Osnova

Lesson Overview

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **LESSON OVERVIEW**

Hi buddy, welcome in lesson 6! You have gone quite a long way, just look, what you know so far!

- Data types,

- conditions,

- loops.

I believe, we can start with something a little bit harder.

Functions! Why are they so important?

- Functions can repeat any code you type. You just need to know how to type it properly.

- By using function, you can save many effort and huge amount of time.

- This is very important subject and it serves as a base for next lessons. Keep it in mind, please.

100% z Lekce 7

Osnova

00:32

REVIEW EXERCISES

Fizz Buzz

Write a program that prints the integers from 1 to 100 (inclusive).

100% z Lekce 7

- for multiples of three, print **Fizz** (instead of the number)

    Osnova

- for multiples of five, print **Buzz** (instead of the number)

- for multiples of both three and five, print **FizzBuzz** (instead of the number)

Example of running the program for numbers between 10 to 20:

```
~/PythonBeginner/Lesson6 $ python fizzbuzz.py
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
```

Online Python Editor

```
1 |
```

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution ▲

```python
1  for i in range(1, 101):
2      if i % 15 == 0:
3          print("FizzBuzz")
4      elif i % 3 == 0:
5          print("Fizz")
6      elif i % 5 == 0:
7          print("Buzz")
8      else:
9          print(i)
```

Chessboard

PYTHON ACADEMY / 6. INTRODUCTION TO FUNCTIONS / REVIEW EXERCISES / CHESSBOARD

Write a program that prints a chessboard of given size to the terminal. The program inputs should be

100% z Lekce 7

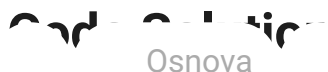2. ... ... at ... " serve to fill in the black squares
        Osnova

Example of running the program for the edge length of 5 and fill character "#":

```
~/PythonBeginner/Lesson6 $ python chessboard.py
# # #
 # #
# # #
 # #
# # #
```

Online Python Editor

```
1  |
```

[spustit znovu]

100% z Lekce 7

Code Solution

Osnova

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

```
 1   size = 5
 2   sym = ['#',' ']
 3   desk = []
 4
 5   for row in range(size):
 6       line = []
 7
 8       for cell in range(size):
 9           i = (row+cell) % len(sym)
10           line.append(sym[i])
11
12       desk.append(''.join(line))
13
14   print('\n'.join(desk))
```

Convert Time

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **REVIEW EXERCISES** / **CONVERT TIME**

Our task is to create a program that will take time input in 24 hours format and will convert it into english time format including PM and AM .

Example of running the program:

```
~/PythonBeginner/Lesson6 $ python convert_time.py
```

100% z Lekce 7

Python Online Editor

Osnova

1 |

spustit znovu

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

In the solution below, we use ternary operator to decide, whether the hours part of the time

100% z Lekce 7

Al─   sm ʺ trick to determine, whether to use **PM** or **AM**. If the condition
i   Osnova   `True` , then we want to return the item at index 1 (True is equal to 1). If
the condition returns False, we are getting in other words number 0 and so **AM** will be
retrieved from the list.

Although our version is quite slick, it can be better to write the code more explicitly. So the
alternative to our trick might be: `daytime = ['AM','PM'][True if int(hours)>=12`
`else False]`

Neither version is not necessarily better or worse. It depends on the context.

```
time = input('Please enter your time: ')

hours, mins = time.split(':')

adjusted_h = hours if int(hours)==12 else str(int(hours) % 12)

daytime = ['AM','PM'][int(hours)>=12]

print('Converted to English: ' + adjusted_h + ':' + mins + ' ' +
daytime)
```

How many seconds

Create a program that will ask the user for a number representing the number of seconds that
have passed **since midnight**. The program should then print to the terminal time in format
hours:minutes (how many full hours and full minutes are passed since midnight)

- the number of hours can be represented by the numbers between 0 and 12

- the number of minutes can be represented by the numbers between 0 and 59

- the `AM` and `PM` flag should signal, whether it is afternoon (PM) or morning (AM)

100% z Lekce 7

For example, if input is 3900, then 3900 seconds have passed since midnight So the program should print AM.

```
~/PythonBeginner/Lesson6 $ python seconds_to_time.py
Please enter the no. of seconds since midnight: 3900
It's: 1:05 AM
```

Osnova

Online Python Editor

```
1  |
```

[spustit kód]

Code Solution

100% z Lekce 7

We use the built-in function `divmod` to perform floor division between the value stored in the variable `seconds` and 3600 (number of seconds in one hour). Function `divmod` returns two values:

1. Result of the floor division btw. the two numbers and

2. The remainder of the division

Then we adjust the remainder to minutes again using floor division (to get the whole number of minutes).

The rest of the tricks are the same as in the Convert time solution.

```python
 1  seconds = int(input('Please enter the no. of seconds since midnight: '))
 2
 3  hours, remain_seconds = divmod(seconds,3600)
 4  adjusted_h = hours if hours==12 else hours% 12
 5  minutes = remain_seconds // 60
 6
 7  if len(str(minutes)) == 1:
 8      minutes = '0' + str(minutes)
 9  else:
10      minutes = str(minutes)
11
12  daytime = ['AM','PM'][hours>=12]
13
14  print("It's: " + str(adjusted_h) + ':' + minutes + " " + daytime)
```

100% z Lekce 7

Osnova

ASSIGNMENT STATEMENTS

Classic Assignment

The most basic form of assignment in Python looks as follows: `variable = expression`

When Python encounters these statements it performs the following operations:

1. evaluates expression to a value

2. create a reference between the value and the variable

Examples of such classic assignment expressions:

```
1  age = year_now - birth_year
2  value = (5 or 4) and (0 or 6)
3  result = 'Thank you' if user_input else 'Please answer'
```

We have been working with this kind of assignment statement in every Python script so far.

100% z Lekce 7

Y　　Osnova　Ɏ　ssignment

Collection assignment is kind of a statement, when we have a **collection on the right side** of the equal sign and **multiple variables on the left side**. In general we can express the collection assignment syntax as:

```
var1, var2, var3 ... = collection
```

The number of variables on the left side **has to be equal** to the number of items in the collection on the right side.

If Python encounters collection assignment statement it:

1. Checks, whether the number of variables on one side is equal to the number of items in the collection on the other side,

2. iterates over the variables and creates a reference between the variable and the item at corresponding position on the other side (term position is questionable for dictionaries and sets)

Example:

```
>>> a,b,c = [1,2,3]
>>> a
1
>>> b
2
>>> c
3
```

Collection assignment is very helpful trick which can save us many lines of classic assignment statements. If we wanted to write the above example **using classic assignment** it would look like this:

```
>>> a = 1
>>> b = 2
```
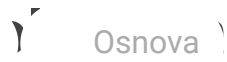
100% z Lekce 7

signment - Examples

We can perform collection with all sequence data types, dictionary and sets.

Strings

```
>>> letter1, letter2, letter3, letter4 = 'John'
>>> letter1
'J'
>>> letter2
'o'
>>> letter3
'h'
```

Tuple

```
>>> name, surname = ('John', 'Smith')
```

We even do not have to put the items inside the parentheses ~~()~~ :

```
>>> name, surname = 'John', 'Smith'
```

And variables now refer to …:

```
>>> name
'John'
>>> surname
'Smith'
```

List

List assignment is in principle equivalent to tuple assignment with the difference that we have to include square brackets **[ ]** :

100% z Lekce 7

```
 |  `>
    `>        Osnova
    `>c
>>> surname
'Smith'
```

Range

```
>>> a, b = range(2)
>>> a
0
>>> b
1
```

Slice returns another sequence

```
>>> lst = list(range(10))
>>> lst
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> penultimate, last = lst[-2:]
>>> penultimate
8
>>> last
9
```

Dictionary Keys

Collection assignment performed on dictionaries **by default** distributes dictionary keys among the variables on the left side. We should not rely on keys being ordered in any way.

```
>>> d = {'name': 'Bob', 'surname': 'Francis'}
>>> category1, category2 = d
>>> category1
'name'
>>> category2
'surname'
```

100% z Lekce 7

'...e... .... dictionary values, we need to use the `dict.values()` method:

Osnova

```
>>> name, surname = d.values()
>>> name
'Bob'
>>> surname
'Francis'
```

Dictionary Items

If we wanted to store tuples of key - value pairs inside the variables on the left side, we need to use the `dict.items()` method:

```
>>> item1, item2 = d.items()
>>> item1
('name', 'Bob')
>>> item2
('surname', 'Francis')
```

Set

We should not rely that the values will be distributed in the order we have added them to the set. Otherwise the syntax is the same as with sequences or dictionary keys:

```
>>> names = {'Hans', 'Helga', 'Hilda'}
>>> n1, n2, n3 = names
>>> n1
'Hans'
>>> n2
'Helga'
>>> n3
'Hilda'
```

100% z Lekce 7

e   Osnova   collection assignment, the numbers of variables and collection items **have to match on both sides**. Otherwise we will get the `ValueError` :

1. More values than variables

```
>>> a,b = range(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

```
>>> name1, name2 = ['Hans', 'Helga', 'Hilda']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

2. More variables than values

```
>>> a,b,c = 10, 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
```

Collection Assignment - Using Parentheses

If more objects are to be assigned than there are variables available, we can pack those redundant objects into parentheses thus creating a new tuple (one single object) from them.

Example:

```
>>> name1, rest = 'Hans', ('Helmut', 'Gerhard')
>>> rest
('Helmut', 'Gerhard')
```

100% z Lekce 7

We can do the same trick with variables. In the example below we:

Osnova

- distribute two strings among the tuple `(a,b)` (string `'Mo'`) and variable `c` (string `'re'`)

- then we further divide the string `'Mo'` among the variables in the tuple `(a,b)` (`'M'` goes to `a` and `'o'` to `b`)

```
>>> ((a,b),c,) = ('Mo','re')
>>> a
'M'
>>> b
'o'
>>> c
're'
```

Collection Unpacking Assignment

Unpacking operation uses star operator `*` in front of variable name into which we want to collect (unpack) multiple items from a collection on the right side of the assignment statement:

```
>>> names = {'Hans', 'Helga', 'Hilda'}
>>> first, *rest = names
>>> first
'Hans'
>>> rest
['Helga', 'Hilda']
```

This collection unpacking works for all other collection types we already know:

```
>>> first, *rest = range(10)
>>> first
0
```

100% z Lekce 7

The unpacked collection is **gathered into a list**, which is then assigned to the variable with the `*`
operator.

Osnova

Dictionaries by default unpack their keys. If we wanted to unpack values or both - keys and
values - we have to use `dict.values()` and `dict.items()` methods respectively:

```
>>> d = {'name': 'Richard', 'surname': 'Wagner', 'age': 69}
>>> first, *rest = d
>>> first
'name'
>>> rest
['surname', 'age']
```

Variables with star operator in front of them collect the values that were not assigned to other
variables around. That means we do not have to place them to the last place among variables:

```
>>> first,*mid, last = range(10)
>>> first
0
>>> last
9
>>> mid
[1, 2, 3, 4, 5, 6, 7, 8]
```

Incorrect Collection Unpacking Assignment

Now we will go over 3 types of errors:

1. Redundant variables

2. Two starred assignements

3. No comma

100% z Lekce 7

What happens if the variable with unpacking operator is redundant meaning there are more variables than objects to be assigned? The variable that contains the unpacking operator is assigned an empty list.

```
>>> a,*b,c,d = 1,2,3
>>> a,b,c,d
(1, [], 2, 3)
```

However, if the unpacking variable is not the only redundant one, we get an error:

```
>>> a, *b, c, d, e= range(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected at least 4, got 3)
```

Two starred assignements

It is not possible to use two starred variables in assignment.

```
>>> a,*b,*c = 1,2,3,4,5,6
>>> a,b,c
File "<stdin>", line 1
SyntaxError: two starred expressions in assignment
```

No comma

Starred variable has to be positioned in a list or tuple - **note** the comma following the variable *a :

```
>>> *a,='abcd'
>>> a
['a', 'b', 'c', 'd']
```

If we do not include the comma (meaning we do not put it inside a tuple containing one variable), we get an error:

```
>>> *a = 'abcd'
```

100% z Lekce 7

❗   Osnova ❗   **Assignment**

Augmented assignments are shorthand for applying arithmetic operations on the same variable as the variable that is being assigned the result of the expression. For example, if we wanted to increase the value of `counter` in a while loop:

Classical assignment	Augmented assignment
counter = counter + 1	counter += 1

Below we list the augmented assignment equivalents to arithmetic operations:

Name	Classical	Augmented
addition	a = a + b	a += b
subtraction	a = a - b	a -= b
multiplication	a = a * b	a *= b
true division	a = a / b	a /= b
floor division	a = a // b	a //= b
modulo	a = a % b	a %= b
power	a = a ** b	a **= b

Augmented assignment syntax saves space and allows us to avoid tedious repetition of the same variable name in the expression. The essence of the assignment statement is that we want to apply an operation (in this case arithmetic) to an object stored in a variable and then store the **result back to the same variable**.

As suggested above, augmented assignment is used often in **while loops:**

```
>>> my_string = 'Hello'
>>> i = 0
>>> while i < len(my_string):
>>>     print('Index ' + str(i) + ': ' + my_string[i])
```

100% z Lekce 7

```
       End
Pr
ra         Osnova

Index 3: l
Index 4: o
```

FUNCTIONS

Introduction

Do you remember the introduction to our course? We introduced a input-output model. Functions are great representatives of IO - they accept inputs, process them inside and return outpus.

100% z Lekce 7

Osnova

looking at a function, when a (variable) name is immediately succeeded by parentheses like this - for example a function that sums the numeric values looks like this: `sum()`

So What is Function?

Function is a tool, that allows programmers to **group code** statements that together **serve a common purpose** - to perform intended action. This means that functions are dedicated to perform or fulfill specific tasks (e.g. convert all letters in a word to upper case).

Bad function would be one, that for example:

1. retrieves text from a file

2. formats the text

3. prints the text to the screen

Good functions have a single purpose. So a good function should take care of only one of the above actions.

Function Inputs

The numbers we want to sum, have to be entered into those parentheses. Each function has its specific requirements, what form should the inputs have and how many inputs there should be. In case of `sum()` we need to pass the input as a tuple of numeric values.

```
>>> sum((1,2,4))
```

Functions DO things

Function `sum()` adds together and returns the result of the calculation as the output.

```
>>> sum((1,2,4))
7
```
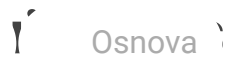
100% z Lekce 7

ʃ Osnova ` fined Functions

Some functions already come with Python inside - they are called **built-in functions**. Python standard documentation [lists all the built-in functions](). We already know some of them - `all()` , `any()` , `sum()` , `sorted()` , `reversed()` , etc.

Of course, the major part of our programming efforts will be to **define** our own functions. The way how the function is defined will tell us everything about what it is supposed to do, what and how many inputs it expects in parentheses and lastly, what output does it return.

Define our own Function

Function is created (**defined**) using `def` statement, which general syntax we have below

```
1  def function_name(param1, param2,...):
2      indented code block
3      indented code block
4      indented code block
5      optional return statement
```

And an actual example could be a function, that sums two numbers

```
1  def add_two(a,b):
2      return a + b
```

Each part of the function definition syntax is important and will be discussed in more detail across the following sections, but here are its components:

- `def` reserved keyword that tells Python to create a new function object

- `function_name` - variable name to which the created object will be assigned

100% z Lekce 7

- ` ` - function inputs - always listed inside parentheses

- `:` colon - tells us, we are creating compound statement (header + body)

- indentation - all the code that belongs to the function, should be indented to the right

- `return` - reserved keyword - makes Python to send the calculated value back to the requester of the value

Function does not exist until the program execution reaches the `def` statement in the code. `def` statement then creates a function object (also functions are objects - of type function) and assigns it to the function name.

When our program reaches the definition of function `add_two` it creates a variable `add_two` and assigns the function object to it. We can see how the function object representation looks like if we write the function name to the terminal

```
>>> add_two
<function add_two at 0x7f655a7b0ae8>
```

Function Name

Function names should reflect the operation of purpose of the function. Function calculating square root of a number can be given name squared. Function name is just another variable name, and therefore the variable name rules apply here as well.

Function Inputs

Some functions require inputs, which they modify later inside the function body. These inputs have to be placed inside the parentheses.

Function definition

In order our functions can require arguments, we have to **define functions** that way. The function

100% z Lekce 7

```
1        nt  ar(char, num_repetitions):
   Osnova
2            n  ar * num_repetitions
```

Function call

We can **call the function** `repeat_char` in our code providing it with inputs it expects. Function inputs are called **arguments** when calling the function:

```
>>> repeat_char('=',20)
'===================='
```

Why do we distinguish among function parameters and arguments?

When we call a function, arguments hand over their values to parameters. Parameters are actually variable names that refer to the values we have provided to the function.
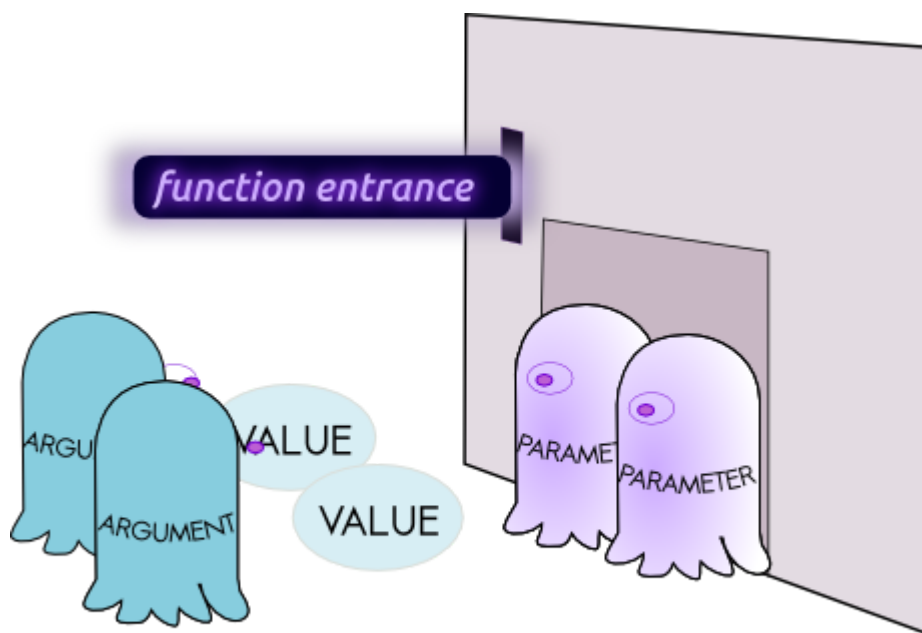
Function Output

100% z Lekce 7

Functions are usually designed in a way that they take some inputs in and return some outputs. To do so, we use a function to **return a result using** `return` keyword. When Python encounters `return` statement the function is **immediately exited** returning the value that follows the `return` keyword:

```
1   def max(sequence):
2       max_item = sequence[0]
3
4       for item in sequence[1:]:
5           if max_item < item:
6               max_item = item
7
8       return max_item
```

The values are returned back to the place, from where the function has been called. Therefore the **result of the function call can be stored inside a variable**:

```
>>> max_num = max([32,2,4,1,4,54,23,12])
>>> max_num
54
```

Multiple Return Statements

There can be multiple `return` statements in one function. Again, once any of the `return` statements is encountered, the value it precedes is returned back to the caller and the function execution is terminated. Therefore, if the statement `if not word.istitle():` in the below function evaluates `True`, then `return False` is executed and `print('Bye')` is never executed.

```
1   def all_title(words):
2
3       for word in words:
4
5           if not word.istitle():
6               return False
7
```

100% z Lekce 7

```
>>        ⌐ ⌐ob','Frank','mike', 'John'])
      Osnova
al.
>>> all_title(['Bob','Frank','Mike', 'John'])
Bye
True
```

No Return Statement

Functions do not have to be always designed to return a specific value. For example the below function just prints a chessboard to the terminal:

```python
 1  def show_board(size):
 2      index = 0
 3
 4      for row in range(size):
 5          print()
 6
 7          for col in range(size):
 8              index += 1
 9              char = '#' if index % 2 == 1 else ' '
10              print(char, end='')
11      print()
```

Notice that in the inner for loop, we use second parametr in the `print()` function - `end=''` . The parameter is by default set to `'\n'` . This tells Python to print the value/s inside the function and jump on the new line `'\n'` .

In the code above, we've changed the newline `'\n'` to an empty string `''` . This tells Python to **stay on the same line** - do nothing after printing the value/s.

In consequence, this mean that:

- we print a row of characters `'#'` in the `range(size)` - **the inner loop**.

- After we finish 1st row, we move to the next row - **the outer loop** `for row in range(size)` .

- We use `print()` and jump on the next line

```
  `>        `d´ `
       Osnova
   +
  # #
 # # #
  # #
 # # #
```

If we do not write `return` statement into the function, function returns the value `None` when the function execution arrives to the end of the function body. We can prove this by assigning the result of `show_board()` function to a variable and then print the variable's content.

```
>>> result = show_board(5)
# # #
 # #
# # #
 # #
# # #
>>> print(result)
None
```

Other examples of function which return None are `print()` function or `list.sort()` method and many more. If we try to assign the value returned by these two functions to a variable, we find out that the variable contains value `None` .

Create Function - Exercise

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **FUNCTIONS** / **CREATE FUNCTION - EXERCISE**

Our task is to create a function that will:

1. Take inputs

   - starting point of the integer range

   - ending point of the integer range

100% z Lekce 7

3. ...ut... ...eturn the sum of all numbers in range between start and stop

Osnova

Online Python Editor

```
1  |
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution ▲

100% z Lekce 7

```
      m_ r  ge(from_num,to_num):
Osnova      L    = 0
3       for number in range(from_num, to_num):
4           result += number
5       return result
```

If we call the function after we have defined it, this should be the result:

```
>>> result = sum_range(1,5)
>>> result
10
```

Executing a Function

In order code inside the function definition can be executed, we have to **call the function**. Function is called by stating its name and appending parenthesis immediately after its name. Parentheses can, but do not have to, contain inputs. In the example below, the built-in function `all()` requires one input object in form of a collection:

```
>>> result = all([True, False, True, True])
False
```

This means, that

1. function has to be first defined

2. after the function has been defined, we can repeatedly call it from any place in the code

3. the result that the function produces can be stored in a variable

If we try to call a function that has not been defined yet, we can the `NameError` :

```
>>> no_func()
```

100% z Lekce 7

```
                    'no_func' is not defined
        Osnova
```

We can call the function `all()` without problems, because it is predefined (built-in) inside Python.

Function name without parentheses

If we write the function name without parentheses, we are in fact retrieving the actual function object.

```
>>> all
<built-in function all>
```

Therefore, if we want **to "activate"** or invoke the function, we **need to append parentheses** behind the function name and fill them with expected inputs:

```
>>> sum((1,2,3))
6
```

Function is an Object

In Python, function is also an object. **That means** that the below function object created by the following `def` statement:

```
1  def add_two(a,b):
2      return a + b
```

- can be retrieved from the memory

The sequence of numbers and letters following the `function add_two at` is the actual address in memory, where the function object lives. Function name points to that address and so when we write the function name, Python retrieves the object from that address.

100% z Lekce 7

n' **other variable**

Osnova

```
>>> new_add = add_two
```

The line above is the same in principle to the variable new_add referencing to whatever variable add_two references. The new_add function now works the same as add_two - because the both variable names reference the same function object:

```
>>> result = new_add(5,7)
>>> result
12
```

- can be passed as input into another function

```python
1  def greet():
2      print('Hello')
3
4  def wrapper(func):
5      return func()
```

The wrapper function will call the function object and return its result:

```
>>> wrapper(greet)
Hello
```

Functions vs. Methods

We will often see two types of functions but both will have parentheses appended.

Functions

Some functions will **stand alone**, just as the built-in sum() function

100% z Lekce 7

Osnova

Methods

But some functions will be **chained to another object** via a dot operator `.` :

```
>>> 'abcd'.upper()
'ABCD'
```

This latter type of functions is called methods and they perform their operations on objects, they are connected to via the dot operator.

For example, the `.upper()` method does not have anything entered between the parentheses. This is because it just performs operation on the string, it is attached to. The only input to upper is this string.

Additional Comments

Functions make program better understandable and readable for program readers by:

1. allowing for code reuse and thus saving space

2. providing means of abstraction

Code Reuse

Function allows statements to be run repeatedly during the program course without actually copying and pasting the same code across the code. Functions are kind of processing boxes, that take data in, run the code that possibly processes or otherwise uses the inputs and can also return the result of the computing.

Abstraction

100% z Lekce 7

only on one task within a program and therefore should not be too long.

Osnova readability and code maintenance in the future. Functions should not perform more than 1 task. They should be designed in a way, they represent only 1 task. It also helps when designing the program. We just need to think, what actions we want it to perform and then just package those individual features into functions.

Example

We want to create a program that will calculate a person's body mass index (BMI). We would like to be able to convert weight from pounds to kilograms and height from inches to meters.

Procedure to calculate body mass index could be divided into following steps:

1. Convert height

2. Convert weight

3. Calculate BMI (Divide the weight in kilograms by your height squared)

That means we now need not only functionality for calculating BMI, but also for conversion from pounds and inches.

```python
def to_kilograms(weight):
    return weight * 0.45

def to_meters(height):
    return height * 0.025

def bmi(weight, height):
    return to_kilograms(weight) / to_meters(height) **2
```

```
>>> my_bmi = bmi(125, 63)
```

100% z Lekce 7

Osnova

SPECIAL STRING CHARACTERS

Overview

Special characters have special meaning for Python. They serve purposes as line termination, returning to the beginning of a line, represent tabulator etc. Here are listed some useful special characters and the way they are written:

- `\n` **ASCII linefeed** (LF)

- `\r` **Carriage Return** (CR)

- `\t` **Horizontal Tab** (TAB)

- `\b` **ASCII backspace** (BS)

- `\f` **ASCII formfeed** (FF)

Escaping

100% z Lekce 7

Before ... 'oc' ...t examples using special string characters, we should speak about ..., ... Osnova ... .es them. You may have noticed that all the special characters in the previous section included a backslash `'\'` character. Backslash is used to signal that the following character should be interpreted in a special way.

However, how can we tell Python that we actually want backslash to be printed to the terminal? In that case we have to prepend another backslash in front of it:

```
>>> print('\\')
\
```

Printing a single backslash would return SyntaxError:

```
>>> print('\')
  File "<stdin>", line 1
    print('\')
             ^
SyntaxError: EOL while scanning string literal
```

Putting backslash in front of the another backslash or other character that has special meaning for Python, is called **escaping**. **Escaping** assures that the actual character following backslash will be printed out.

Escaping quotes

If we wanted to include a single quote inside a string enclosed in single quotes, we need to prepend backslash in front of this single quote:

```
>>> print('It\'s time for breakfast')
It's time for breakfast
```

If we don't escape the single quote, we will get a SyntaxError

```
>>> print('It's time for breakfast')
  File "<stdin>", line 1
    print('It's time for breakfast')
              ^
```

100% z Lekce 7

Another option is to enclose the string in double quotes. Then we do not have to escape single quote in the string and the code looks nicer:

```
>>> print("It's time for breakfast")
It's time for breakfast
```

All the above is valid for including double quote inside a string, too. If we want to do that, we should escape it or enclose the string in single quotes.

Linefeed / Newline

Newline character makes Python to **advance downward to the next line**. It is commonly escaped as `"\n"` , abbreviated LF or NL. In Windows, newline character is used together with carriage return `'\r'` character to end the line and move the cursor to the beginning of the next line `"\r\n"` .

```
>>> print('This is the first line\nThis is the second line')
This is the first line
This is the second line
```

Historical Perspective (for those interested in history)

CR = Carriage Return and LF = Line Feed, two expressions have their roots in the old typewriters and terminals. LF moved the paper up (but kept the horizontal position identical) and CR brought back the "carriage" so that the next character typed would be at the leftmost position on the paper (but on the same line). CR+LF was doing both, i.e. preparing to type a new line. As time went by the physical semantics of the codes were not applicable, and as memory and floppy disk space were at a premium, some OS designers decided to only use one of the characters, they just didn't communicate very well with one another.

Most modern text editors and text-oriented applications offer options/settings etc. that allow the automatic detection of the file's end-of-line convention and to display it accordingly.

100% z Lekce 7

Osnova **urn**

Carriage return character makes Python to return to the beginning of the current line without advancing downward. The name comes from a printer's carriage, as monitors were rare when the name was coined. This is commonly escaped as `"\r"`, abbreviated CR:

```
>>> print('This is the first line\rThis is the second line')
This is the second line
```

In the above example string `'This is the first line'` is printed first, however, carriage return character makes the cursor to return back to the beginning of the same line. Then the string `'This is the second line'` is printed over the original first line string. Therefore we cannot see the first sentence.

Tabulator

Makes computer to fill in multiple white spaces (normally 4) in the place where it is found. It is commonly escaped as `'\t'` and abbreviated as TAB.

```
>>> print('This is the first line\tThis is the second line')
This is the first line    This is the second line
```

Raw String

We now know, that if we print one of the characters `'\n'`, `'\r'`, `'\t'`, we get to beginning of the line or move cursor multiple spaces to the right. But what if we wanted to print actual

100% z Lekce 7

We would l... ... ape these special characters using backslash, what does not look very
... Osnova
...

```
>>> print('\\n','\\r', '\\t')
```

The solution to prettyfying our strings is **raw string**. Raw string is created by prepending letter r
in front of the string we want to be printed:

```
>>> print(r'\n, \r, \t')
\n, \r, \t
```

Special characters are not interpreted as special, but are printed as they are written.

```
>>> print(r'First line \n Second Line \t tabbed part')
First line \n Second Line \t tabbed part
```

Backspace

PYTHON ACADEMY / 6. INTRODUCTION TO FUNCTIONS / SPECIAL STRING CHARACTERS / BACKSPACE

Erases the preceding character. It is commonly escaped as `"\b"` . The below example erases
letter `e` from the word `line` :

```
>>> print('This is the first line\bThis is the second line')
This is the first linThis is the second line
```

Formfeed

PYTHON ACADEMY / 6. INTRODUCTION TO FUNCTIONS / SPECIAL STRING CHARACTERS / FORMFEED

Makes computer to advance downward to the next "page". It was commonly used as a page
separator, but now it is also used as a section separator. Text editors can use this character

100% z Lekce 7

```
>>          hj  's the first line\fThis is the second line')
      Osnova
.l.         .i.   line

                    This is the second line
```

PYTHON LIBRARY

Python Modules

The same way we can package code into a function and then reuse it later, we can package function definitions and other statements inside a bigger unit - Python file. Python files (with the suffix **.py**) are also called **modules**.

100% z Lekce 7

These function definitions create functions that can be accessed from other Python scripts thus giving us functionality reservoir.

Module advantages

- we don't have to write all the code ourselves

- modules make code more readable

- we can reuse the code

Standard Modules

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **PYTHON LIBRARY** / **STANDARD MODULES**

Python installation comes with a **library of standard modules**. Standard modules contain useful functions that tackle various areas of programming (networking, data types, work with files, concurrency and much more). Modules contained in standard library provide standardized solutions for many problems that occur in everyday programming.

To get an overview of the modules available in Python's standard library, you can check Python's documentation.

We also have so called **3rd party modules** which have to be additionally installed.

Importing

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **PYTHON LIBRARY** / **IMPORTING**

To access this functionality, we have to so called **import the module**. We import modules in Python using `import` keyword followed by the name of the file, which functionality we want to load into our program.

100% z Lekce 7

Osnova

We have to place import statements **at the beginning** of our Python scripts.

MODULE RANDOM

Introduction

We use `random` module when we want to:

- generate a random number in a specified range,
- or we want to shuffle items in a sequence (like cards in a deck)

100% z Lekce 7

The random module can be loaded into our scripts as follows:

Osnova

```
import random
```

The **import** statement should be placed at the top of the script. This is not a syntax rule but a good practice that all Python programmers respect. It is important to know for other programmers that will read our code, what functionality it uses.

We will want to invoke various functions that random module provides us. In order to do that, we will have to tell Python in which module it should be looking for those functions. This is done specifying the **module name** followed by **dot** and **function name**.

Example

Calling `randrange()` function from `random` module:

```
>>> import random
>>> random.randrange(1,100)
27
```

In the following sections we will explain how individual functions work.

Generating random number

We have imported our random module at the top of the script as: `import random`

Now we want to **generate a random number between 0.0 and 1.0**. To do that, we can use random module's `random()` function. We can invoke the function by specifying the module name, dot and the function name: `random.random()`

Code Task

1. What will be the result when we use `random.random()`?

ι

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution ▲

Using `.random()`

100% z Lekce 7

```
ρ ⎺⎺⎽⎼⎺⎺⎻⎽55⎺⎺⎺332
    Osnova  ⎻    ⎵m()
0.6450641021682656
```

Random number between 0.0 - 1.0

```
>>> random.random()*100
93.50701428518701
>>> random.random()*100
52.36821688144858
>>> random.random()*100
43.492309902093716
```

Shuffling items in a sequence

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **MODULE RANDOM** / **SHUFFLING ITEMS IN A SEQUENCE**

In case we would like to randomly shuffle our lists we can use `random.shuffle(list)` function.

Example:

```
>>> lst = [45, 21, 53, 1, 213, 43, 42, 85]
>>> random.shuffle(lst)
>>> lst
[42, 21, 53, 213, 85, 43, 1, 45]
>>> random.shuffle(lst)
>>> lst
[53, 42, 21, 85, 213, 1, 45, 43]
```

In the example above, the list `lst` has been repeatedly reordered in random manner. This kind of functionality is useful when we wanted to implement a card game for example. The list `lst` could represent a card deck. **The input sequence has to be of mutable data type, therefore we used a list**

100% z Lekce 7

Osnova **domly from a sequence**

We have two options:

1. Choose random item

2. Choose random number(s) from a range

Choose random item

Maybe we wanted to create a game, where we will need to randomly choose items from a collection. **For example** a game, where we wanted a player to guess a name. In such a game we would normally choose a **random word from a collection of words**.

To perform a random choice from a collection we can use: `random.choice(sequence)`

```
>>> words = ['hello', 'new', 'write', 'car', 'notebook']
>>> word = random.choice(words)
>>> guess = input('Guess the word: ')
Guess the word: hello
>>> result = 'guessed' if guess == word else 'not guessed'
'not guessed'
```

The `random.choice()` returns a random element from the non-empty sequence seq. If seq is empty, raises `IndexError`

Choose random number(s) from a range

Another situation when we would like to use random module is to choose a random number from a range. This is useful for **generating employee id numbers** for example. To do that we can use `randrange()` or `randint()` function:

1. `random.randrange(start,stop,step)` - returns a number **n** from a sequence between **start** and **stop**, excluding **stop**. Some numbers in input sequence can be skipped by the amount of **step** (for example every other number if step=2).

100% z Lekce 7

2.   ...nd' *(start,stop) - returns a number **n** that is greater or equal to **start** and
        Osnova   .gr   ɔ **stop**

The difference among `randint()` and `randrange()` is that `randrange()` will never return the **stop** number (what is similar to `range()` data type) as a result. Also, `randint()` does not allow for skipping numbers.

In the example below we check, whether an id is already present in our database and if not, we add it:

```
>>> ids = ['X5235', 'X6752']
>>> id = random.randint(1000, 9999)
>>> if 'X' + str(id) not in ids: ids.append('X' + str(id))
```

Lastly we may want to choose randomly a collection of items from a bigger collection. In that case, we want to create a **sample**. Example would be distribution of **n** cards from a deck at the beginning of a game.

To shuffle an immutable sequence and return a new shuffled list, use sample(x, k=len(x)) instead:

3. `random.sample(population, k)`

   ○ returns a `k` length list of unique elements chosen from the `population` sequence or set.

   ○ returns a new list containing elements from the population while leaving the original population unchanged.

Osnova

QUIZ

Assignments

1/15

What will be stored in variable `c` after running the following expression?: `a, b, c = 1, 2, 3`

A. 3

B. 2

C. This is not a permitted expression - SyntaxError

D. 1,2,3

100% z Lekce 7

ᵛ Osnova

1/15

What will Python 3 do, if we enter the following statement into the terminal?:

```
>>> print
```

A. print newline

B. raise SyntaxError

C. print empty string

D. return function object

Modules Intro

1/5

How to load module `random` into our script?

100% z Lekce 7

A. include random

Osnova

B. import random

C. load random

D. use random

HOME EXERCISES

Osnova

Your task is to create 2 functions:

Function that should copy the [built-in function min()](). It should take as input any sequence data type and return item with the lowest value.

Example of using min:

```
>>> min([43,45,87,21,23])
21
```

Function that should copy the [built-in function max()](). It should take as input any sequence data type and return item with the highest value.

Example of using max:

```
>>> max([43,45,87,21,23])
87
```

Online Python Editor

```
1   # Min
2
3   # Max
```

100% z Lekce 7

Osnova

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

- The first item of the sequence should be considered the minimum or maximum before we begin to loop over the rest of the sequence

- Once we encounter greater or smaller item, we store it instead of the value we store currently in our result variable

```python
1  def my_min(sequence):
2      min = sequence[0]
3      for item in sequence[1:]:
4          if item < min:
5              min = item
6      return min
```

```python
1  def my_max(sequence):
2      max = sequence[0]
3      for item in sequence[1:]:
4          if item > max:
5              max = item
6      return max
```

100% z Lekce 7

Osnova

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **HOME EXERCISES** / **FIND**

Your task is to create a copy of a built-in string method .find() that will accept:

1. any sequence we want to search through

2. and object, we want to find in the sequence

The value returned from this function should be:

- **The index** at which the first occurrence of the searched object has been found

- **or the value -1** if the object has not been found

Example of using the function:

```
>>> find(['pear', 'apple', (23, 45), 653, {'name': 'John'}] , {'name': 'John'})
4
```

Online Python Editor

```
1  |
```

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

- if we want to use for loop, it will be convenient to use built-in function `enumerate()` that returns both the index and the actual item found at the index

- when we have the actual item and its index, it is enough just to compare the extracted item with the searched item. If they match, we return the index, where the object has been found

- if the whole sequence has been iterated and no matching item has been found, we return value -1

```python
1  def my_find(seq, item):
2      for i, obj in enumerate(seq):
3          if obj == item:
4              return i
5      return -1
```

All & Any

100% z Lekce 7

functions:

Osnova

Function that should imitate the built-in function all(). It should take as input any sequence data type and return `True`, if all items in the sequence have boolean value `True`, or the sequence is empty. Otherwise it should be return `False`.

Example of using `all()`:

```
>>> all([43,45,87,21,23])
True
>>> all([])
True
>>> all([0,12,431,3])
False
```

Function that should imitate the built-in function any(). It should take as input any sequence data type and return `True`, if at least one item in the sequence have boolean value `True`. Otherwise it should be return `False` - also if the sequence is empty.

Example of using `any()`:

```
>>> any([43,45,87,21,23])
True
>>> any([])
False
>>> any([0,12,431,3])
True
>>> any(['',[],(),0])
False
```

Online Python Editor

```
1  |
```

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution	▲

- We will use similar principle of searching for a specific value. If we find that value in the sequence, we can return from the function

- In case of function that mimics `all()` , we have to find at least one object, whose boolean value is `False` , therefore we perform the test `if not item:` which is equivalent to `if not bool(item):`

- In case of the function imitating `any()` , we need to find at least one item, whose boolean value is `True` , therefore we perform the test `if item:`

- If the above mentioned conditions never evaluate to `True` , then the function will not return until the foor loop terminates

100% z Lekce 7

```
            for item in sequence:
                not item:
4                   return False
5           return True
```

```
1   def my_any(sequence):
2       for item in sequence:
3           if item:
4               return True
5       return False
```

Reversed

Your task is to create a function that will imitate the built-in function reversed(). It will take any sequence as an input and will return a list of items from the original sequence in reversed order.

Example of using the function:

```
>>> reversed(range(10))
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> reversed(['New', 'Years', 'Eve'])
['Eve', 'Years', 'New']
>>> reversed('Hello World')
['d', 'l', 'r', 'o', 'W', ' ', 'o', 'l', 'l', 'e', 'H']
```

Online Python Editor

```
1  |
```

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

- we can use slicing to reverse the order of any sequence and then convert it into a list

- we can use for loop, where each iteration will insert the current item at the beginning of newly created sequence

```
1  def my_reversed(sequence):
2      return list(sequence[::-1])
```

```
1  def my_reversed(sequence):
2      result = []
```

100% z Lekce 7

Osnova

GCD

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **HOME EXERCISES** / **GCD**

GCD is an acronym for [Greatest Common Divisor](). Your task is to create a function, that will calculate greatest common divisor of two integers.

How to calculate GCD?:

We can use modulo operator to repeatedly calculate the remainder of division among number A and B. If the remainder != 0, then A becomes B and B becomes the remainder in the next iteration. This is done until the remainder of division among A and B is 0.

```
|  A  |  B  |  Remainder  | Division Result|
---------------------------------------------
| 12  |  8  |      4      |       1        |
|  8  |  4  |      0      |       2        |
```

The GCD for numbers 12 and 8 is 4, because after the division by 4, there is 0 remainder.

Example of using `gcd()` function:

```
>>> math.gcd(414,78)
6
>>> math.gcd(414,49)
1
```

Online Python Editor

```
1 |
```

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

- Different pairs of integers will differ in the number of iterations needed to arrive to the greatest common divisor. Therefore we will use the `while` loop

- The other important heuristic we need to apply is that when we want to stop the iteration

- Two numbers are always divisible by 1 without remainder, therefore we should use this condition as a limit.

- On each iteration, we dividie `a` by `b` , that means we have to check for the condition,

100% z Lekce 7

is ' is not equal to 1 and remainder is 0, then we can return the currently
Osnova   variable b

```
1   def gcd(a,b):
2       while b > 1:
3           remainder = a % b
4           if not remainder:
5               break
6           a,b = b,remainder
7       return b
```

Generate Data

Create a function that will be able to generate a table that should have the following headers:
`'Name','Item','Amount','Unit_Price', 'Total_Price'`

The function should take only **one input** - number of rows, the generated table should have. The data should be generated from the following variables:

- values from `customers` should randomly populate the column `Name`

```
1   customers = ['Bettison, Elnora',
2                'Doro, Jeffrey',
3                'Idalia, Craig',
4                'Conyard, Phil',
5                'Skupinski, Wilbert',
6                'McShee, Glenn',
7                'Pate, Ashley',
8                'Woodison, Annie']
```

- values from `products` should randomly populate the columns `Item` - the first part of the tuple, `Unit_Price` - second part of the tuple.

100% z Lekce 7

```
2                  'Claravis', 9.85), ('Nadolol', 12.35),
     Osnova
3                  'Quinapril', 34.89), ('Doxycycline Hyclate', 23.43),
4                 ('Metolazone', 43.06), ('PAXIL', 14.78)]
```

- values for the column `Amount` should be generated randomly from numbers in range 1 to 100 (including)

- values for the column `Total_Price` should be calculated as the product of `Amount` and `Unit_Price`

The resulting list could look like this:

```
 1  dataset = [ ['Name', 'Item', 'Amount', 'Unit_Price', 'Total_Price'],
 2             ['Bettison, Elnora', 'Doxycycline Hyclate', 98, 23.43,
    2296.14],
 3             ['McShee, Glenn', 'DROXIA', 27, 33.86, 914.22],
 4             ['Conyard, Phil', 'Nadolol', 44, 12.35, 543.4],
 5             ['Bettison, Elnora', 'Claravis', 91, 9.85, 896.35],
 6             ['Idalia, Craig', 'Nadolol', 83, 12.35, 1025.05],
 7             ['Woodison, Annie', 'Metolazone', 46, 43.06, 1980.76],
 8             ['Woodison, Annie', 'DROXIA', 50, 33.86, 1693.0],
 9             ['Skupinski, Wilbert', 'Nadolol', 60, 12.35, 741.0],
10             ...
```

We recommend you do this exercise locally. You can save the code f.e. under the name `generate_data.py`. You will use this module again the home exercise about [Statistics](#) (the advanced version) in the following lesson.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

```
 1   import random
```

100% z Lekce 7

```
             Osnova      ['Bettison, Elnora',
   6                      'Doro, Jeffrey',
   7                      'Idalia, Craig',
   8                      'Conyard, Phil',
   9                      'Skupinski, Wilbert',
  10                      'McShee, Glenn',
  11                      'Pate, Ashley',
  12                      'Woodison, Annie']
  13  products = [('DROXIA', 33.86),('WRINKLESS PLUS',23.55),
  14                      ('Claravis', 9.85), ('Nadolol', 12.35),
  15                      ('Quinapril', 34.89), ('Doxycycline Hyclate', 23.43),
  16                      ('Metolazone', 43.06), ('PAXIL', 14.78)]
  17
  18  def generate_dataset(num_rows):
  19
  20      for i in range(num_rows):
  21
  22          name = random.choice(customers)
  23          item, price= random.choice(products)
  24          amount = random.randint(1,100)
  25          total_price = amount * price
  26
  27          row = [name,item, amount, price, total_price]
  28          dataset.append(row)
  29
  30      return dataset
```

Lorem Ipsum Poetry [H]

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **HOME EXERCISES** / **LOREM IPSUM POETRY [H]**

In some situations we need to generate test text as when for example, we need to populate our website with some dummy text before the real content is available. To do this, it is better to have

100% z Lekce 7

Create a list of words, for example:

Osnova

- articles ("the", "a", "an"),

- determiner ("another", "this", "every", "many")

- subjects ("cat", "dog", "man", "woman"),

- verbs ("sang", "ran", "jumped"), and

- adverbs ("loudly", "quietly", "well", "badly")

The function should take input of how many rows should the text have.

The words should be chosen randomly from the above lists, but the order in which the words are added together should be somehow meaningful - your program should again choose randomly one of the following orders:

- article, subject, verb, adverb

- determiner, subject, verb

- determiner, subject, verb, adverb

Example of running the program using the above functionality (number 5 as input for the number of the rows):

```
~/PythonBeginner/Lesson4 $ python lorem.py
another boy laughed badly
the woman jumped
a boy hoped
a horse jumped
another man laughed rudely
```

Online Python Editor

1 |

100% z Lekce 7

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution   ▲

- first, we have to load the functionality from the `random` module as we will want to randomly choose strings from a collection of sequences

- then we have to specify the categories of word types and actual corresponding `words`

- we also want to tell Python, what word combinations are allowed in our poetry - therefore we create variable `combinations`

- inside the lorem_poetry function we collect generated rows as well as individual words chosen random into lists

- to create sentence from a list of words, we use `' '.join(row)` method, which inserts

100% z Lekce 7

... the ... nd, rows have to be joined on newline character `'\n'` in order to have
Osnova ... on a separate row

```python
import random

words = {'articles' : ("the", "a", "an"),
         'determiners' : ("another", "this", "every", "many"),
         'subjects' : ("cat", "dog", "man", "woman"),
         'verbs' : ("sang", "ran", "jumped"),
         'adverbs' : ("loudly", "quietly", "well", "badly")}

combinations = [["articles", "subjects", "verbs", "adverbs"],
                ["determiners", "subjects", "verbs"],
                ["determiners", "subjects", "verbs", "adverbs"]]


def lorem_poetry(num_rows):
    rows = []
    for _ in range(num_rows):
        combination = random.choice(combinations)
        row = []
        for category in combination:
            row.append(random.choice(words[category]))
        rows.append(' '.join(row))

    result = '\n'.join(rows)
    print(result)
    return result

poetry = lorem_poetry(5)
```

Pascal Triangle [H]

PYTHON ACADEMY / **6. INTRODUCTION TO FUNCTIONS** / **HOME EXERCISES** / **PASCAL TRIANGLE [H]**

100% z Lekce 7

Pascal's triangle is a sequence of numbers where each element of each row is either 1 or the number right above it. The first 4 rows of Pascal's triangle look like this:

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

For example, the next row of the triangle would be:

- 1 (since the first element of each row doesn't have two elements above it)

- 4 (1 + 3)

- 6 (3 + 3)

- 4 (3 + 1)

- 1 (since the last element of each row doesn't have two elements above it)

That means that on sides, there will always be number 1.

Task 1

Create a function called `pascal()` , that takes positive or negative integer `n` as input, generates the first `n` rows of the triangle and returns the result as a list. For example if `n` is 4:

```
[[1],[1, 1],[1, 2, 1],[1, 3, 3, 1]]
```

Note that the returned result has to be a list of lists in order we can simulate rows in the triangle.

If the input number determining the number of triangle's rows is a negative integer, then the triangle should be inverted:

```
[[1, 3, 3, 1],[1, 2, 1],[1, 1],[1]]
```

Task 2

Then create a function `print_pascal()` , that will nicely - symmetrically print the triangle to the

For example, for a triangle with 6 rows:

Osnova

```
1
   1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
1 5 10 10 5 1
```

And for the same inverted triangle:

```
1 5 10 10 5 1
 1 4 6 4 1
  1 3 3 1
   1 2 1
    1 1
    1
```

Online Python Editor

```
1
```

100% z Lekce 7

Osnova

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution ▲

Task 1

Our Pascal triangle solution relies on the fact that :

- on the first and last index, we will have always number 1

- the rest of the items will be calculated as the sum of items at the same index as the calculated item and index - 1

```python
def pascal(num_rows):
    triangle = []

    for row in range(num_rows):
        triangle.append([])

        for colum in range(row+1):

            if colum == 0 or colum == len(triangle[row-1]):
                triangle[row].append(1)
            else:
                previous = triangle[row-1]
                triangle[row].append(previous[colum] +
    previous[colum - 1])
```

100% z Lekce 7

$\top$ Osnova

First version of the function that should print our triangle in a neat way could look like the one below. This `print_pascal()` function however performs multiple tasks. It not only prints the triangle, but it also:

- Determine the widest row

- Determines the widest rows actual width if converted to string

```
1   def print_pascal(triangle):
2
3       width = 0
4       i = 0 if len(triangle[0])> len( triangle[-1]) else -1
5       for item in triangle[i]:
6           width += len(str(item))
7
8       for row in triangle:
9           for c,num in enumerate(row):
10              row[c]=str(num)
11          print(' '.join(row).center(width + len(triangle)-1))
```

Principles of good function design dictate, that a function should perform only a single task. We should distribute some parts of the code above to separate functions. Below, we have extracted the part, that:

1. Determines the widest row to the function `longest_row()`

2. Determines the widest row's actual length to the function `max_row_length()`

The `longest_row()` code has been changed from `i = 0 if len(triangle[0])> len( triangle[-1]) else -1` to multiple lines of code. This is because we do not want to rely on the fact, that the Pascal triangle list will be supplied with rows ordered from widest to narrowest or vice versa.

Now the `print_pascal()` function seems a bit clearer. It retrieves the maximum row width and then just prints the content of the triangle.

```
1   def print_pascal(triangle):
2
3       width = max row length(longest row(triangle))
```

100% z Lekce 7

```
6          for c,num in enumerate(row):
Osnova         row[c]=str(num)
8          print(' '.join(row).center(width + len(triangle)-1))
```

```python
1  def longest_row(sequence):
2
3      longest = (0,sequence[0])
4
5      for i, row in enumerate(sequence[1:]):
6          if len(longest[1]) < len(row):
7              longest = (i+1,row)
8
9      return longest[1]
10
11     # i = 0 if len(triangle[0])> len( triangle[-1]) else -1
```

```python
1  def max_row_length(sequence):
2
3      width = 0
4
5      for item in sequence:
6          width += len(str(item))
7
8      return width
```

DALŠÍ LEKCE