

# PYTHON HAS POWER

---

STX Next 2014

# AGENDA

---

9:30 - 11:00

Część 1 - Podstawy

11:15 - 12:45

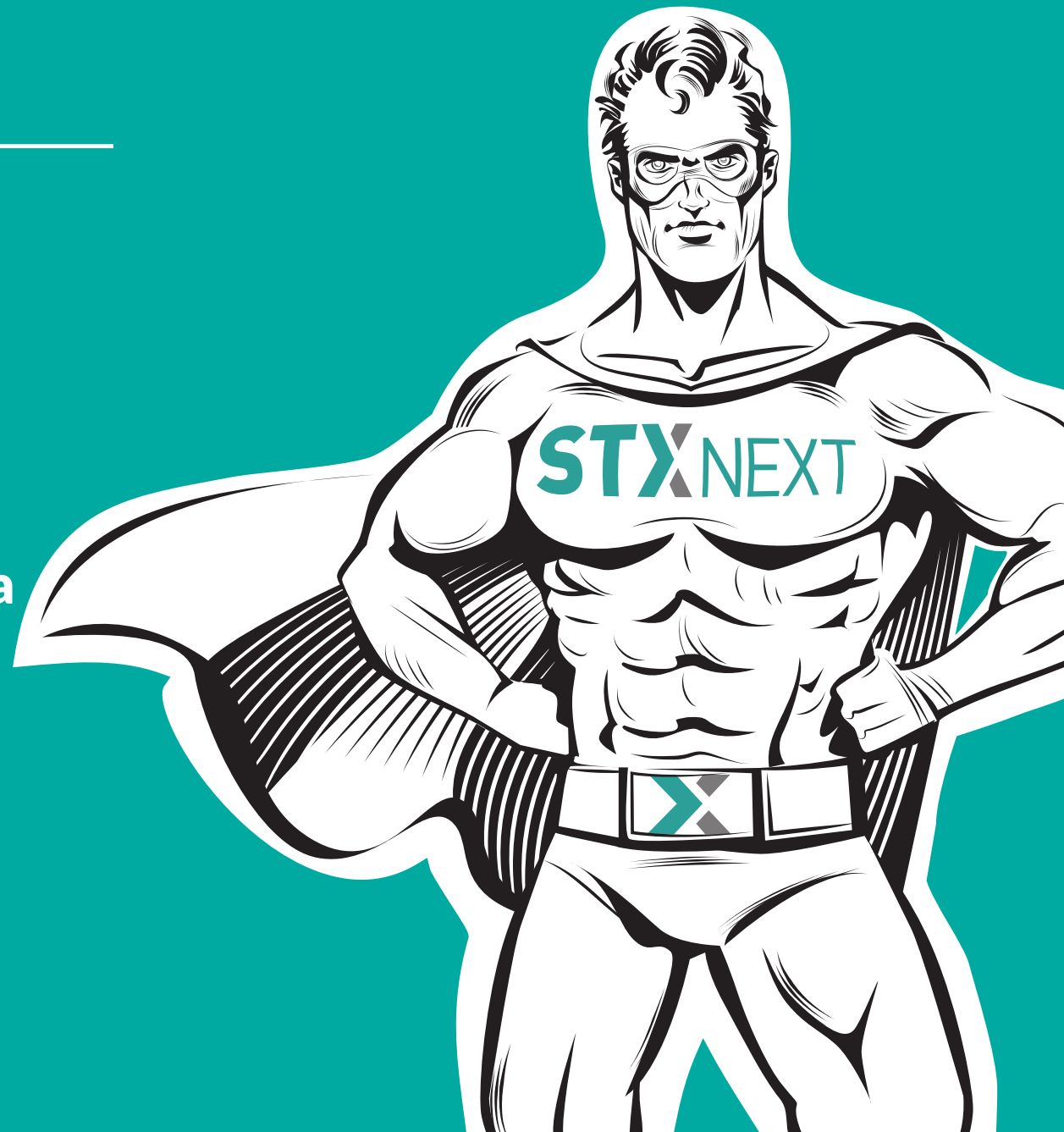
Część 2 - Advanced Python!

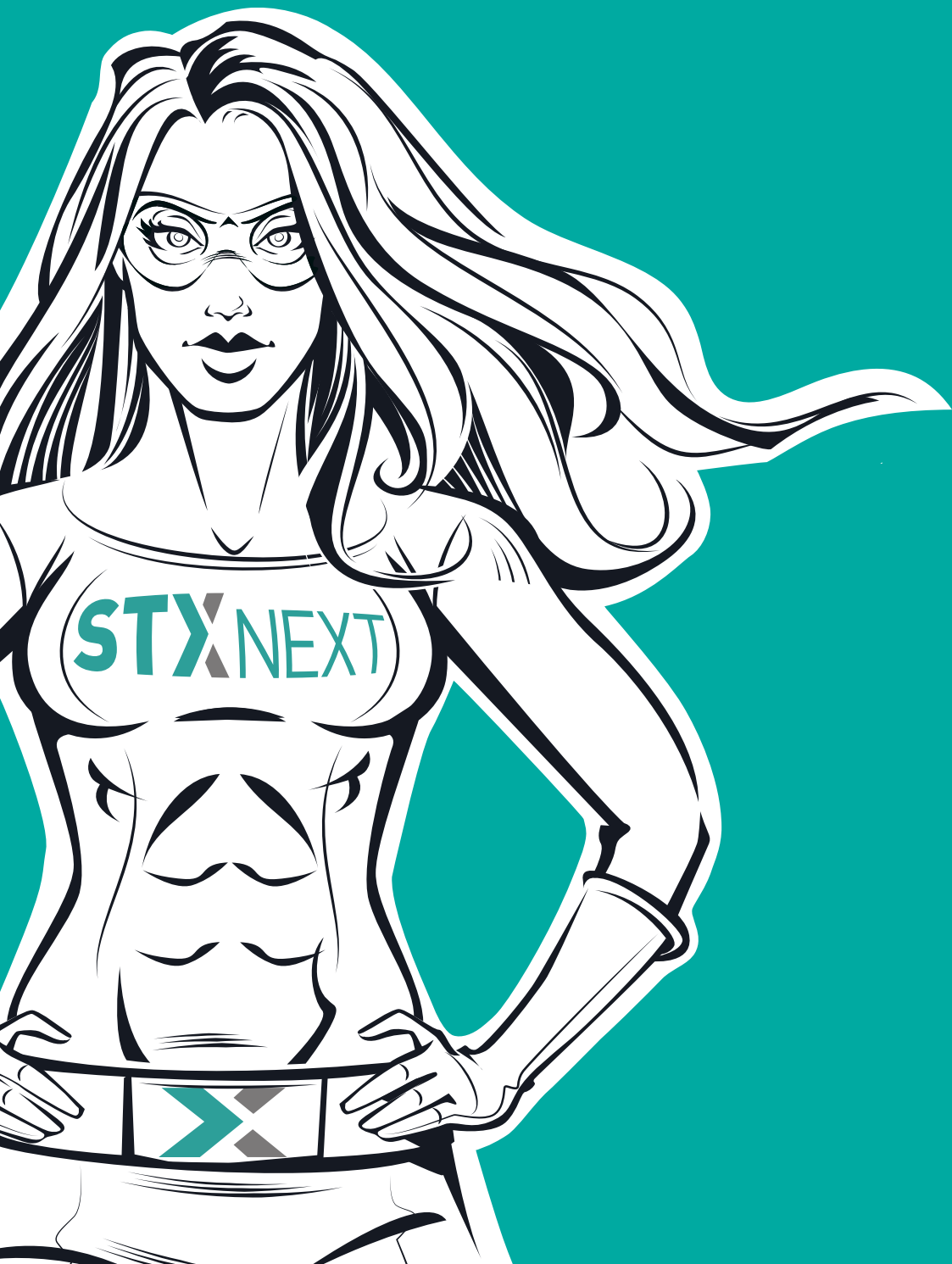
13:15 - 14:45

Część 3 - Standardy kodowania

15:00 - 16:30

Część 4 - Zabawa: GROT





# CZĘŚĆ 1 - PODSTAWY PYTHONA

---

syntax, pętle

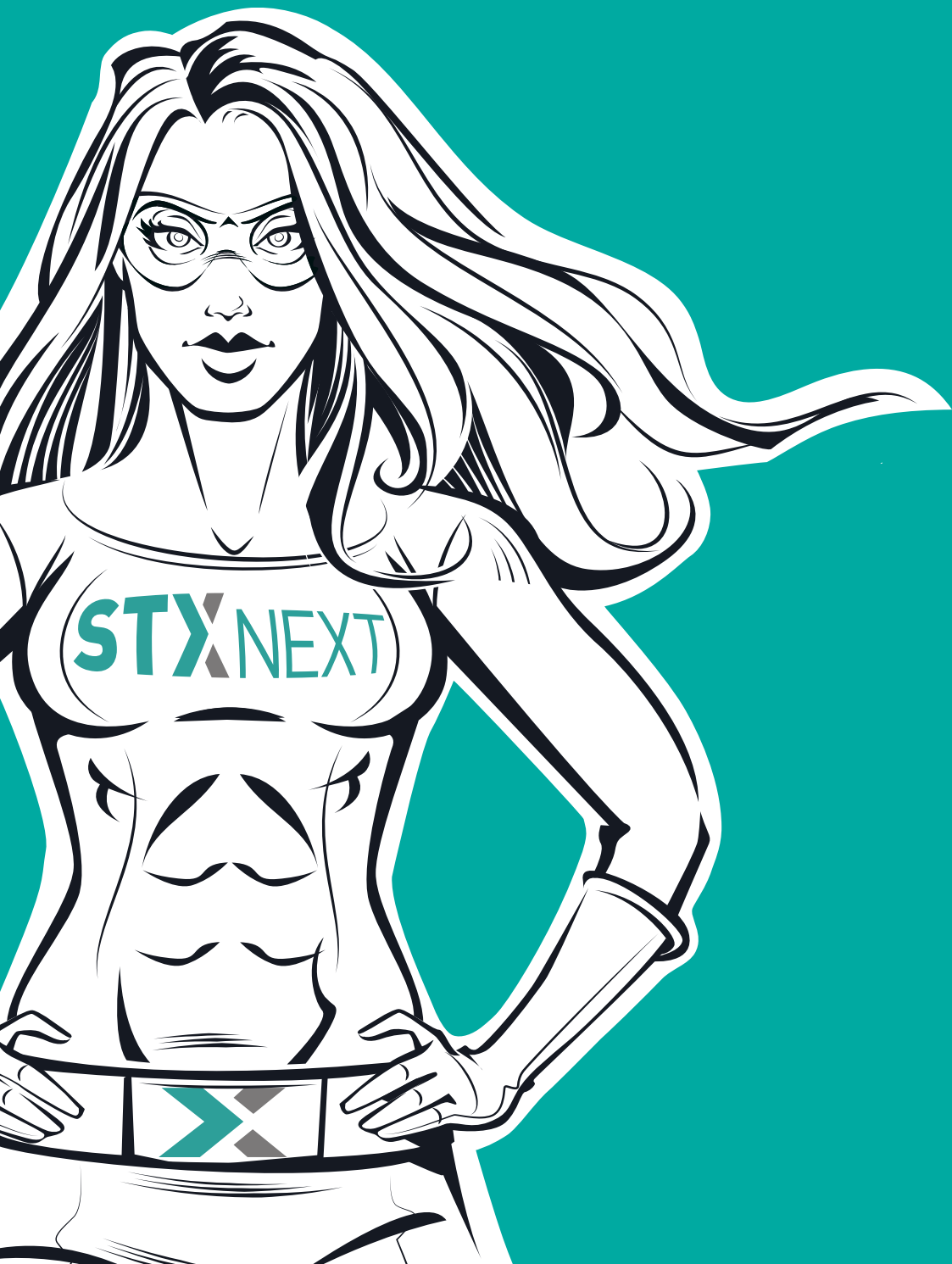
typy danych

deklaracja funkcji

importowanie modułów

powszechne operacje  
na typach danych

pdb



## CZĘŚĆ 2 - ADVANCED PYTHON

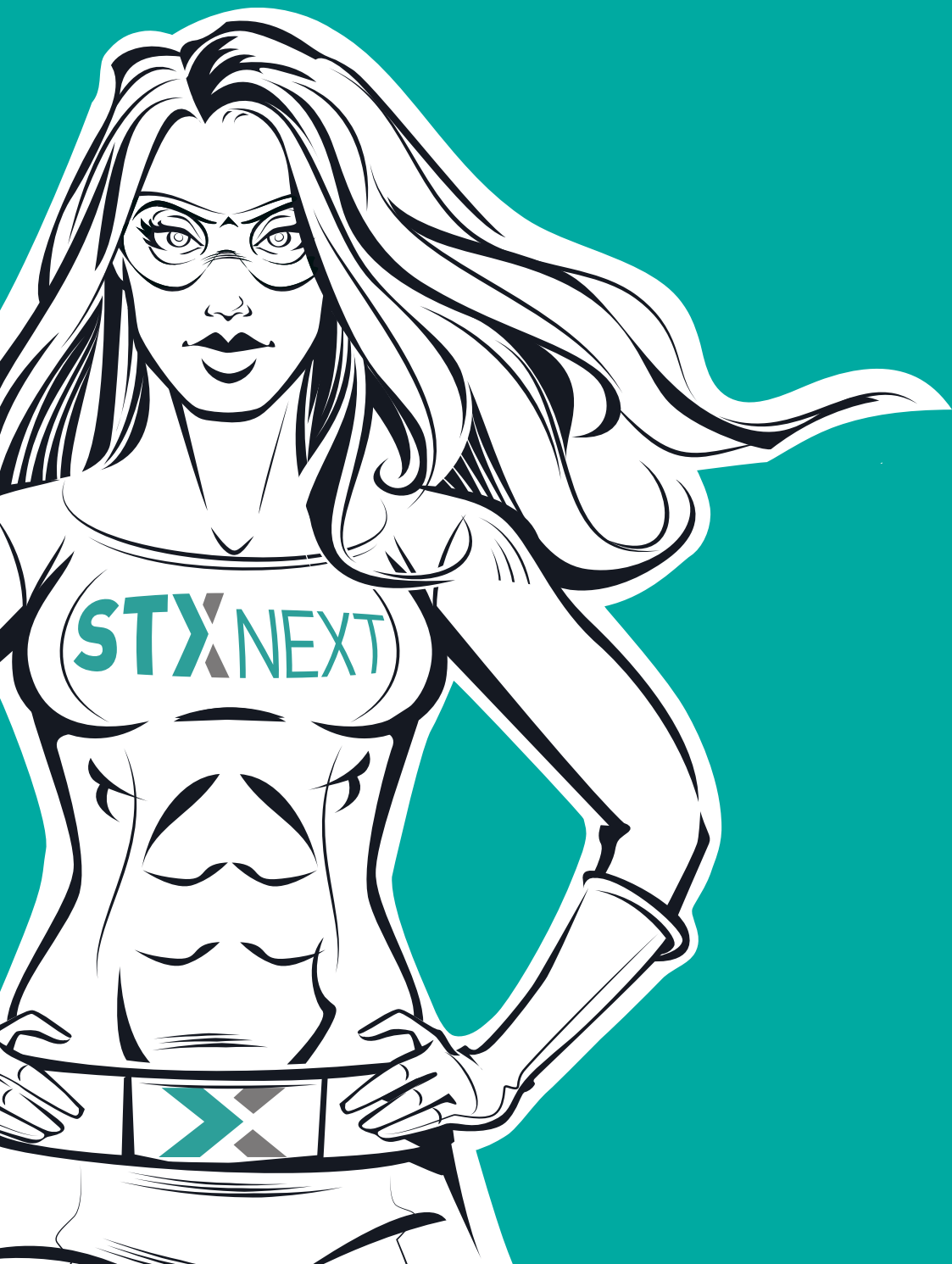
---

list/dict comprehension

generator

iteratory

przekazywanie args, kwargs



## CZĘŚĆ 3 - JAKOŚĆ KODU

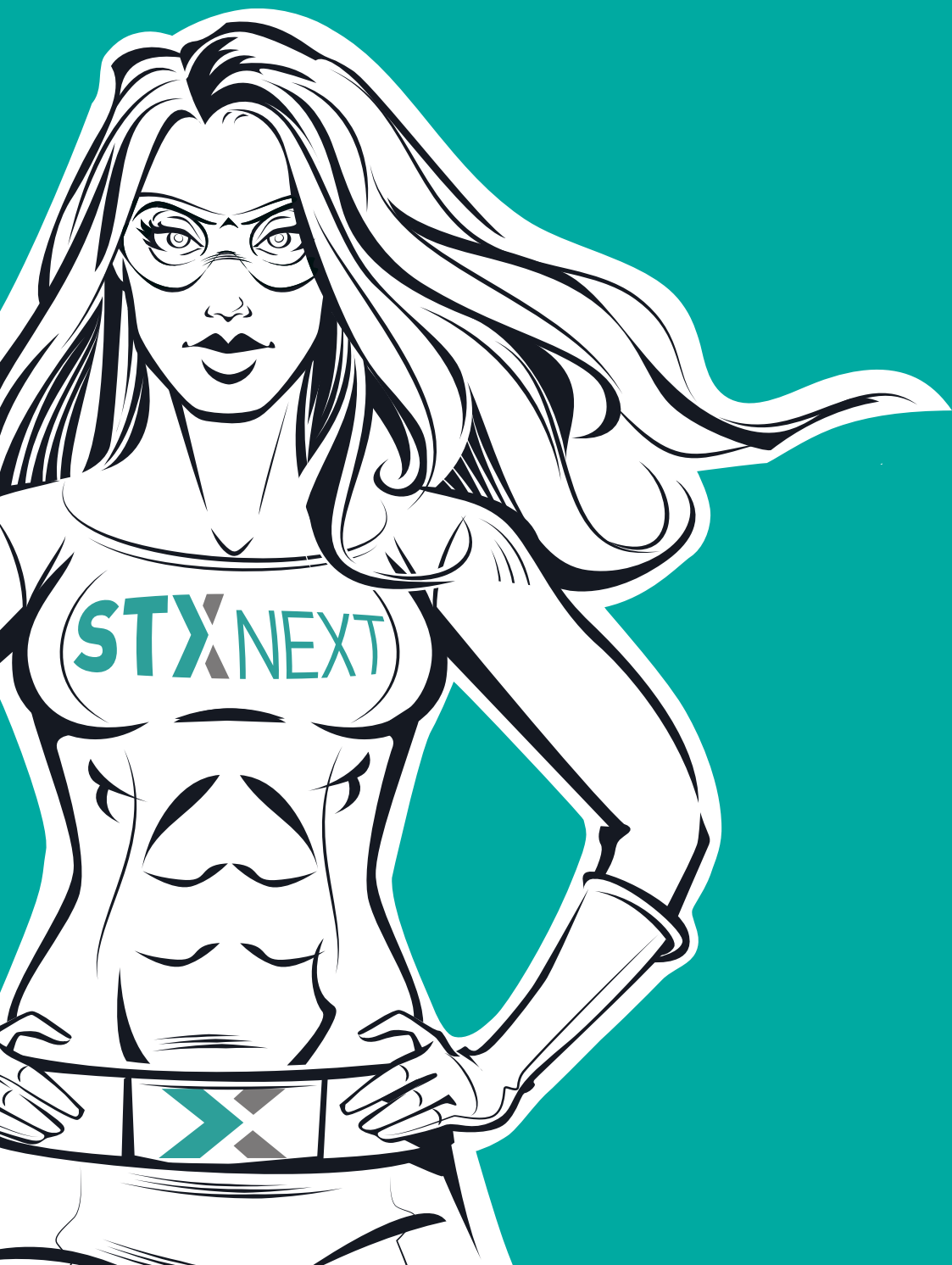
---

pep8

pylint

unittests

coverage



## CZĘŚĆ 4 - GROT

---

programowanie algorytmu  
do gry w Grota

# 1 - OGÓLNY POGLĄD NA SYNTAX

```
print "Witamy w Pythonie!"
```

```
num_var = 5
```

```
limit_var = 10
```

```
def first_function():
```

```
    print "Iteracja po " + str(limit_var) + " liczbach" # komentarz!
```

```
    for x in range(limit_var):
```

```
        if x == num_var:
```

```
            print "x równe {}".format(x)
```

```
        else:
```

```
            print ""
```

# 1 - TYPY DANYCH

---

numeryczne

boolean

ciągi

listy

słowniki

tuple

zbiory



# 1 - TYPY DANYCH

---

Wszystko jest obiektem

Część typów jest niemodyfikowalna (immutable)

# 1 - TYPY NUMERYCZNE

## Int, Long:

```
>>> i = 2
>>> type(i)
<type 'int'>
>>> l = 2L
>>> type(l)
<type 'long'>
```

## Float:

```
>>> a = 0.1
>>> type(a)
<type 'float'>
>>> b = 2e-1
>>> type(b)
<type 'float'>
>>> a + b
0.30000000000000004
```

## Complex:

```
>>> a = complex(2, 3)
>>> a
(2+3j)
>>> type(a)
<type 'complex'>
>>> b = complex(3, 4)
>>> b
(3+4j)
>>> type(b)
<type 'complex'>
>>> a + b
(5+7j)
```

# 1 - OBLICZENIA TYPÓW NUMERYCZNYCH

```
>>> 2 * 3
```

```
6
```

```
>>> 2 * 3L
```

```
6L
```

```
>>> 3 / 2
```

```
1
```

```
>>> 3 / 2L
```

```
1L
```

```
>>> 2 * 3.0
```

```
6.0
```

```
>>> 3 / 2.0
```

```
1.5
```

```
>>> 3.0 / 2L
```

```
1.5
```

```
>>> 3.0 * 2L
```

```
6.0
```

# 1 - TYP BOOLEAN

```
>>> t = True
>>> t
True
>>> type(t)
<type 'bool'>
>>> f = False
>>> f
False
>>> type(f)
<type 'bool'>
```

# 1 - TYP BOOLEAN

## W Pythonie jako *False* brane są:

- `None`
- `False`
- Zero jakiegokolwiek alfa-numerycznego typu np. `0`, `0L`, `0.0`, `0j`.
- Jakikolwiek pusty typ sekwencyjny np. `"`, `()`, `[]`.
- Jakiegokolwiek puste mapowanie np. `{}`.

## Operacje typu *Boolean* wedle priorytetu (rosnąco):

- `x or y` jeżeli `x` jest fałszem, wtedy brany jest `y`, w innym wypadku `x`
- `x and y` jeżeli `x` jest fałszem, wtedy brane jest `x`, w innym wypadku `y`
- `not x` jeżeli `x` jest fałszem, wtedy wyrażenie przyjmuje `True`, w innym wypadku `False`

# 1 - CIĄGI (ŁAŃCUCHY)

```
new_str = 'Test string!'
```

|                                   |  |
|-----------------------------------|--|
| <pre>print new_str[0]</pre>       | <pre># Pierwszy znak</pre>   |
| <pre>print new_str[0:3]</pre>     | <pre># Wyświetla wycinek. Pythonie elementy liczone są od zera</pre> |
| <pre>print new_str[2:]</pre>      | <pre># Wyświetla od trzeciego znaku do końca</pre>                   |
| <pre>print new_str * 2</pre>      | <pre># Wyświetla string dwa razy</pre>                               |
| <pre>print new_str + 'TEST'</pre> | <pre># Łączenie stringów</pre>                                       |

```
'String formatting test. Key {0} has value {1}'.format('first_key', 'Some value')
```

```
'String formatting test. {key} has value {value}'.format(key='first_key', value='Some value')
```

# 1 - LISTY

Przeznaczenie: modyfikowalne kolekcje

```
my_list = [ "first" ,786,2.23,"another" ]
```

```
print my_list[0 ]           # Pierwszy element listy
print my_list[1:3 ]         # Zakres elementów
print my_list[2:]           # Od trzeciego do końca
print [ "example_val" ] * 2 # Duplikuje wartości w liście
print my_list + [ "added_val" ] # Dodaje do siebie listy
```

Proste operacje na listach:

```
>>> l = [2, 3, 4]
>>> l.append(3.14)
>>> l
[2, 3, 4, 3.14]
>>> l.extend([4, 5, 6])
>>> l
[2, 3, 4, 3.14, 4, 5, 6]

>>> l.append([4, 5, 6])
[2, 3, 4, 3.14, 4, 5, 6, [4, 5, 6]]
```

# 1 - SŁOWNIKI

**Przeznaczenie:** mapowanie z szybkim wyszukiwaniem

```
{'key': 'val', 'second_key': 'second_val'}
```

```
my_dict = {}  
my_dict['one'] = "This is one"  
my_dict[2] = "This is two"
```

```
second_dict = {'name': 'Bond', 'code': 0.07}
```

```
print my_dict['one']  
print my_dict[2]  
print my_dict.get('some_value')  
print second_dict.keys()  
print second_dict.values()
```

```
# Wyświetla wartość klucza 'one'  
# Wyświetla wartość klucza 2  
# Wyświetla wartość danego klucza  
# Wyświetla klucze słownika  
# Wyświetla wartości słownika
```



# 1 - SŁOWNIKI: PROSTE OPERACJE

```
>>> d = {'a': 'b', 3: 4}
>>> d
{'a': 'b', 3: 4}
>>> d['a']
'b'
>>> d[3]
4
```

```
>>> 'a' in d
True
>>> 'b' in d
False
>>> d[u'x'] = u'y'
>>> d
{'a': 'b', u'x': u'y', 3: 4}
>>> del d['a']
>>> d
{u'x': u'y', 3: 4}
```

# 1 - TUPLE (KROTKI)

**Przeznaczenie:** niemodyfikowalna sekwencja danych o stałej strukturze (immutable)

```
my_tuple = ( 'one', 786, 2.23, 'another' )
```

```
print my_tuple[0]
```

```
print my_tuple[1:3]
```

```
print my_tuple[2:]
```

```
print my_tuple * 2
```

```
print my_tuple + ('added_value',)
```

# Pierwszy element

# Zakres elementów

# Od 3 do końca

# Powiela elementy

# Łączy tuple

Przecinek na końcu

- jednoelementowa tupla!

# 1 - PROSTE OPERACJE NA TUPLACH

```
>>> point = (3, 4)
```

```
>>> point[0]
```

```
3
```

```
>>> point[2]
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

**IndexError:** tuple index out of range

```
>>> len(point)
```

```
2
```

```
>>> x, y = point
```

```
>>> x
```

```
3
```

```
>>> y
```

```
4
```

```
>>> point[0] = 2
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

**TypeError:** 'tuple' object does not support item assignment

# 1 - ZBIORY

**Przeznaczenie:** kolekcja unikalnych elementów

**# zbiór zmienny i niezmienny**

```
set_a=set([1,2,3])
```

```
set_a={1,2,3}
```

```
set_b=frozenset([2,3,4])
```

**# dodaje element do zbioru**

```
set_a.add(123)
```

**# usuwa element ze zbioru**

```
set_a.discard(1)
```

**# sprawdza czy zbiór a jest nadzbiorem b**

```
set_a>=set_b
```

# 1 - DEFINIOWANIE FUNKCJI

```
def print_word(word):  
    print word
```

```
def print_args(*args):  
    for index, value in enumerate(args):  
        print '{0}. {1}'.format(index, value)
```

```
print_args('apple', 'banana', 'cabbage')
```

```
def print_kwargs(**kwargs):  
    for name, value in kwargs.items():  
        print '{0} is {1}'.format(name, value)
```

```
print_kwargs(apple = 'fruit', cabbage = 'vegetable')
```

```
def print_default_value(apple, cabbage='vegetable', **kwargs):  
    for name, value in kwargs.items():  
        print '{0} is {1}'.format(name, value)  
    print 'Apple is {}'.format(apple)  
    print 'Cabbage is {}'.format(cabbage)
```

```
print_default_value(apple = 'fruit')  
lub  
print default_value('fruit')
```

# 1 - IMPORTOWANIE W PYTHONIE

```
import math  
math.floor(x)  
dir(math)
```

# importuje bibliotekę math

# zwraca dostępne nazwy (np. metod) w obrębie zaimportowanej biblioteki

```
from math import floor  
from math import *  
floor(x)
```

# importuje metodę floor z biblioteki math

# importuje wszystkie metody z biblioteki math (niezalecane)

# po obydwóch powyższych importach metody można użyć już bezpośrednio

Kolejność importowania to:

- biblioteki systemowe, wbudowane,
- zewnętrzne frameworki/narzędzia (instalowane np. przez pip),
- lokalne np. moduły naszej aplikacji (wyszukiwane w folderze projektu).

```
import math  
import sys
```

```
from django.forms import *
```

```
import our_application.module
```

# 1 - CZĘSTE OPERACJE NA TYPAH DANYCH

# program losuje zbiór 6 unikatowych  
liczb od 1 do 49

```
from random import choice
```

```
wylosowane = set()
```

```
while len(wylosowane) < 6:  
    wylosowane.add(choice(range(1,50)))
```

```
for x in wylosowane:  
    print x
```

# przykład iteracji po słownikach

```
dishes = {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}
```

```
for val in dishes.values():  
    print val
```

```
for key in dishes.keys():  
    print key
```

```
for key, val in dishes.items():  
    print key, val
```

# 1 - CZĘSTE OPERACJE NA TYPACH DANYCH

# Lista Ocen Studentów przy wykorzystaniu klas

```
class Student:
```

```
    imie=""
```

```
    nazwisko=""
```

```
    ocena=0.0
```

```
studenci = []
```

```
while True:
```

```
    nazwisko = raw_input('Podaj nazwisko studenta (pusta wartosc=koniec): ')
```

```
    imie = raw_input('Podaj imie studenta > ')
```

```
    ocena = raw_input('Podaj ocene studenta > ')
```

```
    if not(nazwisko and imie and ocena):
```

```
        break
```

```
    student = Student()
```

```
    student.nazwisko = nazwisko
```

```
    student.imie = imie
```

```
    student.ocena = float(ocena)
```

```
    studenci.append(student)
```

```
for idx, student in enumerate(studenci):
```

```
    print '{}. {} {} {}'.format(idx+1, student.nazwisko, student.imie, student.ocena)
```



# 1 - DEBUGOWANIE ZA POMOCĄ PDB

Wbudowane w Pythona moduł debuggera, który można wykorzystać wszędzie.

Rozszerzenia: ipdb, rpdb, wiele innych.

```
import pdb; pdb.set_trace()
```

Na c9.io snippet: pd [TAB]

# 1 - ZADANIA

Napisać funkcję, która wypisze litery alfabetu w osobnych liniach,

Napisać funkcję, która wypisze co drugą literę alfabetu,

Napisać funkcję rozpoznającą palindromy,

Napisać funkcję odfitrowującą znaki spoza alfabetu,

Przy użyciu *filter* napisać funkcję, który z podanej listy słów odfiltruje słowa dłuższe niż *n* znaków.

## 2 - PYTHONICZNOŚĆ

Python zawiera dużo ułatwień w syntax'ie.

Każdy efekt uzyskany przy użyciu specjalnego syntaxu da się uzyskać bez niego.

## 2 - COMPREHENSIONS

[ **EXPRESSION** for **VARIABLE** in **SEQUENCE** if **CONDITION** ]

[ **n\*\*2** for **n** in **range(10)** if **n%2==0** ]

[ **n** for **n** in **range(10)** ]

↑  
Warunek  
jest opcjonalny!

Jednakowy efekt zawsze da się uzyskać bez stosowania comprehensions.  
Jest to skrótowy zapis.

## 2 - COMPREHENSIONS

[ **EXPRESSION** for **VARIABLE** in **SEQUENCE** if **CONDITION** ]

KAŻDE z wymienionych wyżej miejsc może zawierać wyrażenia Pythonowe.  
Zarówno wywołania funkcji jak i wielokrotne przypisanie itp.!

```
[n.upper() for i, n in enumerate(string.ascii_lowercase) if i%2 == 0]
```

## 2 - PRAKTYKA

Użyj list comprehension, aby uzyskać listę liczb od 0 do 100.

Użyj list comprehension, aby uzyskać listę liczb parzystych od 0 do 100.

## 2 - CZYM MOŻE BYĆ SEKWENCJA?

Wszystko, po czym da się iterować:

- ciągi znaków,
- generatory,
- własne iteratory itd.

```
>>> [letter for letter in 'string']  
['s', 't', 'r', 'i', 'n', 'g']
```

Analogia do PHP:

**Interface Iterator.**

## 2 - SYNTAX DICT COMPREHENSION

Zamiast nawiasów kwadratowych - klamry.

Klucz : wartość w EXPRESSION.

```
>>> {key: value for key, value in enumerate('abcde')}  
{0: 'a', 1: 'b', 2: 'c', 3: 'e'}
```



## 2 - ZADANIA

Użyj list comprehension i obiektu string aby uzyskać listę liter alfabetu.

Użyj dict comprehension i obiektu string aby uzyskać słownik zawierający wszystkie litery alfabetu jako klucze i ich kody ascii jako wartości. (użyj - ord)

Użyj list comprehension aby z listy słów odfiltrować te, których długość jest mniejsza niż 5.

## 2 - LAMBDA

Lambda jest funkcją definiowaną “w locie”.

lambda **VARIABLES**: **EXPRESSION**

lambda **x**: **x\*\*2**

lambda **x,y,z**: **(x\*\*2 + y) / z**

## 2 - LAMBDA

Np. użycie odpowiedniego indeksu podczas sortowania.

```
data = [{  
    'name': 'Book',  
    'price': '10',  
}, {  
    'name': 'Phone',  
    'price': '20',  
}, {  
    ...  
}]
```

```
sorted(data, key=lambda x: x['price'])
```

## 2 - ZADANIA

---

Użyj list comprehension, aby uzyskać kwadraty liczb od 1 do 50.

Użyj map oraz lambda, aby uzyskać ten sam efekt.

## 2 - GENERATOR - YIELD

```
def squares():
```

```
    x = 1
```

```
    while True:
```

```
        yield x
```

```
        yield x**2
```

```
        x = x + 1
```

```
sq = squares()
```

```
for x in range(100):
```

```
    import pdb; pdb.set_trace() # put PDB here and step into functions
```

```
    print "number %s" % sq.next()
```

```
    print "square %s" % sq.next()
```

## 2 - GENERATORY

Funkcje mają tylko jedną szansę na zwrócenie wyników  
- dlatego muszą zwracać cały wynik na raz.

Generatory mogą zwracać wyliczony wynik co iterację.

## 2 - ZADANIA

Napisz generator zwracający dni w odstępach  
jednodniowych od dzisiejszej daty wzwyż.

Następnie użyj list comprehension, aby wypisać  
tylko dni robocze. (*Uwaga na nieskończone pętle!*)

## 2 - INTROSPEKCJA KODU

Python udostępnia szereg metod umożliwiających badanie obiektów w trakcie działania programu.

Wszystko w Pythonie jest obiektem, zatem wszystko można zbadać pod względem posiadanych metod itp.

`dir`, `callable`, `getattr`



## 2 - DIR, CALLABLE, GETATTR

Zbadaj kilka obiektów funkcją dir

Spróbuj zbadać, czy dany obiekt można wywoływać przy pomocy callable

Spróbuj pobrać kilka atrybutów obiektu przy pomocy getattr.

## 2 - ZADANIE

Napisz funkcję która przyjmuje na wejściu obiekt i wypisuje nazwy wszystkich atrybutów danego obiektu.

Następnie przefiltruj tylko te, które można wywołać (callable).

Napisz funkcję która wypisze na ekran wszystkie metody danego obiektu wraz z ich docstringiem.

# 3 - JAKOŚĆ KODU W PYTHONIE

DRY

Dokumentacja opisująca styl (PEP8)

Analizowanie kodu pod względem stylu pisania  
oraz struktury (pylint)

Testy jednostkowe i pokrycie kodu testami  
(unittest i coverage)

# 3 - PEP8

Dokument zawierający wskazówki dotyczące stylu kodowania

Wpływa na jakość kodu

Poprawia czytelność kodu

Google style guide

# 3 - PEP8 - PRZYKŁADY

## Poprawnie:

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

```
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)
```

```
foo = long_function_name(  
    var_one, var_two,  
    var_three, var_four)
```

## Źle:

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

```
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)
```

# 3 - PEP8 - URUCHOMIENIE I SPRAWDZENIE

```
username@demo-project:~/workspace (master) $ sudo pip install pep8
```

```
Downloading/unpacking pep8
```

```
  Downloading pep8-1.5.7-py2.py3-none-any.whl
```

```
Installing collected packages: pep8
```

```
Successfully installed pep8
```

```
Cleaning up...
```

```
username@demo-project:~/workspace (master) $ pep8 python/hello.py
```

```
username@demo-project:~/workspace (master) $ pep8 python/hello.py
```

```
python/hello.py:8:62: W292 no newline at end of file
```

```
username@demo-project:~/workspace (master) $ pep8 --exclude=*.pyc python/*  
)
```

# 3 - PYLINT

Sprawdzenie powtarzających się zmiennych

Sprawdzenie importów

Sprawdzenie zmiennych i ich nazw

Sprawdzenie funkcji

Jakość kodu wyświetlona x.x/10

# 3 - PYLINT - RODZAJE BŁĘDÓW

- \* (C) convention - dotyczy standardów kodowania
- \* (R) refactor - dotyczy refactoru kodu (cycle-imports)
- \* (W) warning - dotyczy błędów pythonowych
- \* (E) error - dotyczy błędów w kodzie
- \* (F) fatal - dotyczy błędów w wykonywaniu sprawdzania pylintem

Najczęściej pojawiają się błędy type W, E i C. Błędy C bardzo podobne do błędów, które wyświetla PEP8.



# 3 - PYLINT - URUCHOMIENIE I BLOKOWANIE

```
username@demo-project:~/workspace (master) $ sudo pip install pylint
Requirement already satisfied (use --upgrade to upgrade): pylint in /usr/lib/python2.7/dist-packages
Cleaning up...
```

Blokowanie pylint:

globalne - na początku pliku podajemy linię:

```
# pylint: disable=C0103,C0302 lub # pylint: disable=unused-variable
```

lokalne

można zablokować pylint w całej funkcji czy metodzie

można zablokować pylint w jednej linii

w całym projekcie jako .pylintrc

Przykład:

[MASTER]

```
ignore=fixtures,migrations,local.py,settings.py,urls.py
```

[MESSAGES CONTROL]

```
disable=C0111,I0011,I0012,W0142,W0212,W0232,W0613,R0801,R0901,F0401,E1101,E1002,W1401,
R0903,R0904,E1103,C1001,R0201
```

[REPORTS]

```
report=yes
```

# 3 - TESTY JEDNOSTKOWE

---

Sprawdzanie poprawności kodu

Dokumentacja

Przypadki użycia

Obsługa sytuacji skrajnych

Pomiary (pokrycie testami)

TDD (Test Driven Development)

# 3 - TESTY JEDNOSTKOWE - PRZYGOTOWANIE I URUCHOMIENIE

---

Minimalny test:

```
import unittest
```

```
class MathTest(unittest.TestCase):
```

```
    def test_add(self):
```

```
        self.assertEqual(2 + 3, 5)
```

```
    def test_sub(self):
```

```
        self.assertEqual(5 - 3, 2)
```

```
if __name__ == '__main__':
```

```
    unittest.main()
```

Uruchomienie:

```
username@demo-project:~/workspace (master) $ python python/tests/test.py
```

```
..
```

```
-----
```

```
Ran 2 tests in 0.000s
```

OK

# 3 - TESTY JEDNOSTKOWE - BŁĘDY

```
username@demo-project:~/workspace (master) $ python python/tests/test.py
```

```
.F
```

```
=====
```

```
FAIL: test_sub (__main__.MathTest)
```

```
-----
```

```
Traceback (most recent call last):
```

```
File "python/tests/test.py", line 9, in test_sub
```

```
    self.assertEqual(5 - 3, 3)
```

```
AssertionError: 2 != 3
```

```
-----
```

```
Ran 2 tests in 0.001s
```

```
FAILED (failures=1)
```

# 3 - TESTY JEDNOSTKOWE - ASERCJE

```
self.assertEqual(2 + 3, 5)
self.assertAlmostEqual(0.1 + 0.2, 0.3, delta=1e-6)
self.assertNotEqual('żółw', u'Żółw')
self.assertTrue([0])
self.assertFalse([])
x = []
y = x
self.assertIs(x, y)
self.assertIn('x', ['x'])
self.assertIsInstance([], list)
self.assertIsNone(None)
self.assertItemsEqual((2, 3), [2, 3])
```

# 3 - TESTY JEDNOSTKOWE - COVERAGE

```
username@demo-project:~/workspace (master) $ sudo pip install coverage
username@demo-project:~/workspace (master) $ coverage run python/hello.py
Hello world from Cloud9!
username@demo-project:~/workspace (master) $ coverage report -m
Name      Stmts Miss Cover Missing
-----
python/hello  1    0 100%
```

username@demo-project:~/workspace (master) \$ coverage html

# 3 - TESTY JEDNOSTKOWE - ZADANIE

Napisać metodę, która przyjmuje dwa parametry, słowo i literę, a następnie sprawdza czy dana litera jest w słowie i zwraca True lub False. W przypadku gdy parametr jest inny niż litera, zwrócić odpowiednią informację. Dopisać testy jednostkowe dla możliwie jak największej liczby przypadków (np. podanie liczby, podanie dwóch liter, brak podanego parametru)

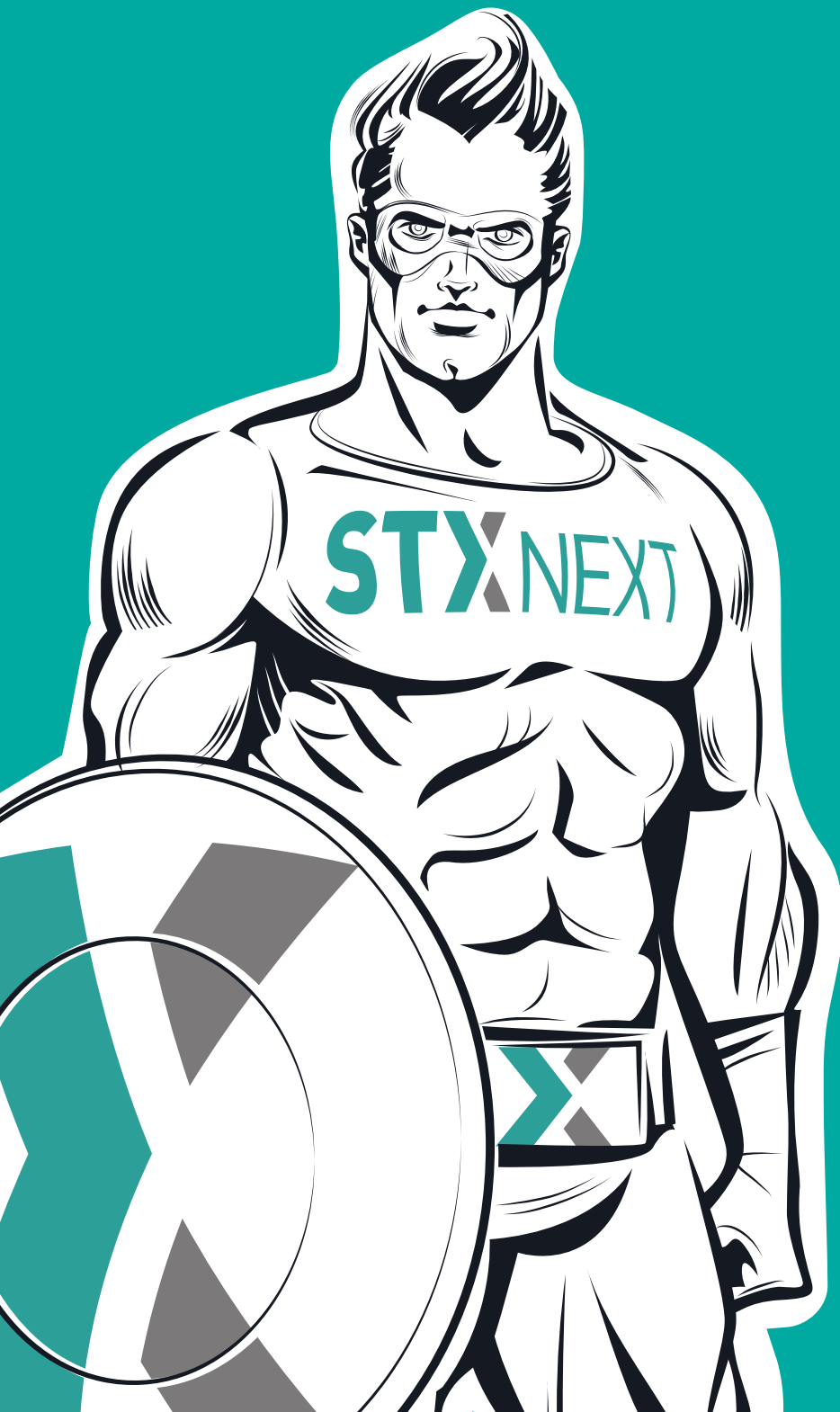
wykorzystać poprzednią metodę, aby była możliwość podania listy słów i zwrócić listę True/False, dopisać test.

## 4 - GROT - NO TO GRAMY

---

<http://grot.hackathons.stxnext.pl/>





[www.stxnext.pl](http://www.stxnext.pl)

[www.facebook.com/stxnext](https://www.facebook.com/stxnext)