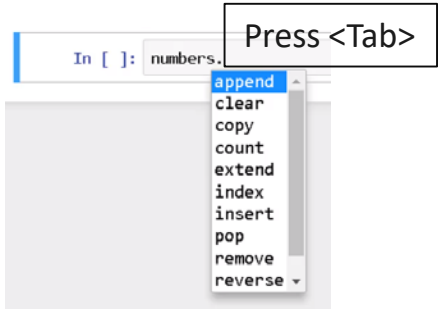
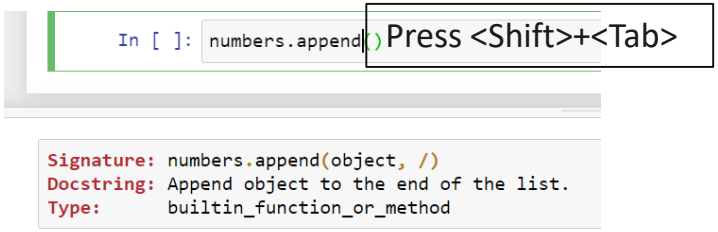
Python Essentials. Lection 2

Python Essentials. Functions intro.

For example in Python object “list” has a definite number of methods.

```
In [7]: numbers = [1,2,3]
        numbers.append(4)

In [8]: print(numbers)

        [1, 2, 3, 4]

In [10]: help (numbers.append)

         Help on built-in function append:

         append(object, /) method of builtins.list instance
             Append object to the end of the list.
```





Python Essentials. Built-in functions

There a lot of built-in functions in Python. Some of them are shown below.

```
In [11]: abs(-1)
Out[11]: 1

In [12]: abs(1)
Out[12]: 1

In [13]: max(1,2,3,4,5,6,7,8,9)
Out[13]: 9

In [14]: min(1,2,3,4,5)
Out[14]: 1

In [15]: pow(2, 10)
Out[15]: 1024

In [16]: round(3,3789)
Out[16]: 3

In [19]: round(3.3789, 1)
Out[19]: 3.4
```

```
In [2]: sum([1,2,3,4])
Out[2]: 10

In [6]: all_true=all([True, True, True])
        print(all_true)

        True

In [7]: not_all_true=all([True, False])
        print(not_all_true)

        False

In [8]: students=[('Ivan', 181), ('Dmytro', 178),
                  ('Olena', 175), ('Mykyta', 169)]

In [9]: all(rating > 170 for _, rating in students)
Out[9]: False

In [10]: all(rating > 165 for _, rating in students)
Out[10]: True

In [11]: any_true=any([False, True, False])
         print(any_true)

         True
```

```
In [13]: code = ord('a')
         code
Out[13]: 97

In [14]: c = chr(code)
         c
Out[14]: 'a'
```

Python Essentials. “HowTo” fuction.

Main aim of any function is to solve the problem and to be reusable.

No arguments

```
In [16]: def jobsdone():
             print("Job's done")
         jobsdone()

         Job's done

In [17]: jobsdone
Out[17]: <function __main__.jobsdone()>

In [18]: help(jobsdone)

         Help on function jobsdone in module __main__:

         jobsdone()
```

Brief info for help()

```
In [22]: def jobsdone():
             '''
             DOCSTRING: about
             INPUT: none
             OUTPUT: "Job's done"
             '''
             print("Job's done")
         jobsdone()

         Job's done

In [23]: help(jobsdone)

         Help on function jobsdone in module __main__:

         jobsdone()
             DOCSTRING: about
             INPUT: none
             OUTPUT: "Job's done"
```

With arguments

```
In [24]: def print_name(name):
             print(name)

In [25]: print_name('Kuzma')

         Kuzma

In [26]: print_name()

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-26-43fda9c5862d> in <module>
         ----> 1 print_name()

         TypeError: print_name() missing 1 required positional argument: 'name'

In [27]: def print_name(name='None'):
             print(name)

In [28]: print_name()

         None

In [29]: result = print_name()
         print(result)
         print(type(result))

         None
         None
         <class 'NoneType'>
```

Python Essentials. “HowTo” fuction.

Main aim of any function is to solve the problem and to be reusable.

Map works with functions as arguments

```
def square(number):
    return number*number
numbers=[1,2,3,4,5]
mapped_seq=map(square, numbers)
for i in mapped_seq:
    print(i)
```

```
1
4
9
16
25
```

```
print(type(mapped_seq))
```

```
<class 'map'>
```

```
list(map(square, numbers))
```

```
[1, 4, 9, 16, 25]
```

Filter also works with functions as arguments

```
def is_adult(age):
    return age>=18
ages= [14,18,21,16,30]
filter(is_adult, ages)
```

```
<filter at 0x21fa7e20d00>
```

```
list(filter(is_adult, ages))
```

```
[18, 21, 30]
```

Python Essentials. “HowTo” fuction.

Main aim of any function is to solve the problem and to be reusable.

Map works with functions as arguments

```
def square(number):
    return number*number
numbers=[1,2,3,4,5]
mapped_seq=map(square, numbers)
for i in mapped_seq:
    print(i)
```

```
1
4
9
16
25
```

```
print(type(mapped_seq))
```

```
<class 'map'>
```

```
list(map(square, numbers))
```

```
[1, 4, 9, 16, 25]
```

Filter also works with functions as arguments

```
def is_adult(age):
    return age>=18
ages= [14,18,21,16,30]
filter(is_adult, ages)
```

```
<filter at 0x21fa7e20d00>
```

```
list(filter(is_adult, ages))
```

```
[18, 21, 30]
```

```
is_adult = lambda age: age>=18
list(filter(is_adult, ages))
```

```
[18, 21, 30]
```

Python Essentials. “HowTo” fuction.

Variables and scopes. In mostly cases global variable is not needed.

Beware of reassigning built-in variables.

```
greeting = "Hello from the global scope"

def greet():
    greeting = "Hello from enclosing scope"

    def nested():
        greeting = "Hello from local scope"
        print(greeting)
    nested()
```

```
greet()
print(greeting)
```

```
Hello from local scope
Hello from the global scope
```

```
greeting = "Hello from the global scope"

def greet():
    greeting = "Hello from enclosing scope"

    def nested():
        #greeting = "Hello from local scope"
        print(greeting)
    nested()
```

```
greet()
print(greeting)
```

```
Hello from enclosing scope
Hello from the global scope
```

```
greeting = "Hello from the global scope"

def greet():
    #greeting = "Hello from enclosing scope"

    def nested():
        #greeting = "Hello from local scope"
        print(greeting)
    nested()
```

```
greet()
print(greeting)
```

```
Hello from the global scope
Hello from the global scope
```

Python Essentials. “HowTo” fuction.

Functions with function, as an argument. Decorators

```
def say_something(func):
    func()

def hello_world():
    print('Hello, world!')

say_something(hello_world)
```

```
Hello, world!
```

```
def log_decorator(func):
    def wrap():
        print(f'Calling func {func}')
        func()
        print(f'Func {func} finished its work')
    return wrap
```

```
def hello():
    print('hello, world!')
```

```
wrapped_by_logger = log_decorator(hello)
wrapped_by_logger()
```

```
Calling func <function hello at 0x00000190B5072EA0>
hello, world!
Func <function hello at 0x00000190B5072EA0> finished its work
```

```
@log_decorator
def hello():
    print('hello, world!')
```

```
hello()
```

```
Calling func <function hello at 0x00000190B54BED08>
hello, world!
Func <function hello at 0x00000190B54BED08> finished its work
```

Python Essentials. Errors and exceptions.

Errors and exceptions.

```
def devide(a, b):
    return a/b
devide(9,3)
```

```
3.0
```

```
devide(9,0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-68-0f1c3f4afa01> in <module>
----> 1 devide(9,0)

<ipython-input-67-770b5f8af7e9> in devide(a, b)
      1 def devide(a, b):
----> 2     return a/b
      3 devide(9,3)

ZeroDivisionError: division by zero
```

```
def devide(a, b):
    try:
        return a/b
    except ZeroDivisionError as exception:
        print(f'An error occured: {exception}')
devide(9,3)
```

```
3.0
```

```
devide(9,0)
```

```
An error occured: division by zero
```

```
def devide(a, b):
    try:
        return a/b
    except ZeroDivisionError as exception:
        print(f'An error occured: {exception}')
    except:
        print('An unknown error occured')
divider = input()
devide(9, divider)
```

```
po
An unknown error occured
```

Python Essentials. Unit Testing basics.

FizzBuzz and unit test for it

```python
def get_reply(number):
    if number%5==0 and number%3==0:
        return 'FizzBuzz'
    elif number%3==0:
        return 'Fizz'
    elif number%5==0:
        return 'Buzz'
    else:
        return ''
```

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\User>cd C:\Users\User\Google Drive\EPAM_CI_CD\Python Essentials\Python3.8root

(base) C:\Users\User\Google Drive\EPAM_CI_CD\Python Essentials\Python3.8root>python fizz_buzz_tests.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK

(base) C:\Users\User\Google Drive\EPAM_CI_CD\Python Essentials\Python3.8root>
```

```python
import unittest
import fizz_buzz

class FizzBuzzTests(unittest.TestCase):

    def test_fizz(self):
        number=6

        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'Fizz')

    def test_buzz(self):
        number=10

        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'Buzz')

    def test_fizzbuzz(self):
        number=15
        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'FizzBuzz')

if __name__ == '__main__':
    unittest.main()
```

Python Essentials. Unit Testing basics.

FizzBuzz and unit test for it

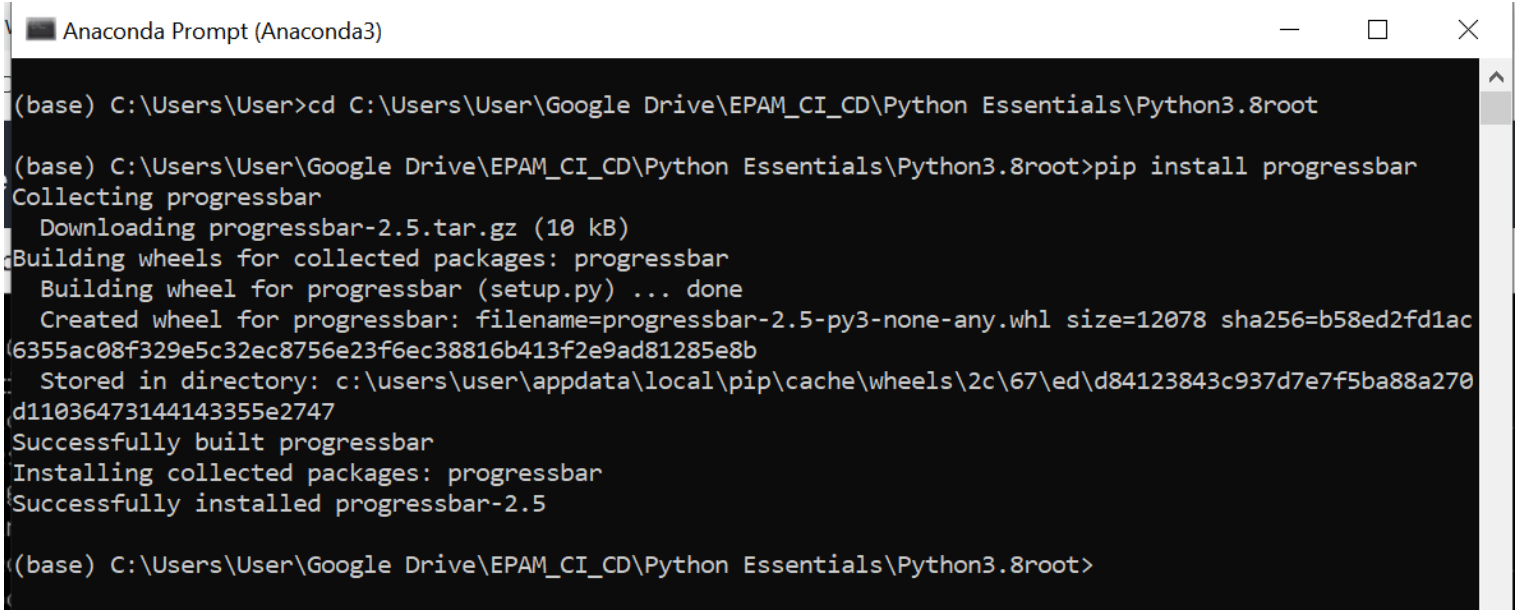
```
def get_reply(number):
    if number%5==0 and number%3==0:
        return 'FizzBuzz'
    elif number%3==0:
        return 'Fizz'
    elif number%5==0:
        return 'Buzz'
    else:
        return ''
```

```
import unittest
import fizz_buzz

class FizzBuzzTests(unittest.TestCase):

    def test_fizz(self):
        number=6

        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'Fizz')

    def test_buzz(self):
        number=10

        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'Buzz')

    def test_fizzbuzz(self):
        number=15
        result = fizz_buzz.get_reply(number)

        self.assertEqual(result, 'FizzBuzz')

if __name__ == '__main__':
    unittest.main()
```

Python Essentials. Modules, packages, libraries.

PyPi - open source repository of libraries (packages)

PiP - installer for open source repository of libraries (packages) PyPi



Python Essentials. Fist steps in OOP.

```
class Character():

    def __init__(self, race, damage=10, armor=20):
        self.race = race
        self.damage = damage
        self.armor = armor
```

```
unit = Character("Elf", damage=20, armor=40)
type(unit)
```

```
__main__.Character
```

```
unit.race
```

```
'Elf'
```

```
print(unit.damage)
print(unit.armor)
```

```
20
40
```

```
class Character():
    max_speed = 100
    dead_health = 0

    def __init__(self, race, damage=10, armor=20):
        self.race = race
        self.damage = damage
        self.armor = armor
        self.health = 100

    def hit(self, damage):
        self.health -= damage

    def is_dead(self):
        return self.health == Character.dead_health
```

```
unit = Character('Ork')
print(unit.race)
print(Character.max_speed)
```

```
Ork
100
```

```
unit.hit(20)
print(unit.health)
```

```
80
```

Q & A


Thank you!