Politechnika Wrocławska

Inżynieria obrazów

Laboratorium nr 2

Wprowadzenie do języka Python

Szymon Datko & Mateusz Gniewkowski

szymon.datko@pwr.edu.pl , mateusz.gniewkowski@pwr.edu.pl

Wydział Elektroniki,

Politechnika Wrocławska

semestr letni 2020/2021

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Cel ćwiczenia

1. Zapoznanie się ze składnią języka Python.

2. Obeznanie się ze środowiskiem pracy.

Politechnika Wrocławska

Wprowadzenie do języka

Python

Politechnika Wrocławska

Czym jest Python?

- Ściśle mówiąc jest to **język** programowania.
  - Specyfikacja składni, semantyki i możliwych konstrukcji.
  - Do uruchomienia programów potrzebny jest **interpreter**.
  - Domyślną, referencyjną implementacją jest **CPython**.
- Bardzo szybko i dynamicznie rozwijający się (patrz: dokumenty **PEP**).
- Zasadniczo **obiektowy**, chociaż z elementami innych paradygmatów.
- Przejrzysta składnia, nastawiona na czytelność i brak zbędnych udziwnień.
- Mnogość dostępnych modułów do wszelakich zadań = popularność!

Politechnika Wrocławska

Skąd go wziąć?

- Źródła i instalator z oficjalnej strony – https://www.python.org/.
- Najczęściej dostępny jest także w menadżerach pakietów:
  - Arch: `pacman -Sy python`;
  - Ubuntu: `apt install python3`;
  - macOS: `brew install python3`.
- Przydatny będzie również dowolny edytor plików tekstowych:
  - **vim** jest zawsze dobrym wyborem ;-)
  - Visual Studio Code, PyCharm, Spyder.
- Popularne są także gotowe dystrybucje:
  - np. Anaconda (https://www.anaconda.com/)
    = Python oraz dodatkowe moduły i narzędzia do danologii :-)

Podstawy składni (1/2)

- Dostępny jest szereg podstawowych, wbudowanych funkcji: https://docs.python.org/3/library/functions.html.
- Dodatkowe funkcje można wczytać z modułów, albo definiować.
- Zmienne muszą być stworzone jawnie, ale są dynamicznie typowane.
- Komentarze w kodzie występują wyłącznie w wariancie liniowym.

```
a = 'Najpierw zmienna może być ciągiem znaków'
a = 2 + 2 * 2  # a chwilę później już liczbą!

print(a)
print(type(a))
print(dir(a))

from random import randint
help(randint)
```

Podstawy składni (2/2)

- Domyślnie pojedynczą instrukcję kończy nowa linia...
  - Chyba, że wcześniej otwarty był nawias, klamerka, itp.
  - Albo na końcu poprzedniej linijki wstawiono wtyłciach \ ;-)
- Bloki są wyznaczane za pomocą wcięć.
  - Po znaku dwukropka : zawsze następuje wcięcie!

```
a = 'Dosyć \
długa treść'

def surface(radius):
    return 3.14159 * radius**2

print(
    surface(
        3
))
```

Politechnika Wrocławska

Podstawowe kolekcje

- Lista:
  - elementy mogą być różnych typów, kolejność ustalona przy definicji,
  - przykład: `a = [1, 2, 3]`.
- Krotka:
  - niemodyfikowalny odpowiednik listy, odrobinę mniejszy narzut,
  - przykład: `a = (1, 2, 3)`.
- Zbiór:
  - nieuporządkowana kolekcja unikalnych elementów (brak powtórzeń),
  - przykład: `a = {1, 2, 3}`.
- Słownik:
  - nieuporządkowany zbiór par typu klucz-wartość, klucze unikalne,
  - przykład: `a = {'a': 1, 'b': 2, 'c': 3}`.

Instrukcje warunkowe

- Dostępna jest jedynie konstrukcja `if - elif - else`:

```
if warunek:
    operacje
elif warunek:
    operacje
else:
    operacje
```

  - `elif` może się pojawić raz, kilka razy, albo wcale,
  - `else` jest opcjonalnym elementem, może wystąpić co najwyżej raz.

- Zapis alternatywny:

  *operacja*1 `if` *warunek* `else` *operacja*2

  - *operacja*1, jeśli *warunek* jest prawdziwy,
  - w przeciwnym razie *operacja*2.

- Typowe operatory: `<`, `>`, `<=`, `>=`, `==`, `!=`, `is`, `is not`, `and`, `or`, `not`.
  - `is`, `is not` – porównują identyczność (wartośc funkcji `id(`*obiekt*`)`).

Politechnika Wrocławska

Pętle

- Przeglądanie / iterowanie po kolekcji:

```
for element in kolekcja:
    operacje
else:
    operacje
```

- Powtórzenie zależnie od warunku:

```
while warunek:
    operacje
else:
    operacje
```

  - `else` jest opcjonalnym elementem, może wystąpić co najwyżej raz,
  - wykonywany jest po pętli, o ile nie została ona przerwana.

- Wyrażenia sterujące: `break`, `continue`.
- Instrukcja, która nic nie robi: `pass`.

Odwzorowywanie listy

- Znane powszechnie pod angielskim terminem *list comprehension*.
- Mechanizm, który pozwala łatwo przekształcić jedną listę w inną.
- Wariant podstawowy:
  - *lista* = [*operacja* `for` *element* `in` *kolekcja*]
- Wariant rozszerzony – z filtracją:
  - *lista* = [*operacja* `for` *element* `in` *kolekcja* `if` *warunek*]
- Przykład:

```
argumenty = [0, 7, -2, 'a', 3.5, [1, 2, 3], {'a': 2}]
wartości = [(liczba**2)**0.5 for liczba in argumenty
                        if isinstance(liczba, int)]
```

Funkcje (1/2)

- Ogólny sposób definiowania funkcji:

```
def nazwa(argumenty):
    operacje
```

- Argumenty funkcji:
  - może ich nie być w ogóle,
  - wymagane – podajemy same ich nazwy, rozdzielone przecinkiem,
  - opcjonalne – dodatkowo podajemy ich domyślną wartość,
  - `*args` – dowolna liczba argumentów nienazwanych,
    - widoczne jako tablica `args`,
  - `**kwargs` – dowolna liczba argumentów nazwanych
    - widoczne w ciele funkcji jako słownik `kwargs`.
- Przykład:

```
def super_funkcja(wymagany, opcjonalny=1337, *args, **kwargs):
    print(wymagany, opcjonalny, args, kwargs)
```

Funkcje (2/2)

- Zwracane wartości – przez słowo kluczowe `return`.
  - Może ich nie być w ogóle (brak `return`).
    - Fachowo mówimy wtedy o **procedurze**.
  - Może zostać zwrócona pojedyncza wartość dowolnego typu.
  - Może zostać zwrócone kilka różnych wartości.
    - Wartości zwracane są wtedy jako krotka.
- Teoretycznie funkcja może zwracać wartości różnego typu.
  - W praktyce jednak **nie jest to zalecane** podejście.
- Domyślne wartości funkcji **nie powinny** być typu mutowalnego.
- Przykład zwracania wielu wartości:

```
def funkcja():
    return True, 1337, ['a', 'b', 7]

a, b, c = funkcja()
```

Keyword arguments i rozpakowywanie kolekcji

- Nazwane argumenty do funkcji można podać w dowolnej kolejności.
  - Pozwala to uniknąć przedefiniowywania wielu domyślnych wartości.
- Argumenty funkcji można przechowywać w osobnych kolekcjach.
- Wyodrębnienie wartości i przekazanie do funkcji przy użyciu operatorów:
  - `*` – argumenty nienazwane,
  - `**` – argumenty nazwane.
- Przykład:

```
print(1, 2, 3, sep=' -- ', end=' !!\n')
print(1, 2, 3, end=' !!\n', sep=' -- ')

dane = [1, 2, 3]
opcje = {'sep': ' -- ', 'end': ' !!\n'}
print(*dane, **opcje)
```

Politechnika Wrocławska

Klasy (1/2)

- Ogólny sposób definiowania klasy:

```
class Nazwa(KlasaBazowa):

    def __init__(self, argumenty):
        self.zmienna = wartość
        operacje

    def metoda(self, argumenty):
        operacje
```

- Wszystko w języku Python jest obiektem.
  - Domyślną klasą bazową jest `Object`.
  - Dostępne w klasie metody i pola zwraca funkcja `dir(obiekt)`.
- Obiekt (instancję klasy) tworzy się, wywołując jej nazwę w kodzie.
  - Jeśli konstruktor wymaga argumentów, to należy też je podać.

Klasy (2/2)

- Wszystkie pola i metody klasy są zawsze **publiczne**.
  - Według konwencji wyróżnia się jeszcze pola i metody prywatne.
  - Oznacza się je, rozpoczynając nazwę elementu od _ (podkreślenie).
- Słowo kluczowe `self` pozwala odnieść się do elementów instancji klasy.
- Dostęp do pól i metod klasy realizowany jest przez operator . (kropka).
- Przykład definicji klasy i stworzenia jej instancji:

```
class Osoba(Object):
    def __init__(self, imie, nazwisko):
        self.imie = imie
        self.nazwisko = nazwisko

    def inicjaly(self):
        return self.imie[0] + self.nazwisko[0]

pracownik = Osoba('Jan', 'Kowalski')
print(pracownik.inicjaly())
```

Politechnika Wrocławska

Moduły

- Mechanizm, pozwalający rozszerzać język o nowe możliwości.
- Rodzaje modułów (na podstawie pola *moduł*`.__file__`):
  - wbudowane – dostępne bezpośrednio w interpreterze,
  - skompilowane – dostępne w formie bibliotek systemowych (`dll`, `so`),
  - napisane w języku Python – zdecydowana większość.
- Technicznie **każdy plik** ze skryptem języka Python jest modułem.
  - Nazwa takiego modułu to nazwa pliku bez rozszerzenia.
- Załadowanie modułu odbywa się za pomocą wyrażenia `import`.
  - Elementy modułu są dostępne tak samo jak każde elementy obiektu.
  - Słowo kluczowe `as` pozwala zmienić nazwę wczytanego modułu.
  - Możliwe jest załadowanie tylko części modułu:
    - `import` *moduł.element* `# dostępny jest moduł.element`,
    - `from` *moduł* `import` *element* `# dostępny jest element`.

Uruchomienie programu

- Podstawowy sposób to przekazanie interpreterowi ścieżki do skryptu:
  - `python3 skrypt.py`
- Skrypt można uruchomić także bezpośrednio jako moduł,
  - `python3 -m skrypt`,
- W powłoce można wykorzystać także mechanizm `shebang`:
  - zalecany: *`#!/usr/bin/env python3`*.

Uwaga!

Załadowanie modułu (`import`) w gruncie rzeczy wykonuje jego kod:

- zostaną uruchomione wszystkie rzeczy inne, niż definicje funkcji, klas, itp.,
- dlatego wywołania zabezpieczamy: `if __name__ == '__main__':`,
- wtedy będą one uruchomione tylko przy bezpośrednim uruchomieniu.

Politechnika Wrocławska

Python Style Guide

- Zbiór dobrych praktyk i zasad, ułatwiających czytanie i pisanie kodu.
- Przykładowe reguły:
  - wcięcia tylko i wyłącznie jako **4 spacje**,
  - w jednej linijce skryptu mniej niż **80 znaków**,
  - obowiązkowe spacje wokół operatorów matematycznych,
  - brak spacji po wewnętrznych stronach nawiasów,
  - brak nieużywanych modułów w skryptach,
  - wczytywanie tylko jednego modułu na raz w linijce,
  - kodowanie pliku ze skryptem używając standardu **UTF-8**.
- Bardzo przydatne przy kontrolowaniu stylu kodu jest narzędzie `flake8`:
  - `pip3 install flake8`,
- Pełny dokument:
  - https://www.python.org/dev/peps/pep-0008/.

Politechnika Wrocławska

Odpluskiwanie węża

- Bardzo przydatnym narzędziem jest wbudowany `pdb`,
  - **p**ython **d**e**b**ugger.
- Pozwala uruchomić interaktywną sesję w dowolnym miejscu skryptu.
- Umożliwia analizę zawartości zmiennych, wyrażeń, wywołań, itd.
- Przykładowe użycie:
  - `import pdb; pdb.set_trace()`,
  - tę linijkę zawsze warto mieć przy sobie!
- W nowych wersjach Pythona wystarczy wywołać `breakpoint()`.
- Jako zewnętrzny moduł dostępny jest także `pdb++`,
  - oferuje on kilka dodatkowych udogodnień przy pracy.

Politechnika Wrocławska

Gdzie szukać informacji?

- Fenomenalna oficjalna dokumentacja:
  - https://docs.python.org/.
- Kod źródłowy interpretera cpython:
  - https://github.com/python/cpython.
- Informacje o alternatywnych interpreterach:
  - https://www.python.org/download/alternatives/.
- Funkcja `help()` – wiele modułów oraz funkcji wbudowanych.
- Standardowa wyszukiwarka internetowa również się przydaje!

Politechnika Wrocławska

Przygotowanie do zajęć

Skonfigurowanie środowiska

Instalacja interpretera

W systemie **macOS**:

- Potrzebne rzeczy powinny być domyślnie zainstalowane.
- Jeśli w systemie brakuje interpretera języka Python w wersji 3:
  - `brew install python3`
  - Narzędzie **brew**: https://brew.sh.

W systemie **Linux**:

- Dystrybucja **Arch Linux** i podobne:
  - `sudo pacman -Syu python python-pip`
- Dystrybucja **Ubuntu Linux** i podobne:
  - `sudo apt update && apt install python3 python3-pip`

W systemie **Windows**:

- Pobranie i zainstalowanie interpretera języka Python.
  - Strona: https://www.python.org/downloads/windows/.
  - W instalatorze zaznaczyć opcję *Add Python to PATH*.

Politechnika Wrocławska

Rozpoczęcie pracy (1/2)

- Pobrać i rozpakować archiwum z plikami do zajęć ze strony:
  https://datko.pl/IOb/
- Wejść do pobranego katalogu z plikami źródłowymi.
- Stworzyć wirtualne środowisko:
  `python3 -m venv venv/`
- Aktywować wirtualne środowisko:
  - w systemie Linux:
    `. venv/bin/activate`
  - w systemie Windows:
    `venv/Scripts/activate`
- Zaktualizować narzędzia budujące wewnątrz writualnego środowiska:
  `pip3 install --upgrade pip setuptools wheel`
- Zainstalować potrzebne zależności:
  `pip3 install -r requirements.txt`

Rozpoczęcie pracy (2/2)

- Uruchomić serwer aplikacji **Jupyter**:
  `jupyter notebook`
- Zaczekać na załadowanie się przeglądarki internetowej.
- W aplikacji **Jupyter** załadować plik `rozwiązania.ipynb`.
- Zrealizować zadania w przygotowanych komórkach aplikacji **Jupyter**.
- Jeśli przygotowane rozwiązanie wymaga dodatkowych bibliotek, to należy koniecznie dopisać je w pliku `requirements.txt`.
- Wszelkie pliki pomocnicze powinny być w katalogu projektu.
  - Należy odwoływać się do nich przy pomocy ścieżek względnych.
- Dla pewności, że wszystko działa, przebudować wszystkie komórki.
  - W aplikacji **Jupyter** z menu `CELL` wybieramy pozycję `run all`.
- Na sam koniec pracy proszę pamiętać, aby w archiwum do zamieszczenia w systemie ePortal **nie dołączać** katalogu `venv/` – czyli usunąć go!

Politechnika Wrocławska

Koniec wprowadzenia.

Zadania do wykonania...

Zadania do wykonania (1)

Na ocenę **3.0** należy przygotować swoje środowisko pracy.

Wskazówki:

- zainstalować wszystkie potrzebne narzędzia,
- w wybranym miejscu systemu plików założyć katalog projektu,
- przygotować w nim prosty program, wyświetlający dowolny napis,
- uruchomić przygotowany program w aplikacji **Jupyter**.

Zadania do wykonania (2)

Na ocenę **3.5** należy wykorzystać w programie zewnętrzny moduł.

Wskazówki:

- wykorzystać mechanizm wirtualnych środowisk,
- określić zależności w pliku `requirements.txt`,
- można użyć dowolnego modułu, proponowanym jest `cowsay`.

```
# Przykładowy kod
import cowsay
cowsay.stegosaurus("Howdy?")
```

Zadania do wykonania (3)

Na ocenę **4.0** należy użyć dodatkowych plików jako wejście i wyjście skryptu.

Wskazówki:

- zdefiniować funkcję, obliczającą wartość jakiegoś wyrażenia,
- warto wykorzystać moduł `math`, np. aby użyć funkcji `sin()`,
- stworzyć plik, który będzie zawierał argumenty zdefiniowanej funkcji,
- program powinien w pliku wynikowym zapisać w każdej linii kolejny argument oraz obliczoną dla tego argumentu wartość funkcji.

Zadania do wykonania (4)

Na ocenę **4.5** należy stworzyć dwie klasy: `Product` i `Cart`.

Wskazówki:

- klasa produktu powinna zawierać trzy pola (ustawiane w konstruktorze):
  - `name`, `price` i `quantity`,
- klasa koszyka niech zawiera pewną domyślnie ustaloną listę produktów,
- dodać w klasie koszyka metody `add()` i `remove()`,
  - argumentem metod powinna być żądana nazwa produktu,
  - w wyniku działania powinna odpowiednio zmienić się wartość `quantity` dla produktu, którego `name` odpowiada żądanej nazwie,
  - jeśli nie ma produktu o danej nazwie to nic się nie dzieje,
- dodać metodę `total_price()` w klasie koszyka, która zwróci sumaryczną wartość wszystkich produktów (ich cena razy liczba),
- program powinien wyświetlić przykładowe działanie opisanych metod.

Zadania do wykonania (5)

Na ocenę **5.0** należy rozszerzyć program z poprzedniego slajdu.

Wskazówki:

- zaimplementować metodę `__str__` dla obydwu klas,
  - dla produktu niech zwraca ona elegancko jego nazwę i cenę,
  - dla koszyka powinna zwrócić informacje o produktach i ich liczności,
- zaimplementować metodę `__len__` dla klasy koszyka,
  - powinna ona zwracać po prostu sumę `quantity` produktów,
- umożliwić wygodne iterowanie w pętli po produktach w koszyku,
  - zaimplementować metody `__iter__` i `__next__`,
  - tak, aby ostatecznie powinien zadziałał zapis:
    `for produkt in koszyk`,
- pokazać działanie przygotowanych metod na przykładzie.