Coding Conventions and Guidelines (1)

Naming Convention

1. General:

- Avoid using names that are too general or too wordy. For example:
  - × Bad Practice: *data_structure, my_list, info_map, dictionary_for_the_purpose _of_storing_data_representing _word_definitions.*
  - ✓ Good Practice: *user_profile, menu_options, word_definitions*
- When using CamelCase names, capitalize all letters of an abbreviation (e.g. *HTTPServer*).

2. Packages & Modules:

- Packages names should be all lower case.
- When multiple words are needed, an underscore should separate them.
- It is usually preferable to stick to one-word names.

3. Classes:

- Class names should follow the PascalCase convention. For example:

```
5   class Faculty(models.Model):
6       name = models.CharField(max_length=200, null=True)
7
8       def __str__(self):
9           return self.name
```

```
186   class ObeMark(models.Model):
187       course = models.ForeignKey(AssignCours
188       student = models.ForeignKey(AssignStuc
189       # course = models.ForeignKey(AssignCou
```

- Class based model name should normally use the Pascal Case convention and end with "Model". **For Example:** *class ProductModel*
- Class based form name should be Camel casing and end with "Form". **For example:** *signUpForm, productForm.*
- Class based model attribute name should start with "m_" like as *m_productId, m_productName.*
- Class based form attribute name should start with "f_" like as *f_productId, f_quantity*.

4. Variables:

- Instance and Global variables should be in snake_case. For example: ***exam_roll***, ***student_id***.

- Non_public instance variables should begin with a single underscore. For example:

```
class Sample:
    def __init__(self):
        self.foo = "lorem"
        self._bar = "ipsum"
```

5. Methods:

- Method names should be in lowercase (if a single word) or snake_case(if multiple words). For example:

```
def student_account(request):

    student = request.user.student
    form = StudentForm(instance=student)
```

```
def main(request):
    students = Student.objects.all()
    teachers = Teacher.objects.all()
    stdcnt = Student.objects.count()
```

- Non-public method should begin with a single underscore.

6. Method Arguments:

- Instance methods should have their first argument named 'self'.

```
class NonObeMark(models.Model):
    course = models.ForeignKey(AssignCourse, null
    student = models.ForeignKey(AssignStudent,nul
    #course = models.ForeignKey(AssignCourse,null
    tt1 = models.IntegerField(default=-1,null=Tru
    tt2 = models.IntegerField(default=-1,null=Tru
    tt3 = models.IntegerField(default=-1,null=Tru
    att = models.IntegerField(default=-1,null=Tru
    sem = models.IntegerField(default=-1,null=Tru
    mark = models.CharField(default='PENDING',nul
    def avg(self):
        return (self.tt1+self.tt2+self.tt3)/3
```

7. Functions:

- Function names should be all lower case(single word) or snake_case (multiple words).

8. Constants:

- Constant names must be fully capitalized.
- Words in a constant name should be separated by an underscore.

9. Specific Names:

- Is prefix should be used for Boolean variables and methods**.**
  For example*: isSet, isVisible, isOpen*

- Plural forms should be used on names representing a collection of objects.
  For example: int values[]
- n prefix should be used for variables representing a number of objects.
  For Example: *nPoints, nLines*

Code Layout

1. Indentation:

- Use 4 spaces per indentation level.
- Spaces are preferred over tabs.

```
# Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
def long_function_name(
        var_one, var_two, var_three,
        var_four):
    print(var_one)
```

2. Maximum Line Length and Line break:

- Limit all lines to a maximum of 79 characters.
- Make sure to indent the continued line appropriately.
- In Python code, it is permissible to break before or after a binary operator, as long as the convention is consistent locally. For new code Knuth's style is suggested.

```
# Correct:
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

3. Blank Lines:

- Surround top-level function and class definitions with two blank lines.
- Method definitions inside a class are surrounded by a single blank line.
- Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).
- Use blank lines in functions, sparingly, to indicate logical sections.

4. Whitespace in Expressions and Statements:

Avoid extraneous whitespace in the following situations:

- Immediately inside parentheses, brackets or braces:

```
# Correct:
spam(ham[1], {eggs: 2})

# Wrong:
spam( ham[ 1 ], { eggs: 2 } )
```

- Between a trailing comma and a following close parenthesis:

```
# Correct:
foo = (0,)

# Wrong:
bar = (0, )
```

- Immediately before a comma, semicolon, or colon:

```
# Correct:
if x == 4: print(x, y); x, y = y, x

# Wrong:
if x == 4 : print(x , y) ; x , y = y , x
```

- If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority(ies). Use your own judgment; however, never use more than one space, and always have the same amount of whitespace on both sides of a binary operator:

```
# Correct:
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)

# Wrong:
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

5. Documentation String:

- Write docstrings for all public modules, functions, classes, and methods. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

- For one liner docstrings, please keep the closing """ on the same line:

```
"""Return an ex-parrot."""
```

6. Comments:

- Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!

```
x = x + 1                 # Increment x
```

7. Access Modifiers:

- **Public Access Modifier:** The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.
- **Protected Access Modifier:** Data members of a class are declared protected by adding a single underscore '_' symbol before the data member of that class.

```python
class Student:

    # protected data members
    _name = None
    _roll = None
    _branch = None

    # constructor
    def __init__(self, name, roll, branch):
        self._name = name
        self._roll = roll
        self._branch = branch

    # protected member function
    def _displayRollAndBranch(self):

        # accessing protected data members
        print("Roll: ", self._roll)
        print("Branch: ", self._branch)
```

- Data members of a class are declared private by adding a double underscore '__' symbol before the data member of that class.

```python
class Geek:

    # private members
    __name = None
    __roll = None
    __branch = None

    # constructor
    def __init__(self, name, roll, branch):
        self.__name = name
        self.__roll = roll
        self.__branch = branch

    # private member function
    def __displayDetails(self):

        # accessing private data members
        print("Name: ", self.__name)
        print("Roll: ", self.__roll)
        print("Branch: ", self.__branch)
```

References:

1. https://peps.python.org/pep-0008/
2. https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html
3. https://github.com/JU-CSE-27/swe-wiki/blob/master/resources/Updated_coding-standard.pdf
4. https://www.geeksforgeeks.org/access-modifiers-in-python-public-private-and-protected/