

## PEP 8 Conventions

**Done: Go through the activity to the end**

✓What is PEP 8 ?

✓Some Important PEP 8 Rules

### 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](#) 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 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, 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.

- Interview Q&A

### 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 method. **Tabs** should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation.
- Avoid extraneous **whitespaces** 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

```
1 x = 3
2 y = 4
3 long_vars = 5
4
```

NO

```
1 x =          3
2 y =          4
3 long_vars = 5
4
```

- 
- 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](#).

## Comments and Docstrings

**Done: Go through the activity to the end**

✓ [Introduction](#)

✓ [Comments](#)

✓ [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

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 # This is a single line comment
2
```

- 
- Inline comments also begin with hash character  `#` and start from the end of a code line.

---

input :

```
1 print('the cosmos has no superiority to chaos') #
This is an inline comment
2
```

output :

```
1 the cosmos has no superiority to chaos
2
```

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

---

input :




```
1 print(3 + 4)
2 # This is the multi-line comment, line-1
3 # This is the multi-line comment, line-2
4 # This is the multi-line comment, line-3
5
```

output :

```
1 7
2
```



#### 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
```

- Interview Q&A

## 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 def function(): # Don't be confused, we use
  'def()' to create a function.
2 # You will see it in the next lessons.
```



```
3 """
4 Hi, I am the docstring of this code.
5 If you need any information about this function or
module, read me.
6 It can help you understand how the module or
function works.
7 """
8 print(function.__doc__)
```

output :

```
1 Hi, I am the docstring of this code.
2 If you need any information about this function or
module, read me.
3 It can help you understand how the module or
function works.
```

---

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

---

input :

```
1 print(print.__doc__)
2
```

output :

```
1 print(value, ..., sep=' ', end='\n',
file=sys.stdout, flush=False)
2 Prints the values to a stream, or to sys.stdout by
default.
```

```
3 Optional keyword arguments:
4 file: a file-like object (stream); defaults to
the current sys.stdout.
5 sep: string inserted between values, default a
space.
6 end: string appended after the last value,
default a newline.
7 flush: whether to forcibly flush the stream.
8
```

---

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.

- Interview Q&A

## Naming Variables

**Done: Go through the activity to the end**

✓ [General Description](#)

✓ [Conventional \(PEP 8\) Naming Rules](#)

### 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.

### 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	<code>cleaned_data</code>	<code>cdat</code>	it doesn't make sense enough.
Indexes of the Clear Application Syntaxes	<code>clr_app_syntax</code>	<code>ix_app_syntax</code>	it doesn't make sense enough.
Customer Information of the Bank Accounts	<code>customer_bank_info</code>	<code>costomer_info_bank_account</code>	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

- Interview Q&A