General Information about Python

Historical Development of Python

This programming language was developed in the early 90s by a Dutch programmer called Guido van Rossum. Most people think that this programming language is named after the python snake, assuming its name is Python.

However, contrary to the assumption, the name of this programming language does not come from the python snake. Guido van Rossum named this programming language inspired by the show of **Monty Python’s Flying Circus** from an English comedy group called **The Monty Python**.

| **Rossum** |
| --- |
| *Guido van Rossum* |

Even though this is the case, the **logo** of the Python programming language with a serpent yılan type in many places has become almost a tradition.



This language has a huge group of developers around the world. If you have any problem, you can always ask other Python users/developers for help or find a suitable answer on several sites like [**stackoverflow.com**](https://stackoverflow.com/)

General Information about Python

Review of Tools & Installations

The version of Python 3.X was released in 2008. It makes Python more *readable* and *consistent* than previous versions. Throughout this course, we will use the latest versions of **Python 3.X** (currently Python 3.9.0).

There are several Integrated Development Environments (IDE) that we can write and run Python codes on them. We prefer **Anaconda Navigator (Anaconda3) package** program which includes several IDEs options for you. You can **optionally** install Anaconda on your computer. Currently, Python 3.7 is available in the Anaconda package.

Among the IDEs in Anaconda, we prefer to use **Jupyter Lab** to write down the Python codes on it. The Jupyter Lab is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. **Jupyter Notebook** can also be used which is very similar to Jupyter Lab.

Apart from Anaconda, of course, **Python Shell** (IDLE) can also be used as a basic interpreter tool for Python.

**Throughout this course**;

* We will provide you **Playground** module to write and run your code on each lesson page. Therefore, as we stated before **you don't have to** install Anaconda Package, it is **optional**.
* Additionally, we will be using all Jupyter Notebook based IDEs, including the Google Colab application, during in-class sessions.
* You will have several **Scratch** exercises in accordance with Python codes. You can find an explanatory video below about how to solve the Scratch exercises.

If you decide to install Anaconda, you can visit the installation and user guide [**here**](https://lms.clarusway.com/course/view.php?id=7&section=1).

Here is an alternative way to run your code. This alternative program is **Visual Studio Code (VS Code)** which is also available in the Anaconda Package. Although we prefer to use *Jupyter Lab* in Anaconda Package to run our codes, you can install and use the VS Code program either.

VS Code will be used in the **interactive complementary video lessons** you will receive throughout the course. You can watch the video below about the installation of Python and VS Code if you want.

* **How to Get Started with VS Code**

You can also watch a video below about some configuration issues of VS Code.

* **Configuring of VS Code**

## First Steps into Coding

### Introduction

In this lesson, we will show you how to write and develop a code outline with Python programming language. Although these initial codes are quite simple, they will give you great tips on what Python language is like.

### First Program for 'Being a Good Person'

Let's write our first program. We will print 'being a good person' on the screen:

input :



1

2

print('being a good person')

output :



1

2

being a good person

As you see, it prints what you have written between single quotes like: 'text'. Here is another single line of code:

input :



1

2

print('clarusway will change my life')

output :



1

2

clarusway will change my life

https://drive.google.com/uc?export=view&id=1AAtBbH3F0uSt3rVDf7PhBS7L1VcMQAh-**Scratch Time ! :**Solve this example with [**scratch**](https://scratch.mit.edu/projects/341646035/editor/).

We wrote clarusway will change my life using print() **function**. For now, you do not need to understand how this code runs, just start to enjoy the coding looking like the English language.

If we make some statements at this point, print is the name of a function. A **function** is a piece of code that executes some logic for you, e.g. prints a text or square a number.

In some cases, a function is a subprogram we produce which can be reused within your programs. If the name of a function is followed by parentheses, it means that it was called to get the result.

**💡Tips:**

* Surrounding the expression with triple quotes: **"""..."""** or **'''...'''** ensures that the code returns no error, especially in long texts.

## First Steps into Coding

### Matter of Quotes

As a Python programmer, you will deal with quotes a lot. As you know, a string text is surrounded by a pair of quotes.

There are basically two types of quotes we use. **Single** or **double** quotes. Both are the same but we should use them in the correct way:

* Use double quotes if your string includes the single one:e.g. "It's my pleasure!"
* Use single quotes if your string includes the double one:e.g. 'He said: "I am done" and fell down.'
* Use triple quotes if your string is too long which composed of multiple lines : e.g. :

'''

...long string...

..long string..

'''

Here's an example that shows using double-quotes. Note that, the text (it's not a problem) below includes a single quote **'**.

input :



1

2

print("it's not a problem")

output :



1

2

it's not a problem

Here is a new single-line program too. In this case, be careful with double-quotes. It is the same as single quotes. Even triple quotes can be used for the same result. Try to guess output of this code : print('''it's not a problem using "triple" quotes''').

**⚠️Avoid ! :**

* Do not mix two quotes style in one string.

**print('We should have enough time for our family")** This code gives an error because it starts with single quotes ends with double. See below:

input :



1

2

print('We should have enough time for our family")

output :



1

2

3

4

5

File "", line 1

print('We should have enough time for our family")

^

SyntaxError: EOL while scanning string literal

Let's do the same with numbers:

input :



1

2

print(572)

output :



1

2

572

input :



1

2

print(3.14)

output :



1

2

3.14

input :



1

2

print('3.14')

output :



1

2

3.14

**💡Tips:**

* If you have noticed we used 👉🏻**.** not 👉🏻**,** for the decimal number: 3.14.
* Surrounding the expression with quotes makes it in string type. We will see immutable types (int, string, tuple, etc.) in the next lessons.

We assumed that you have learned how to write simple Python code consisting of a single line which just prints a text. But, the real-world codes of Python contain a huge number of lines. So, now we will write codes that print multiple lines.

Let's look at the example below. And this is our first multi-line code. The following code prints three lines:

input :



1

2

3

4

print('no pain')

print('no gain')

print(3.14)

output :



1

2

3

4

no pain

no gain

3.14

If you need an empty line, you can use only print() function. Let's try :

input :



1

2

3

4

print('first line')

print() # second line is empty

print('third line')

output :

PEP 8 Conventions

What is PEP 8 ?

**PEP** stands for Python Enhancement Proposal. PEP 8 is a coding convention, a set of recommendations, about how to write your Python code more readable.

In other words, PEP 8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution. One of Guido's (author of Python) key insights is that code is read much more often than it is written.

The guidelines **(PEP 8)** provided [**here**](https://www.python.org/dev/peps/pep-0008/#introduction) are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.

A style guide is about consistency. Consistency with this style guide is **important**. Consistency tutarlılık within a project is **more important**. Consistency within one module or function is the **most important**.

However, know when to be inconsistent. Sometimes style guide recommendations aren't just applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask someone else.

The main idea of PEP 8 is to use the same code style for all Python projects as if they were written by the same programmer. PEP 8, even for beginners, assures that it will easily understand the code written by any other developer.

Q: What is PEP 8?  
A: PEP stands for Python Enhancement Proposal. PEP 8 is a coding convention, sozlesme a set of recommendation, about how to write your Python code more readable. In other words, PEP 8 is a document that gives coding conventions for the Python code comprising the standard library in the main Python distribution.

## PEP 8 Conventions

### Some Important PEP 8 Rules

We will show you some important PEP 8 traditional rules that you can follow.

* Limit all lines to a maximum of **79 characters**. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to **72 characters**. During this course, we will learn some ways of reducing the length of lines.
* **Spaces**are the preferred indentation Girinti method. **Tabs** should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows izinler mixing the use of tabs and spaces for indentation.
* Avoid extraneous **whitespaces** yabanci bosluklari in the following situations:

Immediately inside parentheses, brackets or braces :

**YES** : spam(meat[1], {milk: 2}) , **NO** : spam( meat[ 1 ], { milk: 2 } )

Between a trailing comma and a following close parenthesis :

**YES** : df[0,] or foo = (2,) , **NO** : df[0, ] or foo = (2, )

Immediately before a comma, semicolon, or colon :

**YES** : if y == 3: print x, y; x, y = y, x , **NO** : if y == 3 : print x , y ; x , y = y , x

Immediately before the open parenthesis that starts the argument list of a function call:

**YES** : print('peace') , **NO** : print ('peace')

More than one space around an assignment (or other) operator to align it with another:

| **YES** | **NO** |
| --- | --- |
| x = 3 | x =mmmmm3 |
| y = 4 | y =mmmmm4 |
| long\_vars = 5 | long\_vars = 5 |

* Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker.
* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -=, etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

Failure to follow the basic rules of PEP 8 does not make your program wrong or unable to work. In the near future, you will learn a lot about Python and become a more skilled programmer, but it will always be important to follow the code style.

There's nothing to worry about following PEP 8. You don't need to learn the traditional PEP 8 rules all at once right away. When you need it, you can open and read it now and then. We will also show you some PEP 8 conventions throughout this course.

You can check if your code complies with traditional PEP 8 rules using this [**module**](http://pep8online.com/)**.**

## Comments and Docstrings

### Introduction

When writing a program, you will need to add explanatory notes to others or even yourself. The longer you write lines, the better you will understand the necessity of this. We can add these explanatory notes to our program as 'comment' or as 'docstring' in more detail.

## Comments and Docstrings

### Comments

**Comments** are used to explain code when the basic code itself isn't clear. Python ignores comments, and so will not execute code in there, or raise syntax errors for plain English sentences.

There are three types of commenting methods. These are :

* **Single-line comments** begin with the hash character 👉🏻# and are terminated by the end of the line. 👉🏻# sign converts all subsequent characters to the comment form that Python does nothing.

input :



1

2

# This is a single line comment

* **Inline comments** satir ici aciklamalar also begin with hash character 👉🏻# and start from the end of a code line.

input :



1

2

print('the cosmos has no superiority to chaos') # This is an

    inline comment

output :



1

2

the cosmos has no superiority to chaos

* **Multi-line comments** basically consist of multiple comment lines.

input :



1

2

3

4

5

print(3 + 4)

# This is the multi-line comment, line-1

# This is the multi-line comment, line-2

# This is the multi-line comment, line-3

output :



1

2

7

**💡Tips:**

* To begin with, after 👉🏻 **#** there should be one space, and in the inline comments, there should be at least two spaces between the end of the code and 👉🏻 **#**.
* A comment is not a piece of code. It should be short. It's better to split a long comment into multiple lines. You have to add 👉🏻 **#** at the beginning of each new line.

Apart from the well-readable syntax itself, in writing Python programs, there are other important things that contribute to understandability of your program. We assume that you are familiar with comments and how they help in understanding codes.

In the real programming world, comments become especially important as the program gets bigger and more complicated. Without using them, things may get confusing for other developers who see your code for the first time. It may get confusing even for you within a couple of months after writing the program.

**⚠️Avoid ! :**

* More comments don't necessarily need to be better. If code is self-explanatory, comments are unnecessary.
* Do not make unnecessary comments. Usually, comments should answer the question **why** as opposed to **what**.
* When necessary, **update your comment**. Be sure that your comments will not be in contradiction to the code.

Q: What are the comments and how do you write it in Python?  
A: Comments are used to explain code when the basic code itself isn't clear. Python ignores comments, and so will not execute code in there, or raise syntax errors for plain English sentences. Comments in Python start with a # character. '#' character converts all subsequent characters to the comment form that Python does nothing.  
  
# this is a single line comment  
  
print("Hello World!") # this is an inline comment

## Comments and Docstrings

### Docstrings

We have to say at the beginning that you will not learn to create and write docstrings in this course. Only what we will show you is what docstrings are and how we will call and display it.

**Docstrings** are - unlike regular comments - stored as an attribute of the function or the module they document, meaning that you can access them programmatically. Docstring runs as an explanatory text of codes and it should be written between triple quotes. Like: """docstring""".

**💡Tips:**

* You don't need to learn or know; 'what the **function** and the **module** are?' for now.
* We will show you these topics in the Python Basics Plus Course.

Although it is not mandatory to learn, for now, you can consider the definitions of these terms below:

* We have briefly mentioned its meaning before. A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions.
* A **module** is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

Normally, when we want to call docstring of a function or module to read, we will use **\_\_doc\_\_** (the keyword doc enclosed by double underscores) syntax. See the example below :

Here is an example:

input :



1

2

3

4

5

6

7

8

9

def function(): # Don't be confused, we use 'def()' to create a

    function.

# You will see it in the next lessons.

"""

Hi, I am the docstring of this code.

If you need any information about this function or module, read me

    .

It can help you understand how the module or function works.

"""

print(function.\_\_doc\_\_)

output :



1

2

3

4

Hi, I am the docstring of this code.

If you need any information about this function or module, read me.

It can help you understand how the module or function works.

Let's see the docstring of print() function:

input :



1

2

print(print.\_\_doc\_\_)

output :



1

2

3

4

5

6

7

8

9

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys

    .stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

Q: What is docstring in Python?  
A: Docstrings are - unlike regular comments - stored as an attribute of the function or the module they document, meaning that you can access them programmatically. Docstring runs as an explanatory text of codes and it should be written between triple quotes.

## Naming Variables

### General Description

As you know, each variable has a unique name that distinguishes it from others. Giving a good name to a variable may not be as simple as it sounds.

A Python **variable** is a reserved memory location to store values. In other words, a variable gives data to the computer for processing. We will discuss variables in detail in the next lessons.

**💡Tips:**

* Remember, a nice and meaningful naming of variables is a skill that can be gained over time. Of course, you also need to be familiar with PEP 8 traditional rules.

Expert programmers care much for naming the variables well to make their codes easy to understand. It is important because programmers spend a lot of time reading and understanding code written by other programmers.

The convention of naming is optional. You can use any names you like but it is useful to follow the convention so that someone (including you) knows what you have written.

## Naming Variables

### Conventional (PEP 8) Naming Rules

If variables have poor names, even your own code may seem unclear to you in a couple of months. Now let's learn how to choose good names for our variables in accordance with PEP 8 rules:

* Choose lowercase words and use underscore to split the words:
* **price = [22, 44, 66]**,
* **low\_price = 12.00**
* Do not use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single-character variable names. In some fonts, these characters are indistinguishable from the numerals one and zero. If you want to use 'l', use 'L' instead.
* **l = 'It is not correct use'**,
* **O = "It's also incorrect use"**

**⚠️Avoid ! :**

* Do not use specific Python keywords (name of a function or phrase) as a name, like sum, max, min, in, or, for, etc.
* Use a sensible name. The variable name needs to be legible and meaningful and explain to the reader what types of values will be stored in it.
* **figures = 'this is better'**,
* **f = 'it is not meaningful'**
* Don't choose too common names. Use a name to describe the meaning of the variable. However, try to limit it to no more than 3 words.
* If the word you intend to choose is long, try to find the most common and expected short form to make it easy to predict later.

| **Variable to be named** | **Sample of Good name** | **Sample of Bad name** | **Why bad?** |
| --- | --- | --- | --- |
| Cleaned Data | cleaned\_data | cdat | it doesn't make sense enough. |
| Indexes of the Clear Application Syntaxes | clr\_app\_syntx | ix\_app\_syntax | it doesn't make sense enough. |
| Customer Information of the Bank Accounts | customer\_bank\_info | costomer\_info\_bank\_account | it's too wordy. |

Q: Which of the following is an invalid statement?  
A:  
a) x, y, z = 1, 22, 333  
b) x\_y\_z = 1\_234\_567  
c) xyz = 1234567  
**d) x y z = 111 222 333**  
  
Spaces are not allowed in variable names