

PyStart #32 Args i kwargs, więcej o argumentach

Czym jest args i kwargs?

- Czasem potrzebujemy odebrać wszystko co zostało przekazane do funkcji. Osobno możemy odebrać argumenty nazwane i osobno argumenty pozycyjne.
- Nazwy **args** i **kwargs** są opcjonalne. Ważne jest to ile gwiazdek jest przed argumentem.



PyStart #32 Args i kwargs, więcej o argumentach

Args

```
1 def average(*args):  
2     print(args)  
3     return sum(args) / len(args)  
4  
5  
6 print(average(10, 20, 30))
```

→ Wynik

```
(10, 20, 30)  
20.0
```



PyStart #32 Args i kwargs, więcej o argumentach

Co to za args z gwiazdką?

```
1 def average(*args):
```

- `*args` odbierze wszystkie **pozostałe** argumenty nienazwane,
- wszystkie argumenty zostaną odebrane w postaci tupli

```
6 def print_args(a, b, *others):  
7     print('a', a)  
8     print('b', b)  
9     print(others)
```



PyStart #32 Args i kwargs, więcej o argumentach

Co to za kwargs z dwiema gwiazdkami?

- `**kwargs` odbierze wszystkie **pozostałe** argumenty nazwane,
- wszystkie argumenty zostaną odebrane w postaci słownika

```
1 def save(filename, **kwargs):  
2     print(filename)  
3     print(kwargs)  
4  
5  
6 save('students.txt', first_name='John', last_name='Smith')  
7
```



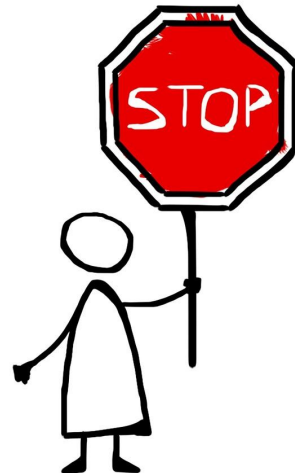
PyStart #32 Args i kwargs, więcej o argumentach

Zadania dla nabrania wprawy

1. Napisz funkcję, która przyjmuje dowolną ilość argumentów: liczb oraz wartość graniczna. Funkcja powinna zwrócić sumę liczb, które są większe niż podana wartość graniczna.

Funkcja powinna przyjmować argumenty pozycyjne `*args` oraz `limit`, który będzie wartością graniczną.

2. Napisz funkcję, która przyjmuje listę słów i zwraca słowo, które występuje najczęściej. Funkcja powinna przyjmować argumenty nazwane `*args` oraz `ignore_case` z wartością domyślną `True`, który będzie oznaczał, czy ignorować wielkość liter podczas zliczania wystąpień.



PyStart #33 Podzielmy projekt na części

Trochę terminologii

- Czym jest funkcja?
- Czym jest moduł?
- Czym jest pakiet?



PyStart #33 Podzielmy projekt na części

Trochę terminologii

→ Czym jest funkcja?

```
1 def count_letters(text, start='(', end=')'):  
2     pass  
3  
4  
5     count_letters('(ała) ma (kota)')  
6     # zwróci 3 + 4  
7  
8     count_letters('<> kod <103>', '<', '>')  
9     # zwróci 3  
10  
11     count_letters('abrakadabra')  
12     #zwróci 0
```



PyStart #33 Podzielmy projekt na części

Trochę terminologii

→ Czym jest moduł?

- ◆ To po prostu plik z rozszerzeniem `.py`.
- ◆ Używamy ich do importowania funkcji.
- ◆ Nazywamy je tak samo jak funkcje (z podkreśleniem).
- ◆ Główny moduł to `main.py`.



PyStart #33 Podzielmy projekt na części

Trochę terminologii

→ Czym jest pakiet?

- ◆ Upraszczając to folder, który posiada plik `__init__.py`
- ◆ Pakietami tworzymy hierarchię
- ◆ Pozwala na grupowanie modułów



PyStart #33 Podział projektu, reużywalność

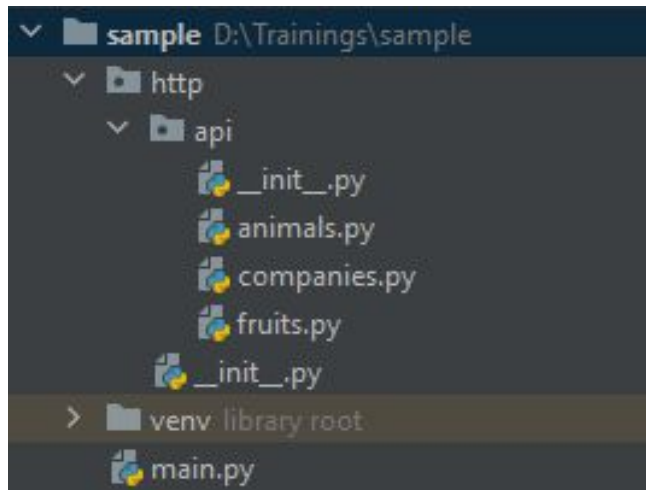
Dlaczego dzielimy projekt?

- Separacja testów
- Grupowanie funkcji według zastosowania
- Przenoszenie pakietów między projektami
- Publikowanie pakietów open-source
- Łatwiejsza nawigacja po projekcie.. Ordnung muss sein!
- Praca zespołowa / dzielenie odpowiedzialności



PyStart #33 Podział projektu, reużywalność

Przykład



http to jest pakiet
api to jest pakiet

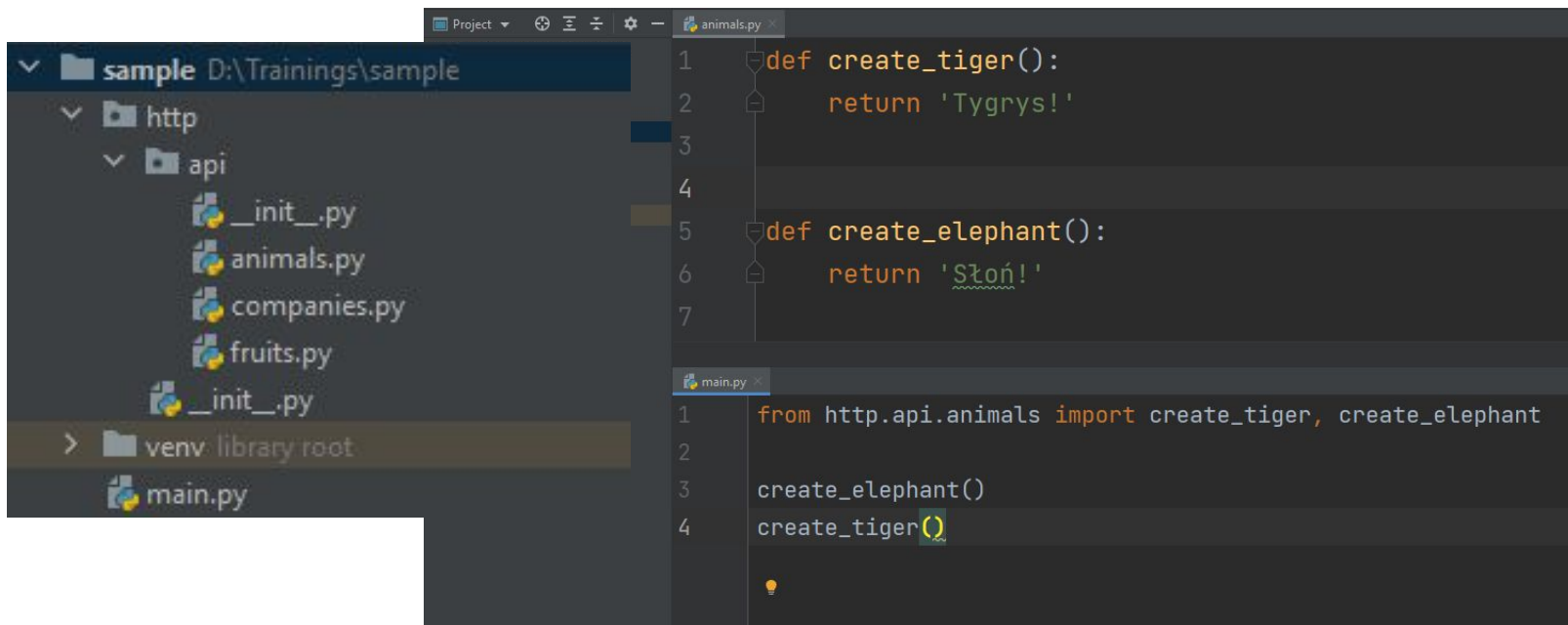
animals to moduł
companies to moduł
fruits to moduł



PyStart #33 Podział projektu, reużywalność

Jak to zaimportować?

Zawsze względem folderu w którym jest środowisko



The screenshot shows a code editor with a project structure on the left and two Python files open on the right. The project structure is as follows:

- sample (D:\Trainings\sample)
 - http
 - api
 - __init__.py
 - animals.py
 - companies.py
 - fruits.py
 - __init__.py
 - venv library root
 - main.py

The `animals.py` file contains the following code:

```
1 def create_tiger():
2     return 'Tygrys!'
3
4
5 def create_elephant():
6     return 'Słoń!'
7
```

The `main.py` file contains the following code:

```
1 from http.api.animals import create_tiger, create_elephant
2
3 create_elephant()
4 create_tiger()
```

PyStart #33 Podział projektu, reużywalność

Jak to zaimportować?

Zawsze względem folderu w którym jest środowisko

```
animals.py
1 def create_tiger():
2     return 'Tygrys!'
3
4
5 def create_elephant():
6     return 'Słoń!'
7

main.py
1 from http.api.animals import create_tiger, create_elephant
2
3 create_elephant()
4 create_tiger()
```

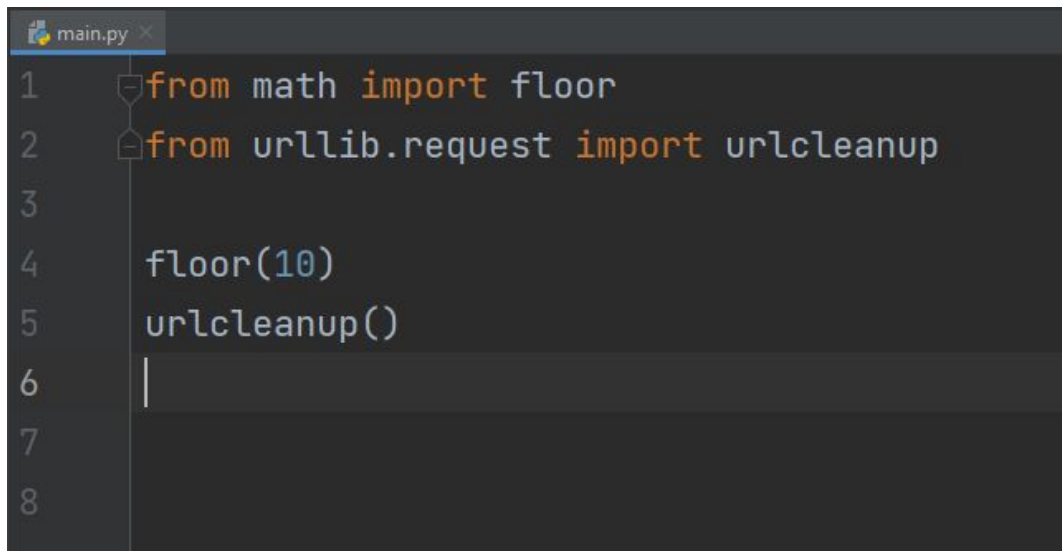
```
animals.py
1 def create_tiger():
2     return 'Tygrys!'
3
4
5 def create_elephant():
6     return 'Słoń!'
7

main.py
1 from http.api import animals
2
3 animals.create_elephant()
4 animals.create_tiger()
```

PyStart #33 Podział projektu, reużywalność

Jak to zaimportować?

Tak samo z innymi funkcjami z venv / biblioteki standardowej

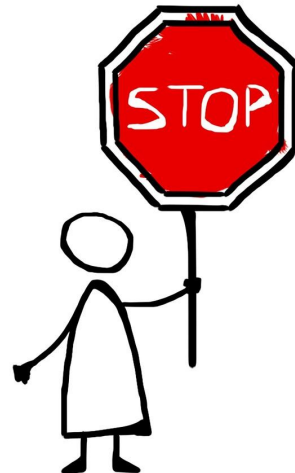


```
main.py x
1 from math import floor
2 from urllib.request import urllcleanup
3
4 floor(10)
5 urllcleanup()
6
7
8
```

PyStart #33 Podział projektu, reużywalność

Zadania dla nabrania wpraw

1. Utwórz pakiet główny o nazwie `pystart`
2. W pakiecie utwórz dwa pakiety: `utils` oraz `calculations`
3. W pakiecie `utils` stwórz moduł `lists` zawierający funkcje
 - a. `add_lists` - dodającą wartości z dwóch list znajdujące się na tych samych indeksach
4. W pakiecie `calculations` stwórz moduł `geometry`, a w nim:
 - a. funkcję do obliczania pola oraz obwodu okręgu i trójkąta równobocznego.
 - b. funkcję do obliczania długości i wyznaczania środka odcinka o zadanych współrzędnych (początek i koniec)



PyStart #34 Twój główny plik main.py

Skąd ta nazwa?

- Nie ma uzasadnienia dlaczego główny plik to main.py po prostu tak jest :-)
- Od tego pliku wszystko się zaczyna i do tego pliku importowane są potrzebne funkcje.
- Jest jeszcze jeden stały fragment, który zawsze zawiera plik **main.py**

```
if __name__ == '__main__':  
    # tu kod Twojego projektu  
    # np. wywołanie funkcji  
    main()
```

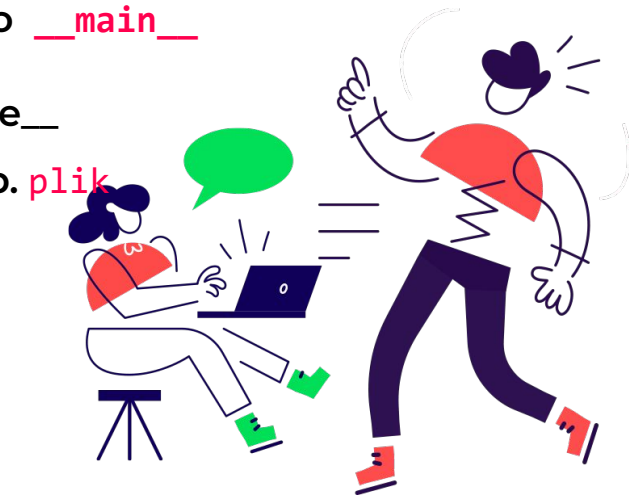


PyStart #34 Twój główny plik main.py

Co oznacza `__name__=='__main__'`

- `__name__` to specjalna zmienna zawierająca nazwę modułu w którym jest używana
jeśli jest to główny skrypt to właśnie nazwa to `__main__`
- Jeśli uruchamiasz moduł `python plik.py` to `__name__` to `__main__`
- Jeśli moduł jest importowany do innego pliku to `__name__`
to nazwa pliku w którym jest kod, ale to nie używane np. `plik`
podobnie będzie przy użyciu `pytest`'a

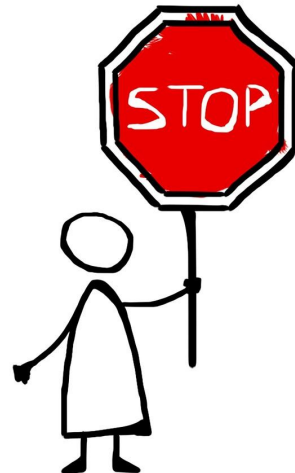
```
if __name__ == '__main__':  
    # tu kod Twojego projektu  
    # np. wywołanie funkcji  
    main()
```



PyStart #34 Podział projektu, reużywalność

Zadania dla nabrania wpraw

1. Utwórz pakiet główny o nazwie `pystart`
2. W pakiecie utwórz dwa pakiety: `utils` oraz `calculations`
3. W pakiecie `utils` stwórz moduł `lists` zawierający funkcje
 - a. `add_lists` - dodającą wartości z dwóch list znajdujące się na tych samych indeksach
4. W pakiecie `calculations` stwórz moduł `geometry`, a w nim:
 - a. funkcję do obliczania pola oraz obwodu okręgu i trójkąta równobocznego.
 - b. funkcję do obliczania długości i wyznaczania środka odcinka o zadanych współrzędnych (początek i koniec)



PyStart #35 Tworzymy własną bibliotekę

Jak to zaimportować?



main.py

 pakiet / some_module.py (w środku jest funkcja some_function)
venv

Opcja 1:

```
from pakiet import some_module  
some_module.some_function()
```

PyStart #35 Tworzymy własną bibliotekę

Jak to zaimportować?



main.py

 pakiet / some_module.py (w środku jest funkcja some_function)
venv

Opcja 1:

```
from pakiet import some_module  
some_module.some_function()
```

Opcja 2:

```
from pakiet.some_module import some_function  
some_function()
```

PyStart #35 Tworzymy własną bibliotekę

Jak to zaimportować?



main.py

 pakiet / some_module.py (w środku jest funkcja some_function)
venv

Opcja 1:

```
from pakiet import some_module  
some_module.some_function()
```

Opcja 2:

```
from pakiet.some_module import some_function  
some_function()
```

Opcja 3 (Tak nie robimy)

```
from pakiet.some_module import *  
some_function()
```

PyStart #36 Tworzymy własną bibliotekę

Jak to zaimportować?



- Założmy, że biblioteka `pystart` przyda Ci się w innym projekcie.
- Stworzymy z biblioteki paczkę, którą można będzie przenieść.
- Utworzymy plik `setup.py`

PyStart #36 Tworzymy własną bibliotekę

Jak to zaimportować?



- Po pierwsze należy utworzyć plik `setup.py` plik powinien być umiejscowiony wewnątrz paczki `pystart`

```
from setuptools import setup, find_packages

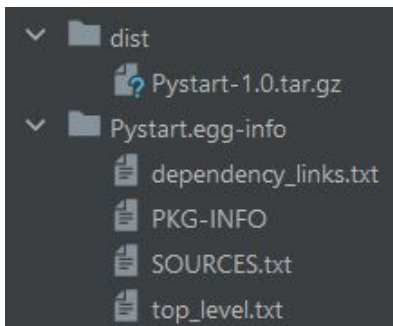
setup(name='pystart',
      version='0.0.1',
      description="this is just description",
      long_description="",
      author="Kacper Sieradziński",
      author_email="ksieradzinski@gmail.com",
      url='https://pystart.pl',
      packages=find_packages(exclude=['tests']))
```

PyStart #36 Tworzymy własną bibliotekę

Jak to zaimportować?



→ Następnie należy wykonać polecenie `python setup.py sdist`



Ten plik będzie nam potrzebny

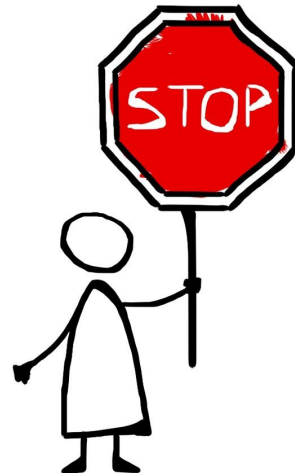
PyStart #35 Tworzymy własną bibliotekę

Zadania dla nabrania wprawy

→ Wykonaj przedstawione polecenia i stwórz paczkę z biblioteki pystart.

Skopiuj plik z rozszerzeniem **tar.gz** w dowolnej miejsce,

bo będzie nam już niedługo potrzebny.



PyStart #36 Zainstalujmy naszą paczkę

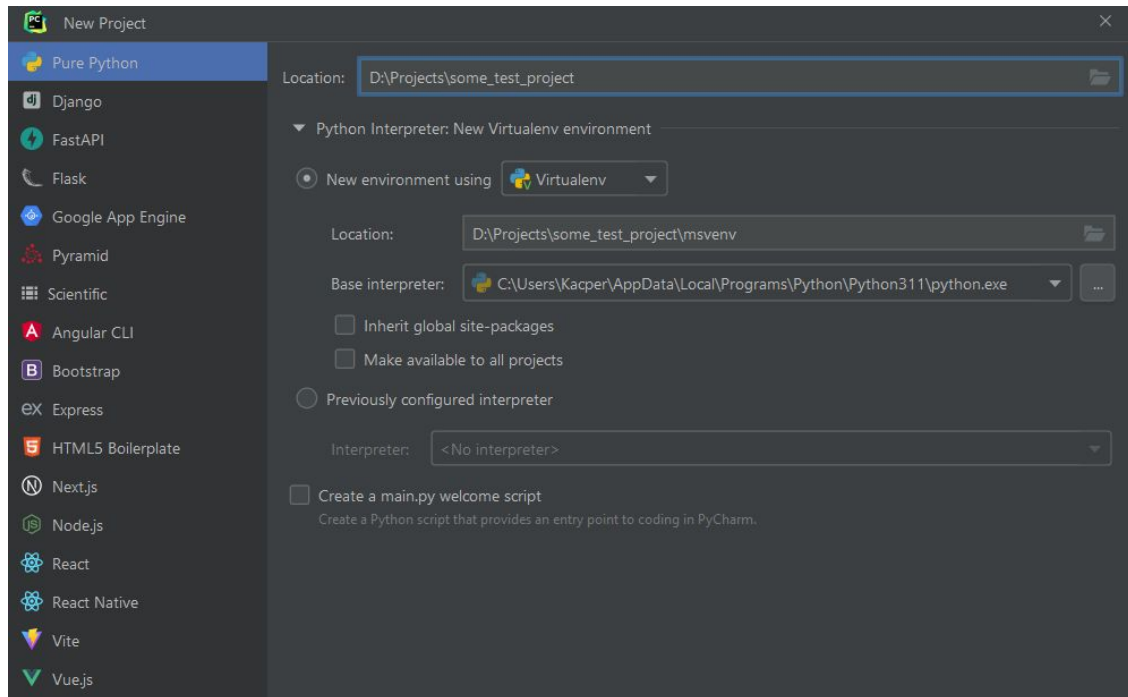
W nowym projekcie



- Potrzebujemy utworzyć nowy projekt wraz ze środowiskiem wirtualnym.
- Przeprowadzimy instalację paczki `tar.gz`

PyStart #36 Zainstalujmy naszą paczkę

Tworzymy nowy projekt

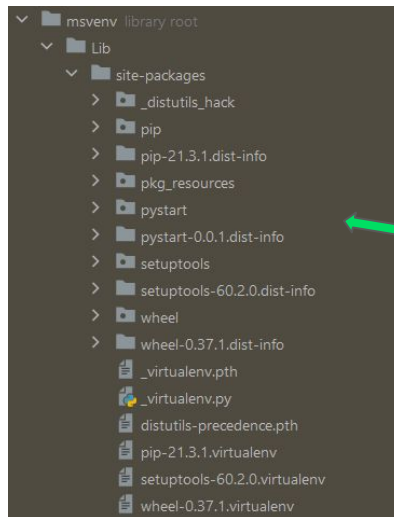
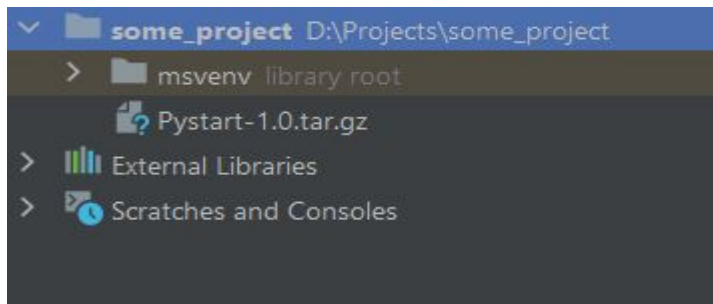


PyStart #36 Zainstalujemy naszą paczkę

Przenosimy paczkę do projektu

→ Może być w innym miejscu ale tak będzie wygodniej

```
pip install paczka.tar.gz
```



PyStart #36 Zainstalujmy naszą paczkę

I jak z tego skorzystać?



→ Normalnie :-)

```
from pystart import cokolwiek
```

Dokładnie tak samo jak sobie wcześniej pokazaliśmy.

PyStart #36 Tworzymy własną bibliotekę

Zadania dla nabrania wprawy

- Wykonaj przedstawione polecenia, a następnie w pliku `main.py` napisz kod, który odbierze od użytkownika dwa stringi w postaci cyfr rozdzielonych przecinkami i zwróci nową listę zawierającą sumy.
- Wykorzystaj funkcje zaimplementowane w bibliotece `pystart` oraz dopisz własne funkcje jeśli czegoś Ci zabraknie.



PyStart #37 Zależności

Czym są zależności?



- Wszystko co instalujemy za pomocą **pip'a**
- Biblioteki pochodzące “z zewnątrz” potrzebne dla uruchomienia lub rozwoju projektu.
- Nie umieszczamy ich w repozytorium!
- Skąd wiemy co doinstalować?

Plik **requirements.txt** lub biblioteka **Poetry**

PyStart #37 Zależności

Jak stworzyć listę zależności?

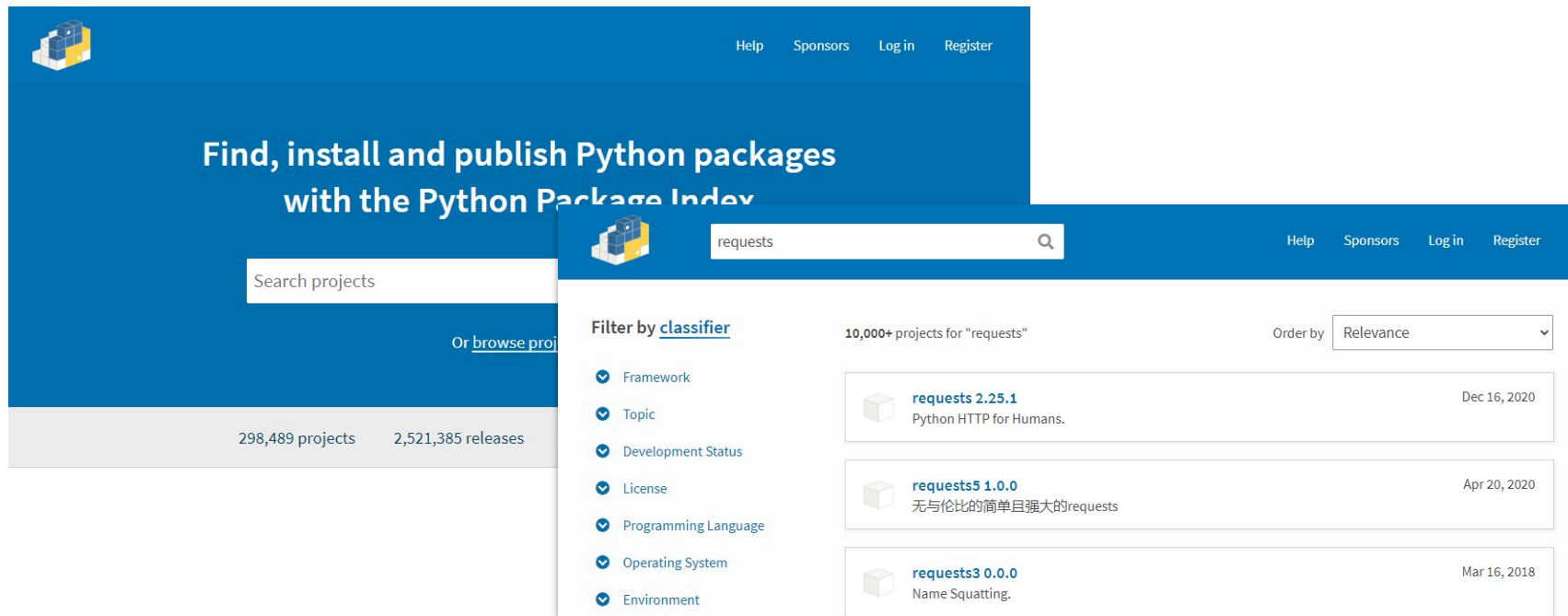


- Wszystko wylistuje nam `pip freeze`
- Potrzebne linie dodajemy do `requirements.txt` (nazwa jest umowna)
- Sprawdzimy to sobie w następnym module gdy będziemy instalowali zależności.

PyStart #38 Pip i gotowe biblioteki?

Skąd brać pakiety?

→ Pypi.org



The image shows a screenshot of the PyPI.org website. The main header is blue with the PyPI logo on the left and links for Help, Sponsors, Log in, and Register on the right. The main text reads "Find, install and publish Python packages with the Python Package Index". Below this is a search bar with the text "Search projects" and a link "Or browse projects". At the bottom, it shows "298,489 projects" and "2,521,385 releases".

A search for "requests" is shown, resulting in 10,000+ projects. The results are filtered by classifier and ordered by Relevance. The top results are:

- requests 2.25.1** (Dec 16, 2020): Python HTTP for Humans.
- requests5 1.0.0** (Apr 20, 2020): 无与伦比的简单且强大的requests
- requests3 0.0.0** (Mar 16, 2018): Name Squatting.

The left sidebar shows filters by classifier:

- Framework
- Topic
- Development Status
- License
- Programming Language
- Operating System
- Environment

PyStart #38 Zależności

Jak wybrać właściwą bibliotekę?

→ Sprawdzamy czy jest wciąż rozwijana + Ilość gwiazdek na githubie + jak użyć



requests 2.28.2

pip install requests

Released: Jan 12, 2023

Python HTTP for Humans.

Navigation

Project description

Release history

Download files

Project links

Homepage

Documentation

Source

Statistics

Project description

Requests

Requests is a simple, yet elegant, HTTP library.

```
>>> import requests
>>> r = requests.get('https://httpbin.org/basic-auth/user/pass', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"authenticated": true, ...}'
>>> r.json()
{'authenticated': True, ...}
```

PyStart #38 Zależności

Jak wybrać właściwą bibliotekę?

→ Instalujemy bibliotekę pytest, rozpoczynamy temat testów

pytest 7.2.1

```
pip install pytest
```



[Latest version](#)

Released: Jan 14, 2023



PyStart #39 TDD - test driven development

Przypomnijmy sobie coś

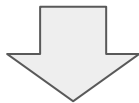
Funkcja zawsze zwraca tą samą wartość jeśli
jest wywoływana z tymi samymi argumentami.



PyStart #39 TDD - test driven development

Przypomnijmy sobie coś

Funkcja zawsze zwraca tą samą wartość jeśli jest wywoływana z tymi samymi argumentami.



Potrafimy przewidzieć kiedy funkcja działa dobrze oraz kiedy działa źle.

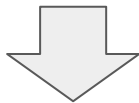


PyStart #39 TDD - test driven development

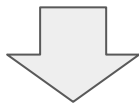
Przypomnijmy sobie coś



Funkcja zawsze zwraca tą samą wartość jeśli jest wywoływana z tymi samymi argumentami.



Potrafimy przewidzieć kiedy funkcja działa dobrze oraz kiedy działa źle.



Za każdym razem sprawdzaliśmy nasz program, używając różnych argumentów



PyStart #39 TDD - test driven development

Przypomnijmy sobie coś



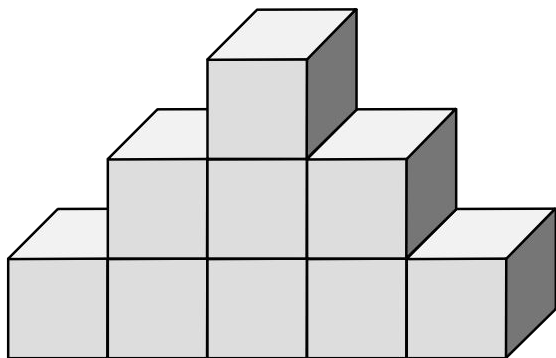
To może da się to sprawdzać automatycznie !?



PyStart #39 TDD - test driven development

Co testujemy?

W przypadku testów jednostkowych naszymi jednostkami są funkcje.



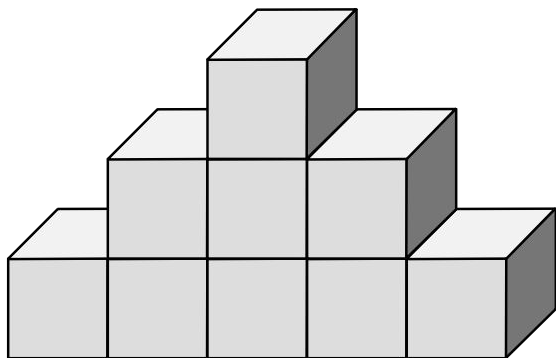
Testy jednostkowe



PyStart #39 TDD - test driven development

Co testujemy?

W przypadku testów jednostkowych naszymi jednostkami są funkcje.



Testy integracyjne

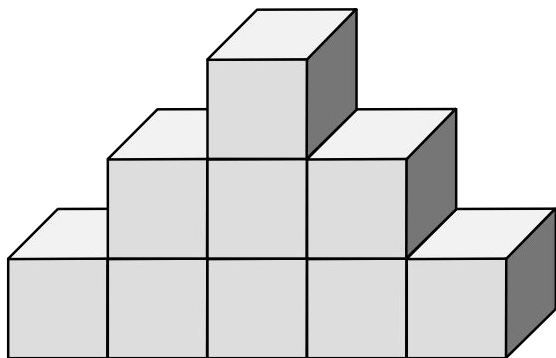
Testy jednostkowe



PyStart #39 TDD - test driven development

Co testujemy?

W przypadku testów jednostkowych naszymi jednostkami są funkcje.



Testy end-end

Testy integracyjne

Testy jednostkowe



PyStart #39 TDD - test driven development

Co warto testować?

- Logikę naszej aplikacji.
- Kod, który napisaliśmy sami.
- Kod, który modyfikujemy aby upewnić się, że nic nie zepsuliśmy.



PyStart #39 TDD - test driven development

Czy wszyscy piszą testy?



PyStart #39 TDD - test driven development

Czy wszyscy powinni je pisać?



PyStart #39 TDD - test driven development

Czy warto?



się,



PyStart #39 TDD - test driven development

Dlaczego testujemy?

- Mamy pewność, że nasz program działa.
- Upewniamy się, że nic nie zepsuliśmy.
- Wprowadzamy zmiany dużo szybciej,
nie musimy wszystkiego sprawdzać za każdym razem.



PyStart #39 TDD - test driven development

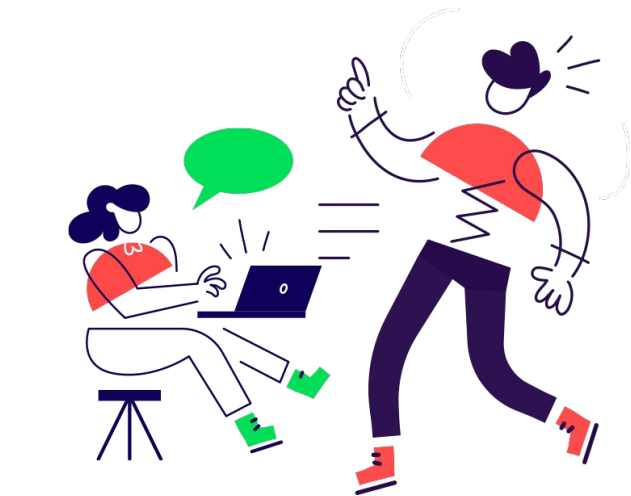
Jak testować?

Instalujemy bibliotekę

```
pip install pytest
```

Uruchamiamy testy

```
python -m pytest program.py
```



PyStart #39 TDD - test driven development

Nic się nie dzieje... brakuje testów

- Test to zwykła funkcja zaczynająca się od **test_**
- Testy zawierają kod napisany w pythonie :)
- Dobry test składa się z trzech części:
 - ◆ Given
 - ◆ When
 - ◆ Then



PyStart #39 TDD - test driven development

Nic się nie dzieje... brakuje testów

- Test to zwykła funkcja zaczynająca się od **test_**
- Testy zawierają kod napisany w pythonie :)
- Dobry test składa się z trzech części:
 - ◆ **Given** na dworze jest zimno, ja nie mam czapki
 - ◆ **When**
 - ◆ **Then**



PyStart #39 TDD - test driven development

Nic się nie dzieje... brakuje testów

- Test to zwykła funkcja zaczynająca się od **test_**
- Testy zawierają kod napisany w pythonie :)
- Dobry test składa się z trzech części:
 - ◆ **Given** na dworze jest zimno, ja nie mam czapki
 - ◆ **When** wychodzę z domu na dwie godziny
 - ◆ **Then**



PyStart #39 TDD - test driven development

Nic się nie dzieje... brakuje testów

- Test to zwykła funkcja zaczynająca się od **test_**
- Testy zawierają kod napisany w pythonie :)
- Dobry test składa się z trzech części:
 - ◆ **Given** na dworze jest zimno, ja nie mam czapki
 - ◆ **When** wychodzę z domu na dwie godziny
 - ◆ **Then** następnego dnia boli mnie głowa



PyStart #37 TDD - test driven development

Jak wygląda przykładowy test?

```
1 def add_numbers(a: int, b: int) -> int:
2     return a + b
3
4
5 def test_add_numbers():
6     a = 2
7     b = 3
8
9     value = add_numbers(a, b)
10
11     assert value == 5
12
13
```



PyStart #37 TDD - test driven development

Gdy wszystko jest ok

→ Tak wygląda szczęście ;-)

```
(venv) D:\Trainings\Pystart>python -m pytest trening.py
===== test session starts =====
platform win32 -- Python 3.8.5, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: D:\Trainings\Pystart
collected 1 item

trening.py . [100%]

===== 1 passed in 0.01s =====
(venv) D:\Trainings\Pystart>
```



PyStart #37 TDD - test driven development

Gdy wszystko jest ok



→ Tak wygląda błąd ;-)

```
===== FAILURES =====
----- test_add_numbers -----

def test_add_numbers():
    a = 2
    b = 3

    value = add_numbers(a, b)

>     assert value == 8
E     assert 5 == 8

trening.py:11: AssertionError
===== short test summary info =====
FAILED trening.py::test_add_numbers - assert 5 == 8
===== 1 failed in 0.06s =====
```

PyStart #37 TDD - test driven development

Zróbmy jeszcze raz zaczynając od testów...

- Przygotuj funkcję, która zliczy ilość znaków w tekście zawierających się wewnątrz nawiasów okrągłych. Nawiasy mogą występować w tekście wielokrotnie, nigdy nie będą się w sobie zawierać.

```
1 def count_letters(text, start='(', end=')'):  
2     pass  
3  
4  
5     count_letters('(ala) ma (kota)')  
6     # zwróci 3 + 4  
7  
8     count_letters('<> kod <103>', '<', '>')  
9     # zwróci 3  
10  
11     count_letters('abrakadabra')  
12     #zwróci 0
```

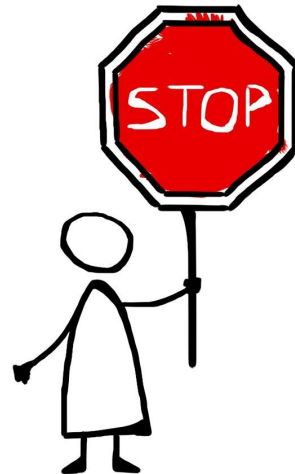


PyStart #37 TDD - test driven development

Zadania dla nabrania wprawy

Napisz testy, a następnie:

- Przygotuj funkcję która zwróci największy wspólny dzielnik dwóch liczb, jeśli takiego nie ma funkcja powinna zwrócić None.
- Przygotuj funkcję, która zwróci listę wszystkich liczb z zakresu od n do m, które będą podzielne przez liczbę z.
- Napisz funkcję, która wczyta tekst i wypisuje wszystkie wyrazy, które zaczynają się na spółgłoskę i kończą na samogłoskę.



SZÓSTA PRACA DOMOWA

UWAGA! UWAGA!

→ Pamiętaj o testach!

→ Przygotuj funkcję, która będzie odbierała argumenty w postaci:

```
group_them(wall="red", tomato="red", light="yellow", lemon="yellow", grass="green")
```

W odpowiedzi funkcja powinna wyświetlić (w osobnych wierszach)

Wall is red \n Tomato is red \n Light is yellow itd.

→ Przygotuj pakiet **finances** posiadający moduły **budget** oraz **ecommerce**.

finances posiada funkcję **calculate_expences(expences:list)** sumującą listę wydatków oraz **calculate_monthly_budget(total_income, total_expences)**

ecommerce posiadać funkcję **calculate_brutto** oraz **calculate_netto** z wartością vat domyślnie równą 0.23, a także funkcje która **calculate_rabat**, która otrzyma listę rabatów np. 0.1, 0.2, 0.3 i będzie potrafiła je odjąć jako wartości procentowe. **PAMIĘTAJ O TESTACH**

Stwórz paczkę instalacyjną.



SZÓSTA PRACA DOMOWA

UWAGA! UWAGA!



Pamiętaj o testach!

- Przygotuj funkcję, która zlicza ilość wystąpień jednego słowa w tekście np.
`count_word('Python i Python', 'Python')` da wynik 2
- Wykorzystaj kod napisany poprzednio do przygotowania funkcji weryfikującej numer PESEL. Funkcja powinna zwracać True lub False w zależności od tego czy PESEL jest prawidłowy czy nie.
- Przygotuj funkcję, która będzie liczyła BMI na podstawie argumentów: wzrost oraz waga. $BMI = \text{masa(kg)} / \text{wzrost (m)}^2$

Druga funkcja powinna na podstawie wartości liczbowej BMI opisywać wynik za pomocą tekstu:

mniej niż 16 – wygłodzenie	16-16,99 – wychudzenie
17-18,49 – niedowaga	18,5-24,99 – wartość prawidłowa
25-25,99 – nadwaga	30-34,99 – otyłość I stopnia
35-39,99 – otyłość II stopnia	powyżej 40 – skrajna otyłość

