

[Implementing coding practices in Python using PEP8.]

**PEP8 is a style guide for python code.**

PEP stands for Python Enhancement Proposal, and they describe and document the way python language evolves. It was written in 2001 by Guido van Rossum, Barry Warsaw, and Nick Coghlan. A PEP is a document that describes new features proposed for Python and documents aspects of Python, like design and style, for the community. They also provide a reference point (and a standard) for the pythonic way to write code. It also has a lot of programming recommendations and useful tips on various topics, which aim to improve readability and reliability of your code.

**PEP8 features:-**

1. Plugin architecture: Adding new checks is easy.
2. Parseable output: Jump to error location in your editor.
3. Small: Just one Python file, requires only `stdlib`. You can use just the `pep8.py` file for this purpose.

**Naming Conventions**

1. Variable
2. Function
3. Class
4. Method
5. Constant
6. Module
7. Package`

**Variable:** A variable is created the moment you first assign a value to it

```
#Wrong Way to Initialize or assigning a name to a variable
#Name Should not start with a number
#Name should be intuitive and not too common.

1variable=2 #Variable name started with a number (Wrong Way)
print(1variable)
```

```
File "<ipython-input-1-d1860915d72c>", line 5
    1variable=2
    ^
```

SyntaxError: invalid syntax

```
#Wrong Way to Initialize or assigning a name to a variable
#Name Should not start with a number
#Name should be intuitive and not too common.
```

```
first_name='Bhavana' #Variable name is self-explanatory and has a readability, and it is seperated using underscores
print(x)
```

Bhavana

**Function:** A function is a block of code which only runs when it is called.

```
#Wrong Way to Initialize or assigning a name to a function
#Name Should not start with a number
#Name should be intuitive and not too common.

def ^function(): #Function name should not be started with a Number or special characters
    print("Not a correct way to represent a function name")

^function()
```

```
File "<ipython-input-5-5f84f1733e34>", line 5
    def ^function():
    ^
SyntaxError: invalid syntax
```

```
#Wrong Way to Initialize or assigning a name to a function
#Name Should not start with a number
#Name should be intuitive and not too common.
```

```
def x(): #Function name is too generic and it can create a confusion in enterprise programming
    print("Function Name is too generic, you can use it but it is not recommended as it is not self-explanatory and intuitive")

x()
```

Function Name is too generic, you can use it but it is not recommended as it is not self-explanatory and intuitive

```
#Wrong Way to Initialize or assigning a name to a function
#Name Should not start with a number
#Name should be intuitive and not too common.
```

```
def display_function(): #Function name is self explanatory
    print("Function Name is self explanatory, name can be more intuitive in case of proper functionality")

display_function()
```

Function Name is self explanatory, name can be more intuitive in case of proper functionality

**Class:** class definitions begin with a [class](#) keyword.

```
#Wrong Way to Initialize or assigning a name to a class
#Name Should not start with a number
#Name should be intuitive and not too common.

1class x:
def display_function(): #Function name is self explanatory
    print("Function Name is self explanatory, name can be more intuitive in case of proper functionality")

display_function()

File "<ipython-input-9-0547726683a1>", line 5
    1class x:
        ^
SyntaxError: invalid syntax
```

```
class Employee:
    def accept(self):
        print("Enter Id:")
        self.Id=int(input())
        print("Enter Name:")
        self.name= str(input())
    def display(self):
        print("ID: %d \nName: %s"%(self.Id,self.name))

emp=Employee()
emp.accept()
emp.display()

Enter Id:
66
Enter Name:
bhavana
ID: 66
Name: bhavana
```

**Method:** A Python method is a label that you can call on an object; it is a piece of code to execute on that object.

```
#Wrong Way to Initialize or assigning a name to a method
#Name Should not start with a number
#Name should be intuitive and not too common.
```

```
1class Method:
    def display(self):
        print("This is method function. ")

c = Method()
c.display()
```

```
File "<ipython-input-26-3e88b14da450>", line 6
1class Method:
```

```
SyntaxError: invalid syntax
```

```
#Wrong Way to Initialize or assigning a name to a class
#Name Should not start with a number
#Name should be intuitive and not too common

class Product:
    def __init__(self):
        self.prod_id = input("Enter the Product ID: ")
        self.prod_name = input("Enter the Product Name: ")
        self.total_no = int(input("Enter the total no. of Items Purchase: "))
        self.unit_price=float(input("Enter the unit Price: "))
    def display(self):
        print("Total Price of %d units of Product %s is: %0.2f" %(self.total_no,self.prod_name,self.total_no*self.unit_price))

p1=Product()
p1.display()
```

```
Enter the Product ID: A20134
Enter the Product Name: Chocolate
Enter the total no. of Items Purchase: 7
Enter the unit Price: 75.50
Total Price of 7 units of Product Chocolate is: 528.50
```

**Constant:** A constant is a type of variable whose value cannot be changed.

```
pi = 3.14                #pi is constant
radius=5
print("Area of circle: %0.2f" %(pi*radius*radius))
```

```
Area of circle: 78.50
```

**Modules:** Modules refer to a file containing Python statements and definitions.

```
# to import standard module math

import math
print("The value of pi is", math.pi)

The value of pi is 3.141592653589793
```

**Packages:** A package is basically a directory with Python files and a file with the name `__init__.py`



### **Code layout:**

#### **WITHOUT SPACE:**

These conventions lead to text that you can read easily, like this:  
This would become increasingly hard to read.

For example have a look at the example below

```
howwillitlookifwedonothavethespace
```

#### **WITH SPACE:**

Now here, we will use space and write it in regular English language, so it will be very easy to read.

How will it look if we do not have the space

### **Maximum line length and line breaking:**

PEP 8 guidelines suggest that each line of code (as well as comment lines) should be 79 characters wide or less. This is a common standard that is also used in other languages including R.

## **CORRECT:**

---

```
# Perform some math
a = 1+2
b = 3+4
c = a+b

# Read in and plot some
precip_timeseries = pd.readcsv("precip-2019.csv")
precip_timeseries.plot()
```

## **#WRONG**

```
#Perform some math and do some things
a=1+2
b=3+4
c=a+b
data=pd.readcsv("precip-2019.csv")
data.plot()
```

## Should a line break Before or After a Binary Operator

Here, it's harder to see which variable is being added and which is subtracted.

# WRONG

```
Total = (Number 1+  
         Number 2-  
         Number 3)
```

You can immediately see which variable is being added or subtracted, as the operator is right next to the variable being operated on.

In the below Example

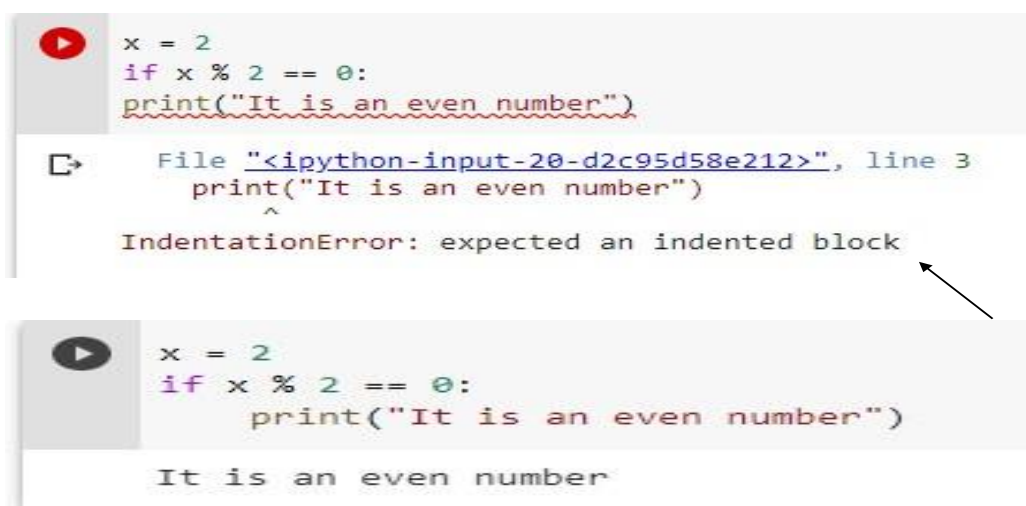
#CORRECT

```
Total = (Number 1  
        + Number 2  
        - Number 3)
```

## Indentation

- Indentation is extremely important in Python.
- The Indentation level of lines of code in python determines how statements are grouped together.


1. Expected an indented block



## 2. Unexpected Indent:

```
x = 2
if x % 2 == 0:
    print("It is an even number")
```


File "<ipython-input-24-a296ed44a7f2>", line 2  
if x % 2 == 0:  
^  
IndentationError: unexpected indent



## 3. Unindent does not match any outer indentation level:

```
def greeting():
    print("Greetings of the day")
    return
greeting()
```

File "<ipython-input-30-698032a46f85>", line 3  
return  
^  
IndentationError: unindent does not match any outer indentation level



```
def greeting():
    print("Greetings of the day")
    return

greeting()

Greetings of the day
```

## Tabs vs. Spaces:

The key indentation rules laid out by PEP 8 are the following:

- Use 4 consecutive spaces to indicate indentation.
- Prefer spaces over tabs.



## Indentation following line breaks:

- Add a comment after the final condition. Due to syntax highlighting in most editors, this will separate the conditions from the nested code:

### # Not Recommended

```
x = 5
if (x > 3 and
    x < 10):
    print(x)
```

### Recommended

```
x = 5
if (x > 3 and
    x < 10):
    # Both conditions satisfied
    print(x)
```

```
x = 5
if (x > 3 and
    x < 10):
    print(x)
```

### # Not Recommended

```
def function(
    arg_one, arg_two,
    arg_three, arg_four):
    return arg_one
```

### # Recommended

```
def function(
    arg_one, arg_two,
    arg_three, arg_four):
    return arg_one
```

## Where to put the closing Braces:

### # Not Recommended

```
▶ list_of_numbers = [ 1, 2, 3,  
                     4, 5, 6,  
                     7, 8, 9]
```

### Recommended Method

```
▶ list_of_numbers = [  
    1, 2, 3,  
    4, 5, 6,  
    7, 8, 9  
]
```

### Method

```
▶ list_of_numbers = [  
    1, 2, 3,  
    4, 5, 6,  
    7, 8, 9  
]
```

## COMMENTS:

Comments are lines that exist in computer programs that are ignored by compilers and interpreters.

Comment begins with a hash mark (#) Generally, comment looks like this:

*# this a comment*

Because comment does not execute ,when you will run program you will not see any indication of the comment there.

## BLOCK COMMENTS:

Each line of block comments starts with a # and a single space

Paragraphs inside a block comment are separated by a line containing a single #.

## Anti-pattern

```
#This comment needs a space  
def print_name(self):  
    print(self.name)
```

## Best practice

```
# Comment is correct now  
def print_name(self):  
    print(self.name)
```

### **INLINE COMMENTS:**

Inline comment should be separated by at least two spaces from the comment.

They should start with a # and a single space

Inline comments are unnecessary and in fact distracting if they state the obvious

## Anti-pattern

```
def print_name(self):  
    print(self.name) #This comment needs a space
```

## Best practice

```
def print_name(self):  
    print(self.name) # Comment is correct now
```

### **DOCSTRING COMMENTS:**

A docstring is added as a comment string right below the function,module,or object.

**Rules:** A docstring is either a single line, or a multi-line comment In latter case, the first line is short description, and after the first line an empty line follows

This is a basic example of what it looks like:

```
def add(value1, value2):  
    """Calculate the sum of value1 and value2."""  
    return value1 + value2
```

In the Python interactive help system, the docstring is then made available via the `__doc__` attribute.

```
>>> print add.__doc__  
Calculate the sum of value1 and value2.
```

## **Whitespace in Expressions and Statements**

### **1. Whitespace Around Binary Operators**

Surround the following binary operators with a single space on either side:

- Assignment operators (=, +=, -=, and so forth)
- Comparisons (==, !=, >, <, >=, <=) and (is, is not, in, not in)
- Booleans (and, or, not)

**Note:** When = is used to assign a default value to a function argument, do not surround it with spaces.

Python

```
# Recommended  
def function(default_parameter=5):  
    # ...  
  
# Not recommended  
def function(default_parameter = 5):  
    # ...
```

Adding space when there is more than one operator in a statement

```
Python
# Recommended
y = x**2 + 5
z = (x+y) * (x-y)

# Not Recommended
y = x ** 2 + 5
z = (x + y) * (x - y)
```

Adding space to if statements where there are multiple conditions.

```
Python
# Not recommended
if x > 5 and x % 2 == 0:
    print('x is larger than 5 and divisible by 2!')

Python
# Recommended
if x>5 and x%2==0:
    print('x is larger than 5 and divisible by 2!')
```

Note : Use the same amount of whitespace either side of the operator.

The following is not acceptable :

```
Python
# Definitely do not do this!
if x >5 and x% 2== 0:
    print('x is larger than 5 and divisible by 2!')
```

## When to Avoid Adding Whitespace

Trailing space Immediately inside parentheses, brackets, or braces:

```
Python

# Recommended
my_list = [1, 2, 3]

# Not recommended
my_list = [ 1, 2, 3, ]
```

Before a comma, semicolon, or colon:

```
Python

x = 5
y = 6

# Recommended
print(x, y)

# Not recommended
print(x , y)
```

```
Python

# Recommended
list[3]

# Not recommended
list [3]
```

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

```
Python

def double(x):
    return x * 2

# Recommended
double(3)

# Not recommended
double (3)
```

Before the open bracket that starts an index or slice:  
Between a trailing comma and a closing parenthesis:

#### Python

```
# Recommended
tuple = (1,)

# Not recommended
tuple = (1, )
```

To align assignment operators:

#### Python

```
# Recommended
var1 = 5
var2 = 6
some_long_var = 7

# Not recommended
var1      = 5
var2      = 6
some_long_var = 7
```

## Programming Recommendations by PEP-8:

C) # Not recommended

```
my_list = []
if not len(my_list):
    print('List is empty!')
```

D) # Recommended

```
my_list = []  
if not my_list:  
    print('List is empty!')
```

*In the above program D is recommended over C by the PEP-8*

### **When to Ignore PEP-8**

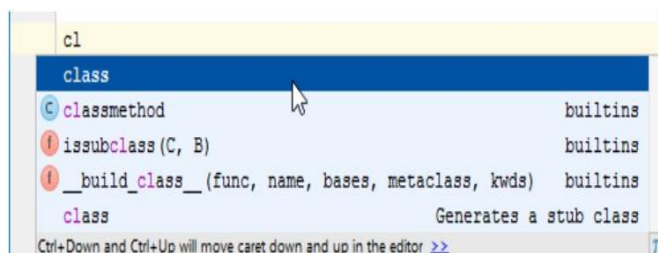
Answer: never.

Though, there are some guidelines in PEP-8 that are inconvenient in some instances:

- Complying with PEP-8
  - Code surrounding
  - Code compatibility

### **Tips and Tricks to Help Ensure Your Code Follows PEP 8**

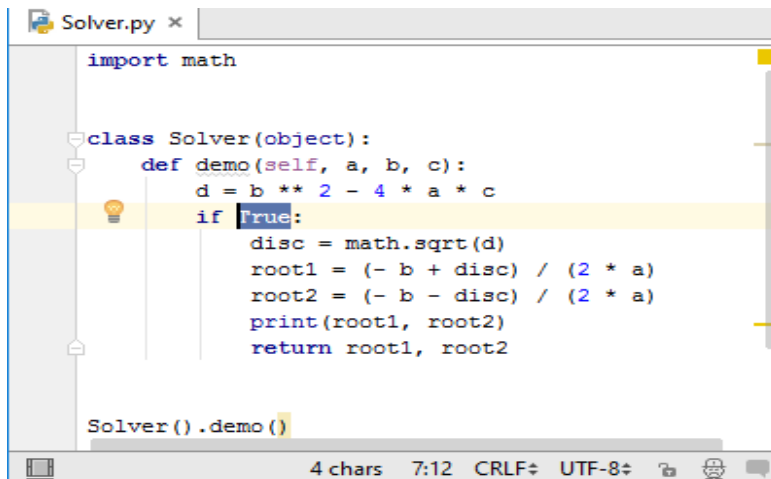
#### **Highlighting code style violations:**





## Generating Source code:

Select if option from the suggestion list. As you see, PyCharm automatically adds `if True:` and indents the selected lines:



## Linter-python-pep8 package

This linter-python-pep8 plugin or Linter provides an interface to pep8. It will be used with files that have the Python syntax.

### Installation:

Before using this plugin, you should make sure that pep8 is installed on your system. You can follow following instructions to install pep8:

Install python.

Install pep8 by typing the following in a terminal:

### Black:

Black can be installed by running `pip install black`. It requires Python 3.6.0+ to run. Once Black is installed, you will have a new command-line tools called black available to you in your shell, and you're ready to start.

```
$ pip install black
```

## Example: Write a program using PEP8 rules.

Answer:

```
# Names of class and methods are self-explanatory as per the naming convention
```

```
class Rectangle:
    def Take_input(self):
        print("Enter the Length:")
        self.l=float(input())
        print("Enter the breadth:")
        self.b=float(input())

    def CalculateArea(self):
        area=self.l*self.b
        print("Area of rectangle is =%f" %(area))

    def CalculatePerimeter(self):
        perimeter=2*(self.l*self.b)
        print("Perimeter of rectangle is =%f" %(perimeter))

c= Rectangle()
c.Take_input()
c.CalculateArea()
c.CalculatePerimeter()
```

```
Enter the Length:
5
Enter the breadth:
7
Area of rectangle is =35.000000
Perimeter of rectangle is =70.000000
```

2.

Used Proper Whitespaces and line breaks so that it can be read easily.

```
data = []  
  
n = int(input('Enter the number of elements: '))  
  
for i in range (0, n):  
    element = int(input('Enter the element: '))  
    data.append(element)  
  
print('The data set is: ', data)
```

```
Enter the number of elements: 7  
Enter the element: 21  
Enter the element: 66  
Enter the element: 67  
Enter the element: 110  
Enter the element: 20  
Enter the element: 65  
Enter the element: 80  
The data set is: [21, 66, 67, 110, 20, 65, 80]
```

3.

Imported the module into a Python script using keyword *import*.

```
import fractions  
print(fractions.Fraction('1.5'))  
print(fractions.Fraction(5))  
print(fractions.Fraction(4,30))
```

```
3/2  
5  
2/15
```

4.

Using the comments while developing a code/script can give the information about the functionality and documentation to other user who is reading the code.

```
# Python program to find simple interest

p = float(input("Enter the principle amount : "))
r = float(input("Enter the rate of interest : "))
t = float(input("Enter the time in the years: "))

si = (p*r*t)/100          # calculating simple interest

print("Principle amount: ", p)
print("Interest rate   : ", r)
print("Time in years   : ", t)
print("Simple Interest : ", si)
```

```
Enter the principle amount : 25000
Enter the rate of interest : 3
Enter the time in the years: 12
Principle amount: 25000.0
Interest rate   : 3.0
Time in years   : 12.0
Simple Interest : 9000.0
```

## 5.

### Make the Proper alignment of statements

```
students = [
    ("Bhavana", ["CompSci", "Physics"]),
    ("Vanshu", ["Maths", "CompSci", "Stats"]),
    ("Jess", ["CompSci", "Accounting", "Economics", "Management"]),
    ("Sarah", ["InfSys", "Accounting", "Economics", "CommLaw"]),
    ("Zuki", ["Sociology", "Economics", "Law", "Stats", "Music"])]
```

```
# Print all students with a count of their courses.
for name, subjects in students:
    print(name, "takes", len(subjects), "courses")
```

```
Bhavana takes 2 courses
Vanshu takes 3 courses
Jess takes 4 courses
Sarah takes 4 courses
Zuki takes 5 courses
```