Practical no: 7

Implementing Coding Practices In Python Using PEP8

What is PEP 8?

Pep 8 is a document that provides guidelines and best practices on how to write Python code. It was written in 2001 by Guido van Rossum, Barry Warsaw and Nick Coghlan. The primary focus of PEP 8 is to improve the readability and consistency of Python code. PEP stands for Python Enchantment Proposal and there are several of them. A PEP is a document that describes new features proposed for python and document aspects of Python, like design and style for the community.

Why follow PEP 8?

- Readable code
    1. Mostly for yourself (and other developers)
    2. Not discussed here
- User-friendly code
    1. Following naming conventions is a way to document your code.
    2. Mostly useful for users(which can be developers too)
    3. Discussed here!

The most important naming conventions

regular_variables

- Variable names should be lowercase, where necessary separating words by underscores

```
1 a_car_name = 'Tesla'
```

CONSTANTS

- In python, all variables can be modified
- Therefore, real constants don’t exist
- But to indicate that a variable should be treated as if it were a constant, names should be uppercase, where necessary separating words by underscores

```
1 CAR_NAMES = ['Tesla','Mercedes','BMW']
```

function_names()

- Names of functions and class methods should be lowercase, where necessary separating words by underscores

```
import random

def random_car_name():
  return random.choice(CAR_NAMES)

print(random_car_name())
```

```
Tesla
```

ClassNames

- Class names should capitalize the first letter of each word

```
class CarName:

  def __init__(self, name):
    self.name = name

  def __str__(self):
    return self.name

mercedes = CarName(name='Mercedes')
print(mercedes)
```

```
Mercedes
```

FactoryFunctionNames()

- Factory functions return objects
- Therefore, to users of your code, factory functions act like class definitions
- To reflect this, factory-function names should also capitalize the first letter of each word

```
def BMW():
  return CarName(name='BMW')

bmw = BMW()
print(bmw)
```

```
BMW
```

_non_public_properties

- In Python, properties can be accessed from anywhere
- Therefore, private class properties don’t exist
- But to indicate that a property should be treated as if it were private, names should be prefixed with an underscore

```
class Tesla(CarName):

  def __init__(self):
    CarName.__init__(self, name='Tesla')
    self._part = 'Hood'

  @property
  def part(self):

    return self._part

tesla = Tesla()
print(tesla.part)
```

```
Hood
```

conflicting_names_

- If a name is already taken, suffix an underscore

```
in_ = 'Car'
```