, `<=`, `>`, `!=`, `==`

op. zawierania `in`, `not in`

```
1  print(1 in range(1, 5))
2  print('a' in 'abażur')
3  print('a' in ['abażur'])
4  print(-1 in {7, 5, 8})
```

```
True
True
False
False
```

Instrukcje sterujące (2)

Pętla for

```
1  for w in 'grzechotka':
2      print('{}_'.format(w), end='')
3  print()
4  for el in [5, 7, 'żuraw', (8.02, 2e7)]:
5      print('{}; '.format(el), end='')
6  print()
7  for k, v in {'imie': 'Masdamer', 'nazwisko': 'Tylżycki',
8               'wiek': 5, 'kolor': 'zielony'}.items():
9      print('{k}: {v}'.format(k=k, v=v))
```

```
1  g_r_z_e_c_h_o_t_k_a_
2  5; 7; żuraw; (8.02, 20000000.0);
3  nazwisko: Tylżycki
4  wiek: 5
5  kolor: zielony
6  imie: Masdamer
```

Instrukcje sterujące (3)

range

```
1  print(list(range(10)))
2  print(list(range(1, 10+1, 2)))
3  print(list(range(9, -1, -1)))
4  print(list(reversed(range(10))))
```

```
1  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2  [1, 3, 5, 7, 9]
3  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
4  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Klasyczny for

```
1  l1 = list(range(0, 20, 2))
2  for i in range(len(l1)):
3      print(l1[i], end='; ')
```

```
1  0; 2; 4; 6; 8; 10; 12; 14; 16; 18;
```

Funkcje wbudowane

`https://docs.python.org/3.4/library/functions.html`

`abs`	wartość bezwzględna/moduł liczby zespolonej
`round`	
`all (any)`	`True`, jeśli wszystkie (którykolwiek) elementy są `True`
`min (max)`	zwraca wartość minimalną (maksymalną) typu sekwencyjnego
`sum`	zwraca sumę elementów typu sekwencyjnego (działa nie tylko dla liczb)
`chr (ord)`	zwraca znak (numer) odpowiadający podanemu numerowi (znaku) z zestawu znaków Unicode (od 1 do 127 to samo co ASCII)
`len`	zwraca ilość elementów typu sekwencyjnego
`sorted (reversed)`	zwraca posortowaną (odwróconą) kopię typu sekwencyjnego

input, rzutowanie

Rzutowanie Nazwa typu jest też funkcją zwracającą element danego typu. Można nią rzutować wyrażenia innego typu na pożądany typ.

```
1  print(int(46.7)) # nie zaokrągla!
2  print(float('6.626e-34'))
3  print(int('0xff', 16))
4  print(complex(2, 7))
5  print(complex('2+6j')) # ale nie complex('2 + 6j'), nie może być
   ↪  spacji!
6  print(list('pomidor'))
```

```
1  46
2  6.626e-34
3  255
4  (2+7j)
5  (2+6j)
6  ['p', 'o', 'm', 'i', 'd', 'o', 'r']
```

input Pobiera jednolinijkowy łańcuch znaków od użytkownika.

```
1  x = input("Podaj wartość: ") # zwraca str
2  x = int(input("Podaj liczbę heksadecymalną: "), 16) # można od razu
   ↪  rzutować
```

Zadania z pierwszej listy

http://fizyka.umk.pl/~gkowzan/python/zadania/intro-zadania.pdf

Typy sekwencyjne - operatory

https://docs.python.org/3.4/library/stdtypes.html#sequence-types-list-tuple-range

```
1   t1 = tuple(range(0, 10, 2))
2   print("t1:", t1)
3   print("2 in t1:", 2 in t1)
4   print("t1 + (1, 3)", t1 + (1, 3))
5   print("t1 * 2:", t1 * 2)
6   print("t1.index(8):", t1.index(8))
7   print("t1.count(8):", t1.count(8))
8   print("t1[0:2]:", t1[0:2])
9   print("t1[::-1]:", t1[::-1])
10  pierwszy, *srodek, ostatni = t1
11  print(pierwszy, srodek, ostatni)
```

```
1   t1: (0, 2, 4, 6, 8)
2   2 in t1: True
3   t1 + (1, 3) (0, 2, 4, 6, 8, 1, 3)
4   t1 * 2: (0, 2, 4, 6, 8, 0, 2, 4, 6, 8)
5   t1.index(8): 4
6   t1.count(8): 1
7   t1[0:2]: (0, 2)
8   t1[::-1]: (8, 6, 4, 2, 0)
9   0 [2, 4, 6] 8
```

Typy sekwencyjne - listy, krotki

Listy

```
1   l1 = list(range(1, 12, 2))
2   l1[-1] = 99
3   print('1:', l1)
4   del l1[-1]
5   print('2:', l1)
6   l1.append(-1)
7   print('3:', l1)
8   l1.extend([7, 99])
9   print('4:', l1)
```

```
1   1: [1, 3, 5, 7, 9, 99]
2   2: [1, 3, 5, 7, 9]
3   3: [1, 3, 5, 7, 9, -1]
4   4: [1, 3, 5, 7, 9, -1, 7, 99]
```

Krotki

```
1   t1 = (5, 10, '15')
2   t1[-1] = 'nie uda się'
```

```
1   Traceback (most recent call last):
2     File "<stdin>", line 2, in
       ↪   <module>
3   TypeError: 'tuple' object does not
     ↪   support item assignment
```

Iteracja po krotkach

```
1   l1 = [(1, 2), (3, 4), (5, 6)]
2   for i, j in l1:
3       print(i, j)
```

```
1   1 2
2   3 4
3   5 6
```

Numerowanie

```
1   l1 = [(1, 2), (3, 4), (5, 6)]
2   for i, t in enumerate(l1):
3       print(i, t, sep=': ')
```

```
1   0: (1, 2)
2   1: (3, 4)
3   2: (5, 6)
```

Łączenie w krotki

```
1   l1 = [1, 2, 3]; l2 = [4, 5, 6]
2   for t in zip(l1, l2):
3       print(t)
```

```
1   (1, 4)
2   (2, 5)
3   (3, 6)
```

Typy sekwencyjne - krotki

Krotki

- często zawierają elementy różnego typu
- krotka stanowi określoną całość

```
1  import sys
2  print(platform.libc_ver()) # krotka: (nazwa, wersja)
```

```
1  ('glibc', '2.9')
```

powyżej funkcja zawsze zwraca krotkę o określonej ilości elementów i określonym znaczeniu każdego elementu

- stosowane jako rekordy

```
1  k = ('Jan', 'Kowalski', 3, 45.81, 2014)
2  print("{0[0]} {0[1]} ma {0[2]:d} zamówień na łączną kwotę
   ↪  {0[3]:.2f} zł. Jest z nami od {0[4]:d} roku.".format(k))
```

```
1  Jan Kowalski ma 3 zamówień na łączną kwotę 45.81 zł. Jest z nami od
   ↪  2014 roku.
```

- podobnie:

```
1  p1 = (1, 5); p2 = (-4, 5) # współrzędna x, potem y
2  d = ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)**0.5
3  print('d(p1, p2) =', d)
```

```
1  d(p1, p2) = 5.0
```

Typy sekwencyjne - odwzorowania list

Odwzorowania list

Zwięzły zapis pętli `for` odwzorowującej jedną listę w drugą listę, operując element po elemencie.

```
1   l1 = [x*7 for x in range(1, 5+1)]
2   l2 = []
3   for x in range(1, 5+1):
4       l2.append(x*7)
5   print(l1)
6   print(l2)
```

```
1   [7, 14, 21, 28, 35]
2   [7, 14, 21, 28, 35]
```

Opcjonalne filtrowanie elementów.

```
1   l1 = [x for x in range(1, 50+1) if not x % 10]
2   l2 = []
3   for x in range(1, 50+1):
4       if not x % 10:
5           l2.append(x)
6   print(l1)
7   print(l2)
```

```
1   [10, 20, 30, 40, 50]
2   [10, 20, 30, 40, 50]
```

Typy sekwencyjne - łańcuchy znaków

```
1  print("6" * 10) # nie zwróci 60
2  print("7" + "87" + "kot") # nie jest zbyt wydajne, każdy + implikuje
    ↪  tworzenie obiektu tymczasowego
3  print(', '.join(["7", "87", "kot"])) # analogicznie do print(s1, ...,
    ↪  sN, sep=', ')
4  print("5, 7, kaczka, pomidor".split())
5  print(''.join([w.upper() for w in "samogłoski" if w in "aeiou"]))
```

```
1  6666666666
2  787kot
3  7, 87, kot
4  ['5,', '7,', 'kaczka,', 'pomidor']
5  AOOI
```

Łańcuchy znaków mają wiele użytecznych metod:
https://docs.python.org/3.4/library/stdtypes.html#text-sequence-type-str

Typy sekwencyjne - zbiory

https://docs.python.org/3.4/library/stdtypes.html#set-types-set-frozenset

- tylko unikalne wartości

```
1  s1 = set([1, 87, 87, 'kaczka'])
2  s2 = {1, 87, 87, 'kaczka'}
3  print(s1 == s2)
4  print(s1)
```

```
1  True
2  {'kaczka', 1, 87}
```

- przykłady

```
1  s1 = {1, 87, 87, 'kaczka'}
2  s2 = {1, 572, 'pomidor'}
3  print(s1.isdisjoint(s1))
4  print(s1 & s2) # przekrój
5  print(s1 | s2) # suma
6  print(s1 - s2) # różnica zbiorów
7  s1.update(s2); print(s1) # s1 = s1 | s2
```

Typy sekwencyjne - zadania (1)

http://fizyka.umk.pl/~gkowzan/python/zadania/sekwencje-zadania.pdf

Typy sekwencyjne - słowniki

https://docs.python.org/3.4/library/stdtypes.html#mapping-types-dict

Odwzorowania klucz → wartość. Klucze nie są uporządkowane.

```
1  print(dict(zero=0, jeden='1', trzy='trzy'))
2  print({'zero': 0, 'jeden': '1', 'trzy': 'trzy'})
3  print(dict([('zero', 0), ('jeden', '1'), ('trzy', 'trzy')]))
4  print(dict(zip(('zero', 'jeden', 'trzy'), (0, '1', 'trzy'))))
```

```
1  {'zero': 0, 'trzy': 'trzy', 'jeden': '1'}
2  {'zero': 0, 'jeden': '1', 'trzy': 'trzy'}
3  {'zero': 0, 'jeden': '1', 'trzy': 'trzy'}
4  {'zero': 0, 'jeden': '1', 'trzy': 'trzy'}
```

```
1  d = {'zero': 0, 'jeden': '1', 'trzy': 'trzy'}
2  print(d['zero'])
3  print(d.get('nie istnieje', 'wartość jeśli nie istnieje'))
```

```
1  0
2  wartość jeśli nie istnieje
```

Typy sekwencyjne - słowniki (2)

```
1   d = {'zero': 0, 'jeden': '1', 'trzy': 'trzy'}
2   print('zero' in d.keys()) # klucz
3   print('1' in d.values()) # wartość
4   print(('jeden', '1') in d.items()) # para (klucz, wartość)
```

```
1   True
2   True
3   True
```

Otwieranie plików

Dane binarne:

`open(<ścieżka>, <tryb>)`

Dane tekstowe (automatyczne dekodowanie):

`codecs.open(<ścieżka>, <tryb>, <kodowanie>)`

`https://docs.python.org/3.4/library/functions.html#open`

```
1  import codecs
2  f = codecs.open('pan-tadeusz,txt', 'r', 'utf-8')
3  ...
4  f.close()
```

Jeśli między open a `f.close()` wystąpi (nieobsłużony) błąd, to plik nie zostanie zamknięty!

```
1  import codecs
2  with codecs.open('pan-tadeusz,txt', 'r', 'utf-8') as f:
3      ...
```

Plik jest automatycznie zamykany.

Otwieranie plików (2)

Ważniejsze metody

`f.read()` zwraca całą zawartość pliku

`f.readline()` zwracał pojedynczą linie, przechodzi do następnej

`f.readlines()` zwraca zawartość plików jak listę linii

`f.write()` zapisuje podany ciąg bajtów (lub łańcuch znaków) do pliku

`f.writelines()` zapisz do pliku listę stringów; jeśli chcemy żeby każdy element listy był w nowej linijce, to muszą być zakończone znakami nowej linii `\n`

Iterowanie po linijkach

```
1  with codecs.open('pan-tadeusz.txt', 'r', 'utf-8') as f:
2      for line in f:
3          print(line) # iterujemy linia po linii
4      for line in f.readlines():
5          print(line) # to samo, tylko readline od razu zwraca cały plik
```

Funkcje

Definiowanie funkcji

```
1  def nazwa_funkcji(arg1, arg2, kwarg1=a, kwarg2=b):
2      ...
3
4      return ret1
```

- `arg1`, i `arg2` są wymagane przy wywoływaniu
- `kwarg1`, `kwarg2` są opcjonalne (podaliśmy domyślne wartości)
- jeśli nie ma `return ret`, to jest zwracana wartość `None`.

Zwracanie wielu wartości

```
1  def f1(x, y):
2      return x*5, y*5
```

Zwracam wiele wartości, tak naprawdę krotkę, którę mogę przypisać

```
1  x1, y1 = f1(9, 78)
```

Funkcje (2)

Argumenty zbiorcze

```
1  def f1(arg1, arg2, *args, kwarg1=False)
```

Pierwsze dwa argumenty będą przypisane do arg1, arg2, odpowiednio. Każdy kolejny będzie umieszczony w liście args.

```
1  def my_print(*args):
2      for arg in args:
3          print(arg, end="; ")
4
5  my_print(1, 97, 65, "*&sdfn")
```

```
1; 97; 65; *&sdfn;
```

Funkcje jako argumenty

Funkcje można przekazywać jako argumenty innych funkcji.

```
1  l = ['aa', 'a', 'aaaaa', 'aaaa']
2  print(sorted(l, key=len)) # zamiast elementów listy porównuje wyniki
    ↪  wykonania len na każdym elemencie
```

```
1  ['a', 'aa', 'aaaa', 'aaaaa']
```

Zadania

http://fizyka.umk.pl/~gkowzan/python/zadania/sekwencje2-zadania.pdf

Importowanie, pakiety

Struktura pakietu

```
html/
    __init__.py
    parser.py
    entities.py
```

Zawartość pakietu

- html/__init__.py:
  html.escape,
  html.unescape
- html/parser.py:
  html.parser.HTMLParser,
  html.parser.HTMLParseError
- html/entities.py:
  html.entities.html5,
  html.entities.entitydefs,
  ...

```
1  import html
2  html.escape(r'\')  # ok
3  html.entities.html5  # nie
   ↪ zadziała, html.entities
   ↪ niezaimportowane
4  import html.entities.html5  # też
   ↪ nie zadziała
5  from html.entities import html5  #
   ↪ ok
6  import html.entities
7  html5 = html.entities.html5  # ok,
   ↪ wyjdzie na to samo
```

Importowanie, pakiety (2)

moduł plik `.py` ze zdefiniowanymi funkcjami, klasami, stałymi, itd.

pakiet katalog z modułami i/lub podpakietami

Importowanie

`import <pakiet/moduł> (as <nazwa>)` można zaimportować pakiety lub moduły

`from <pakiet/moduł> import <pakiet/moduł/\dots> (as <nazwa>)` można zaimportować dowolny obiekt z podanego pakietu/modułu (zmienną, funkcję, klasę, moduł)

Co się dzieje podczas importowania

test.py

```
1  const1 = 'pomidor'
2  def funkcja():
3      return const1
4  print(funkcja)
```

Importowanie, pakiety (3)

Moduły i pakiety są szukane w:

1. bieżącym katalogu lub katalogu, w którym znajduje się uruchamiany skrypt,
2. ścieżkach zdefiniowanym w zmiennej środowiskowej `PYTHONPATH`,
3. ścieżce ustalonej podczas kompilacji Pythona.

```
1  import sys, pprint
2  pprint.pprint(sys.path[4:7])  # zawartość PYTHONPATH, można edytować jak
    ↪  każdą listę
```

```
1  ['/home/gkowzan/documents/nauka/cs/python/MODULES',
2   '/home/gkowzan/documents/nauka/fizyka/dydaktyka/python/WYKLADY/intro',
3   '/home/gkowzan/.pyenv/versions/3.4.3/lib/python34.zip']
```

Zadania

http://fizyka.umk.pl/~gkowzan/python/zadania/moduly-zadania.pdf

Programowanie obiektowe

Obiekty

Dane + zachowania, czyli dane + metody operujące na nich.

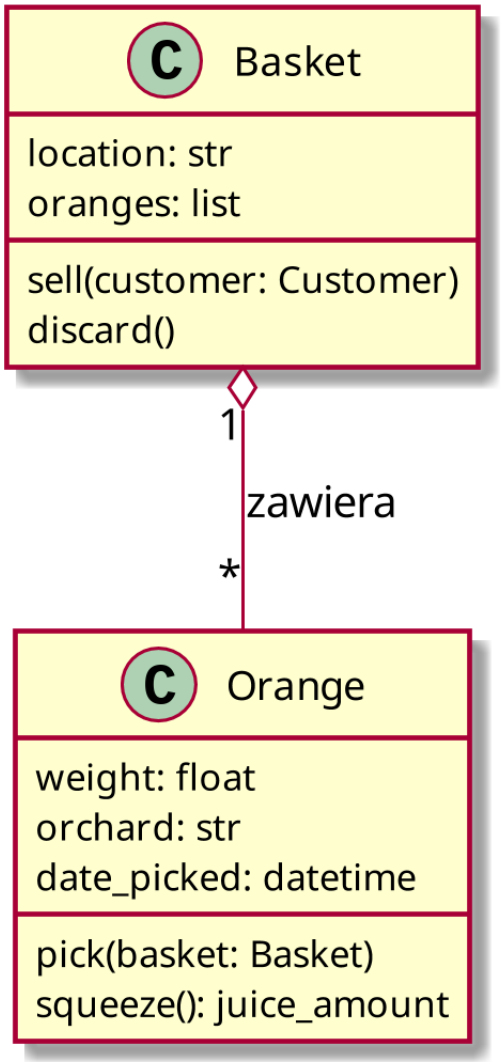
Klasy

Klasy to wzorce, przepisy jak utworzyć obiekt danego typu. Poszczególne obiekty to instancje klasy. Klasa określa metody i atrybuty obiektu, ale każdy obiekt (instancja) może mieć własne, niezależne wartości tych atrybutów.

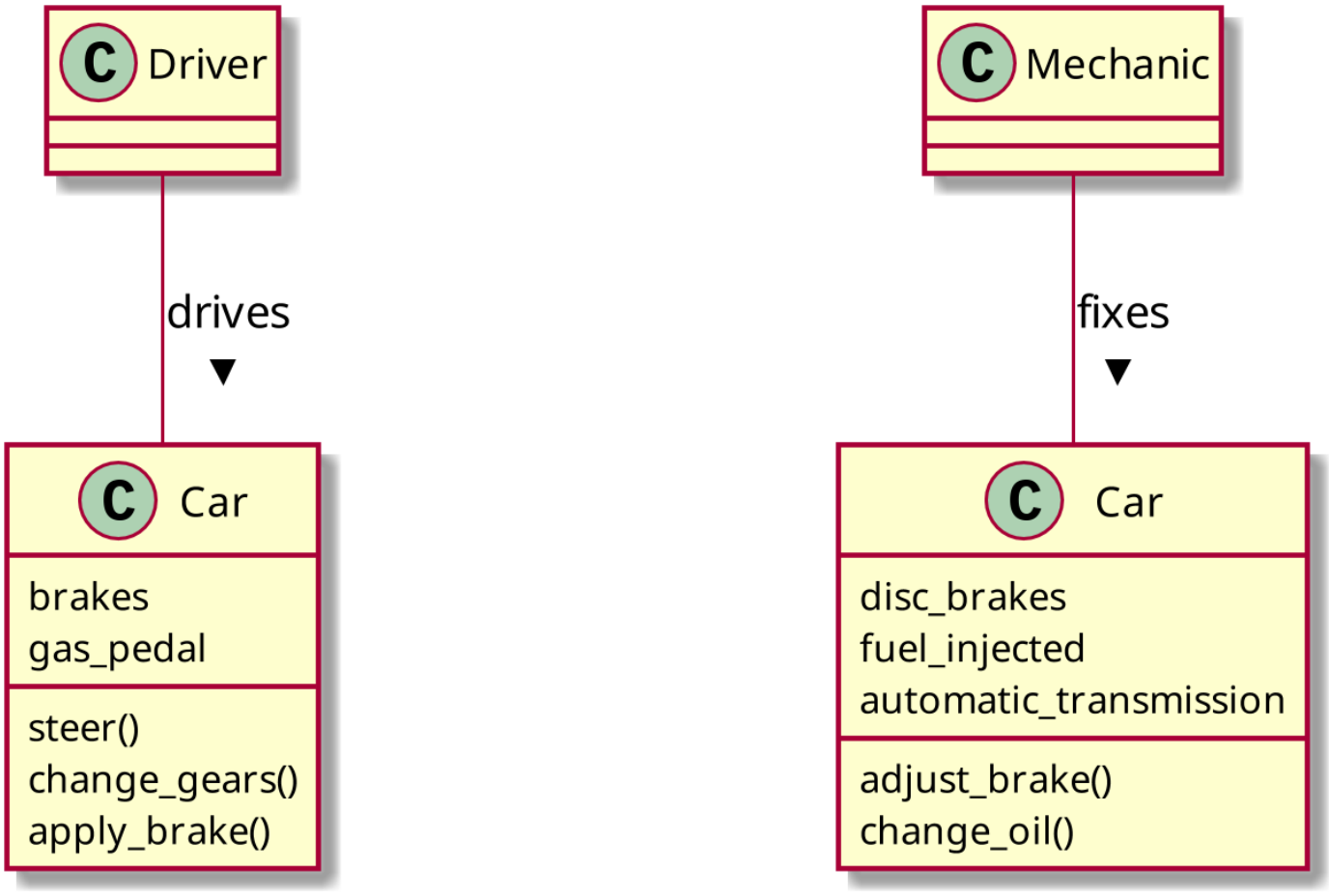
Relacje

agregacja	obiekt zawiera inny obiekt, obiekt zawierany istnieje niezależnie
kompozycja	obiekt zawiera inny obiekt, obiekt zawierany powstaje i ginie razem z zawierającym
dziedziczenie	klasa ma metody i atrybuty zdefiniowane wcześniej w innej klasie, może je nadpisać własnymi

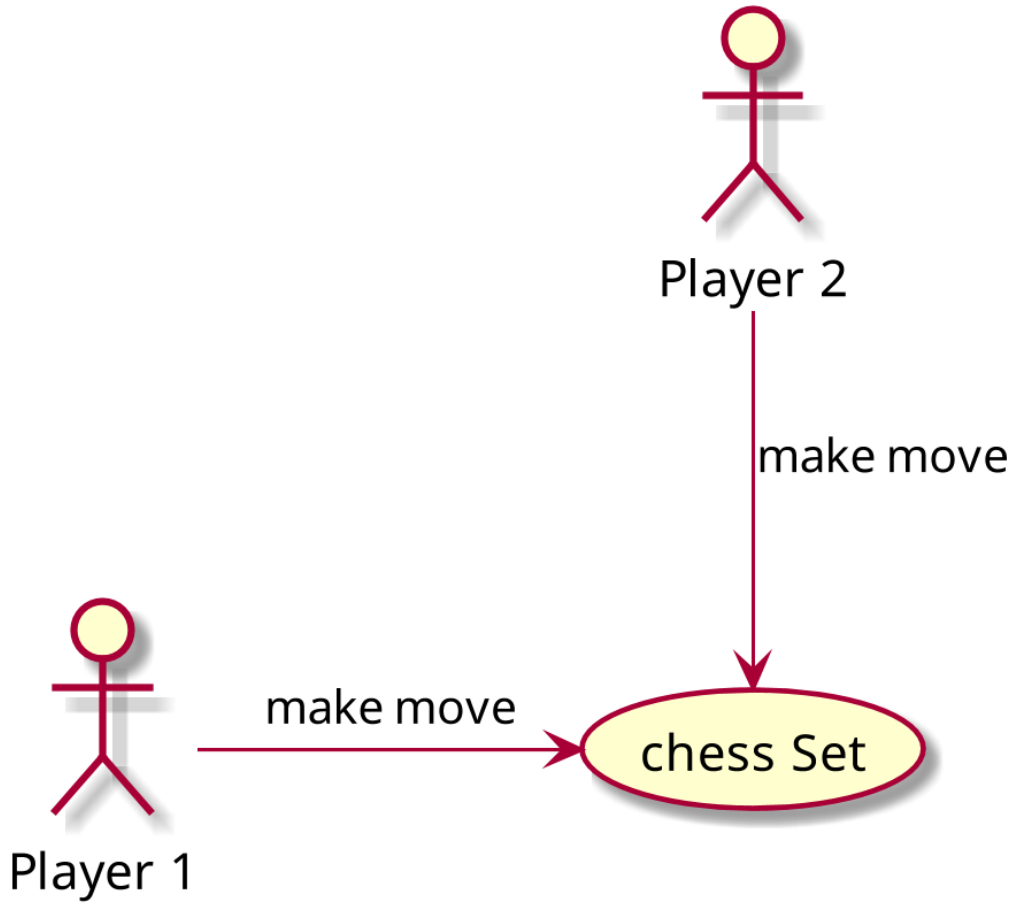
Programowanie obiektowe—przykład (1)



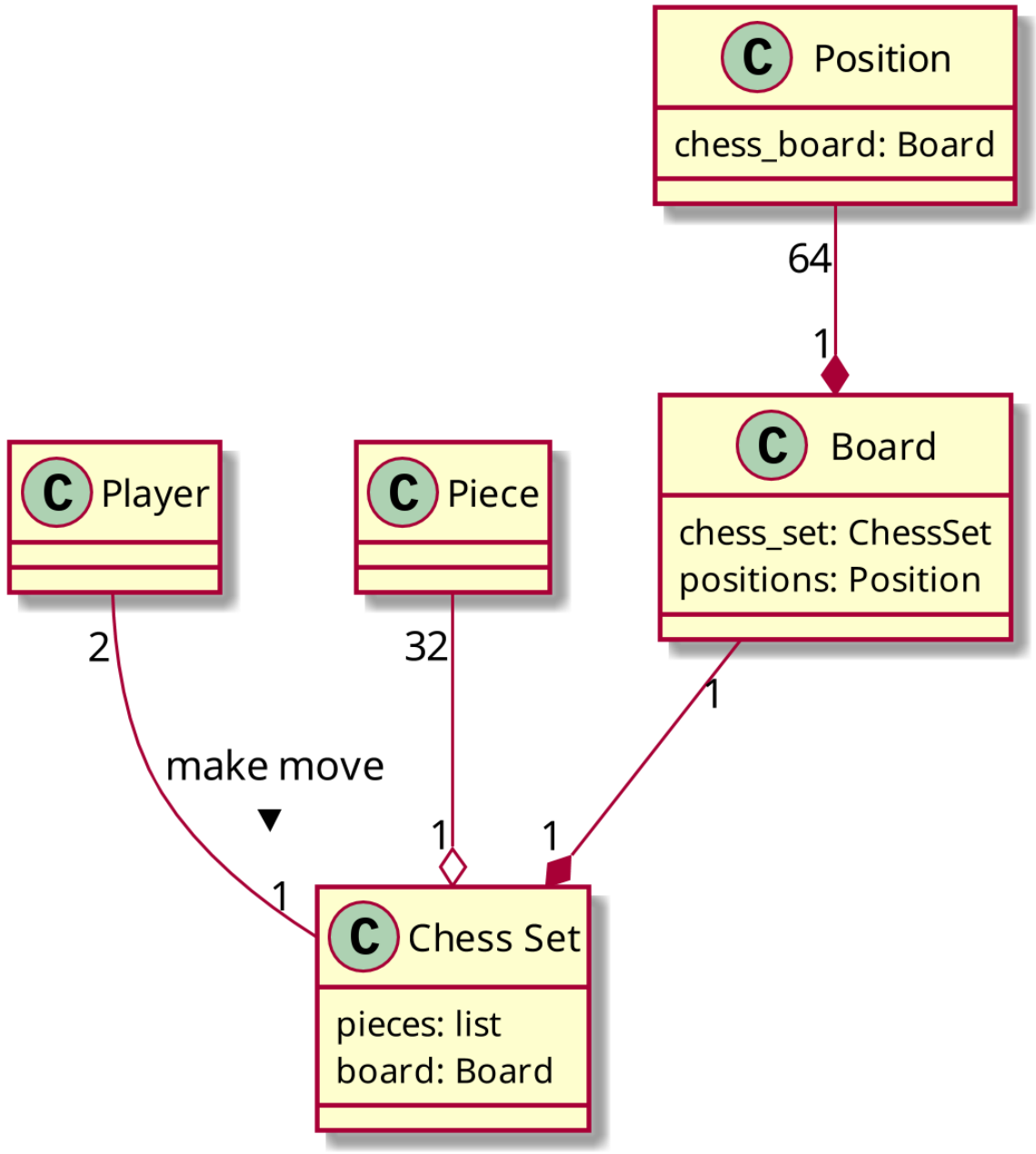
Programowanie obiektowe—enkapsulacja



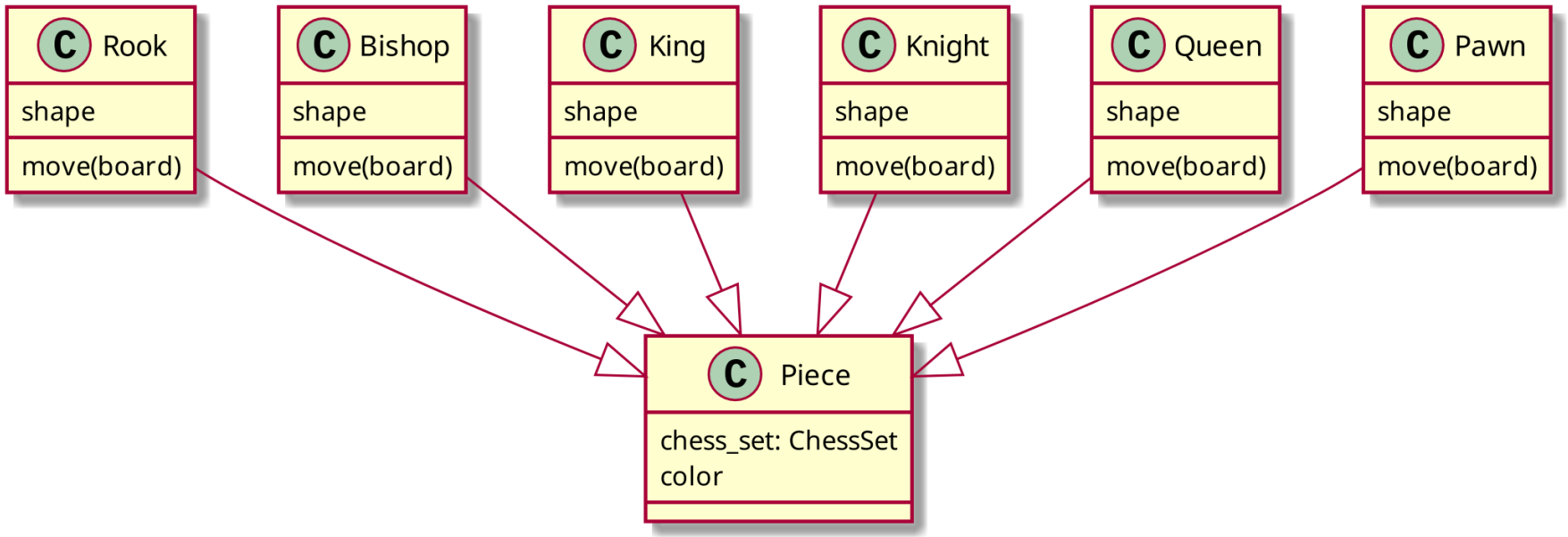
Programowanie obiektowe—kompozycja, agregacja (1)



Programowanie obiektowe—kompozycja, agregacja (2)



Programowanie obiektowe–dziedziczenie



Programowanie obiektowe w Pythonie (1)

Najprostsza klasa

```
1  class SimpleClass:
2      pass
3  a = SimpleClass()
4  print(a)
5  b = SimpleClass()
6  print(b)
```

```
1  <__main__.SimpleClass instance at 0x7f09c66e88c0>
2  <__main__.SimpleClass instance at 0x7f09c66e8908>
```

Dodawanie atrybutów

```
1  class Point:
2      pass
3  p1 = Point(); p2 = Point()
4  p1.x = 5; p1.y = 4
5  p2.x = 3; p2.y = 6
6  print(p1.x, p1.y)
7  print(p2.x, p2.x)
```

```
1  5 4
2  3 3
```

Programowanie obiektowe w Pythonie (2)

Definicja klasy

```
1   import math as m
2   class Point:
3       def __init__(self, x=0.0,
        ↪   y=0.0):
4           self.move(x, y)
5
6       def move(self, x, y):
7           self.x = x
8           self.y = y
9
10      def reset(self):
11          self.move(0.0, 0.0)
12
13      def distance(self, p2):
14          return
            ↪   m.sqrt((self.x-p2.x)**2
            ↪   + (self.y-p2.y)**2)
```

Wykorzystanie klasy

```
1   p1 = Point()
2   p2 = Point(24.0, -5.0)
3   p1.move(-5.8, 6.0)
4   print("d(p1, p2):",
     ↪   p2.distance(p1))
5   p1.reset()
6   print("Norma p2:",
     ↪   p2.distance(p1))
```

```
1   >>> d(p1, p2):
     ↪   31.765389970847203
2   >>> Norma p2: 24.515301344262525
```

Zadania

http://fizyka.umk.pl/~gkowzan/python/zadania/obiektowe-zadania.pdf

Przeciążanie operatorów i funkcji wbudowanych

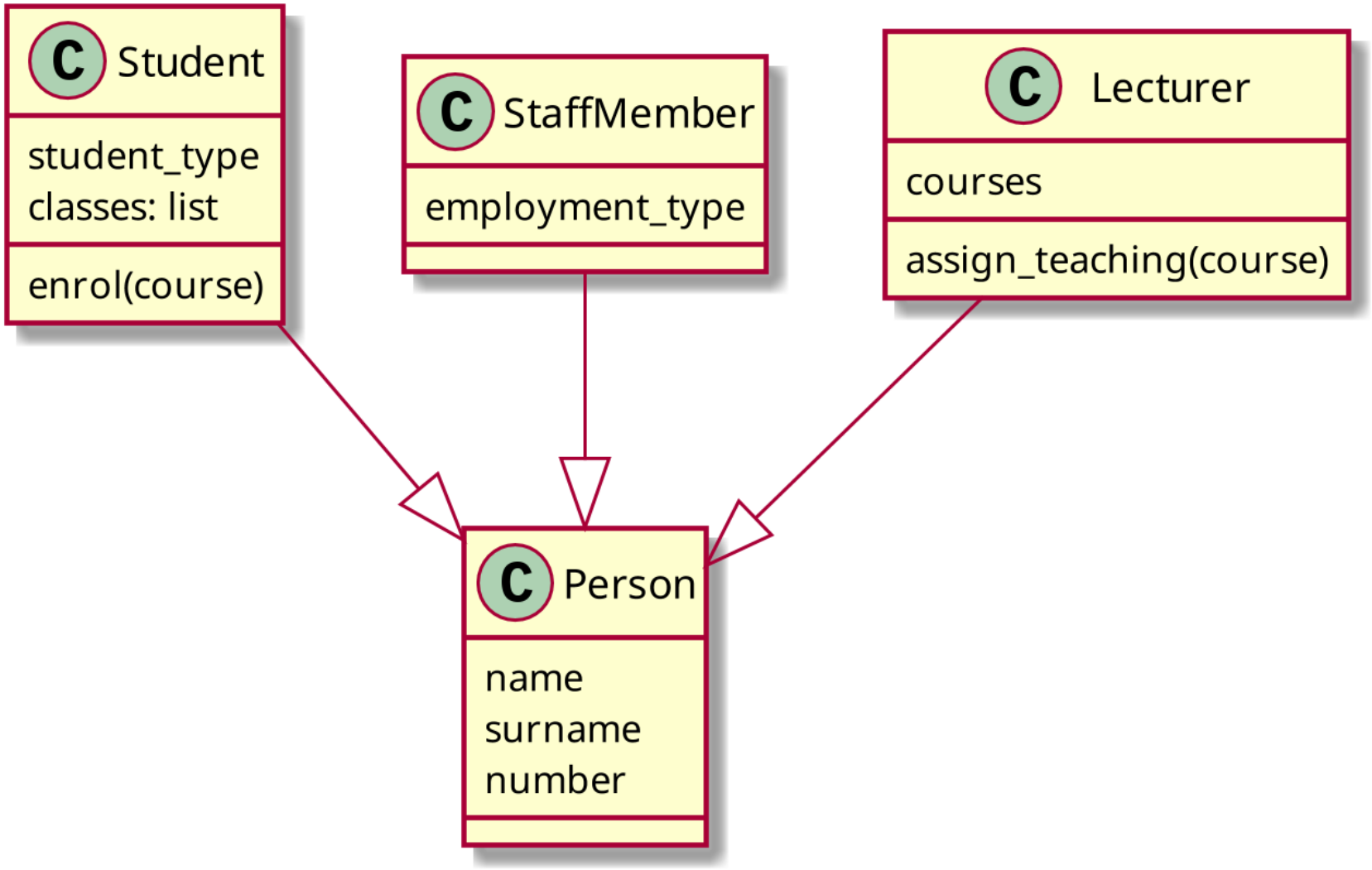
```python
import math as m
from numbers import Number
class Point:
    def __init__(self, x=0.0, y=0.0):
        self.move(x, y)

    def move(self, x, y):
        self.x = x
        self.y = y

    def __len__(self):
        # musi zwracać int
        return 2

    def __repr__(self):
        return "Point({0.x}, {0.y})".format(self)

    def __str__(self):
        # domyślnie to samo co __repr__
        return "(x={0.x}, y={0.y})".format(self)
```

```python
def __add__(self, other):
    if isinstance(other, Number):
        return Point(self.x + other, self.y + other)
    elif isinstance(other, Point):
        return Point(self.x + other.x, self.y + other.y)
    else:
        raise NotImplemented

def __neg__(self):
    return Point(-self.x, -self.y)

def __sub__(self, other):
    return self + -other

def __abs__(self):
    return m.sqrt(self.x**2 + self.y**2)

def __mul__(self, other):
    if isinstance(other, Number):
        return Point(self.x*other, self.y*other)
```

Przeciążanie operatorów i funkcji wbudowanych (2)

```
 1  from point import Point
 2  p1 = Point(1.0, 7.5)
 3  p2 = Point(6.1, -5.75)
 4
 5  print('p1 + p2 =', p1 + p2)
 6  print('p1 - p2 =', p1 - p2)
 7  print('p1 + 1.0 =', p1 + 1.0)
 8  print('p1*5 =', p1*5)
 9  print('abs(p1) =', abs(p1))
10  print('1.0 + p1 =', 1.0 + p1)
```

Dziedziczenie

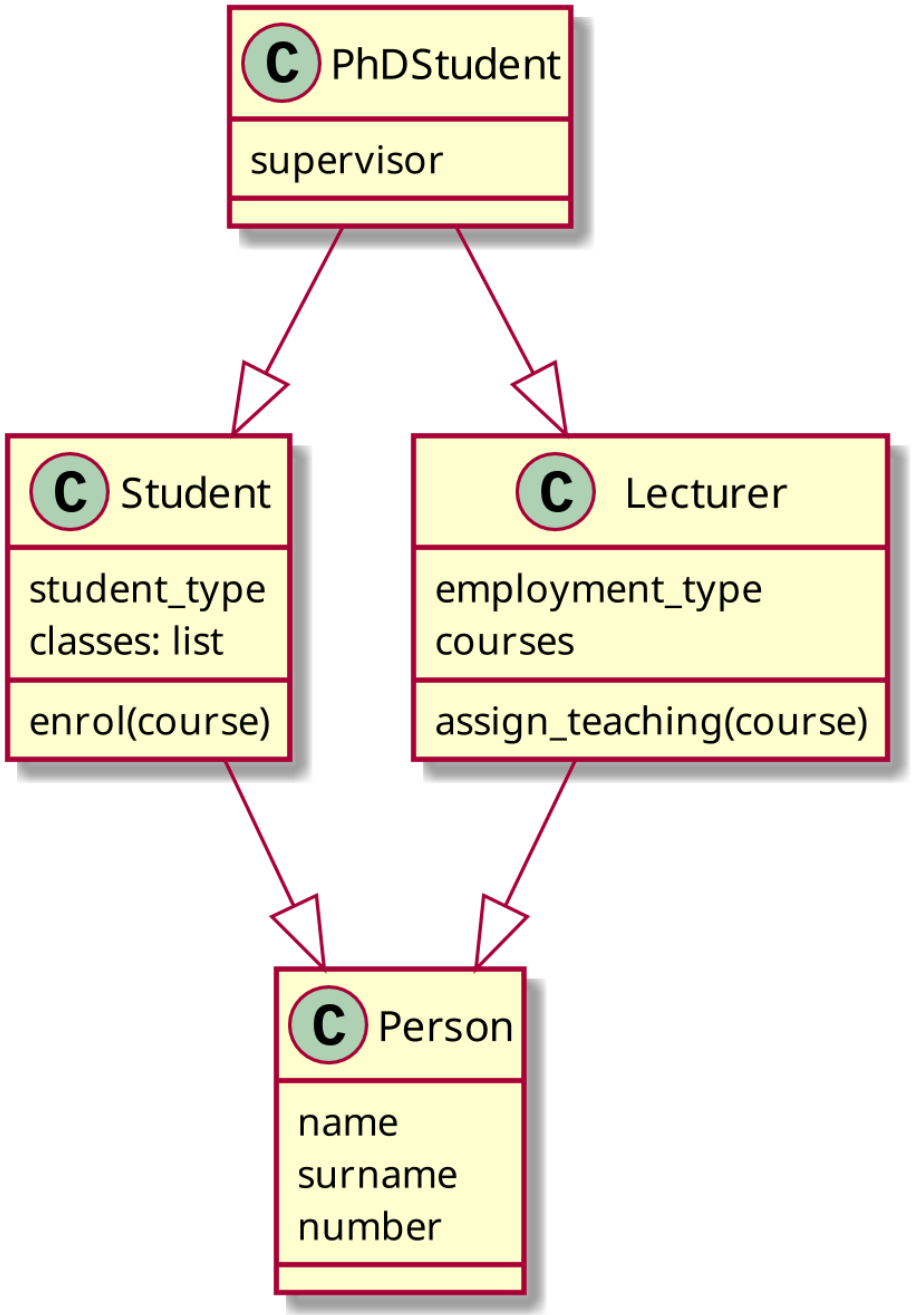


Dziedziczenie (2)

```
1   class Person:
2       def __init__(self, name,
        ↪   surname, number):
3           self.name = name
4           self.surname = surname
5           self.number = number
6
7   class Student(Person):
8       def __init__(self,
        ↪   student_type, *args,
        ↪   **kwargs):
9           self.student_type =
            ↪   student_type
10          self.classes = []
11          super(Student,
            ↪   self).__init__(*args,
            ↪   **kwargs)
12
13      def enrol(self, course):
14          self.classes.append(
15              course)
```

```
1   class StaffMember(Person):
2       def __init__(self,
        ↪   employment_type, *args,
        ↪   **kwargs):
3           self.employment_type =
            ↪   employment_type
4           super(StaffMember,
            ↪   self).__init__(*args,
            ↪   **kwargs)
5
6   class Lecturer(StaffMember):
7       def __init__(self, *args,
        ↪   **kwargs):
8           self.courses_taught = []
9           super(Lecturer,
            ↪   self).__init__(*args,
            ↪   **kwargs)
10
11      def assign_teaching(self,
        ↪   course):
12
            ↪   self.courses.append(course)
```

Diamentowe dziedziczenie



Dziedziczenie—mix-iny

```
1   class Person:
2       def __init__(self, name, surname, number):
3           self.name = name
4           self.surname = surname
5           self.number = number
6
7   class LearnerMixin:
8       def __init__(self):
9           self.classes = []
10
11      def enrol(self, course):
12          self.classes.append(course)
13
14  class TeacherMixin:
15      def __init__(self):
16          self.courses_taught = []
17
18      def assign_teaching(self, course):
19          self.courses_taught.append(course)
20
21  class Tutor(Person, LearnerMixin, TeacherMixin):
22      def __init__(self, *args, **kwargs):
23          super(Tutor, self).__init__(*args, **kwargs)
```

Zadania

http://fizyka.umk.pl/~gkowzan/python/zadania/obiektowe-zadania2.pdf