uc3m | Universidad Carlos III de Madrid

Coding Standard

UC3M Travel

100532491, 100495954, 100495680
Software Development

Table of Contents

Source Code Files

Copyright Legend Format

All source code files should include a copyright legend in the following format:

```
“””
      Copyright © [year] [author]
      UC3M
“””
```

Copyright should not be omitted, nor should any of the components of the example copyright legend above.

File Management

All source code, functions, and executables should be managed using version control. GitHub is the designated version control system for the TravelUC3M project. All team members are required to consistently commit to GitHub with detailed descriptions. No other version control system should be used. Team members must not infrequently commit large amounts of code.

Example of proper use:

```
git add
git commit -m “Edited error handling in the exampleMethod
function in the ExampleClass”
git push origin main
```

Example of improper use:

```
git add
git commit -m “ExampleClass edited”
git push origin main
```

Directory Structure

Follow a clear structure for storing source code files. The directory structure will follow logical organization for a Python project. Related files will be grouped together including essential files such as README.md and .gitignore.

Example proper use:

```
├── src/
│     ├── HotelManager.py
│     ├── HotelReservation.py
│     ├── HotelStay.py
│       ├── HotelManagementException.py
│       ├── main.py
│       └── README.md
├── tests/
├── .gitignore
└── __init__.py
```

Example improper use:

```
project/
├── src/
│       ├── controllers/
│       │       ├── HotelManager.py
│       │       ├── models/
│       │       │       ├── HotelReservation.py
│       │       │       └── HotelStay.py
│       ├── utils/
│       │       ├── HotelManagementException.py
│       ├── main.py
│       └── README.md
├── tests/
├── .gitignore
└── __init__.py
```

Standard for Names and Variables

File Per Class

Each class should have its own separate file. This helps maintain a clear structure.

Name each class file according to the class it contains. Include ".py" suffix.

Example proper use:

A filename "HotelManager.py" includes a python file with only one class named "HotelManager"

Example improper use:

A filename "HotelManager.py" a class that is not named "HotelManager" or more than one class than "HotelManager"

Header

Each class should have a header section with information about the functionality of the class. This description should include:

- Description of functionality
- Parameters or methods to be called

Example of proper use:

```
"""
This class represents a hotel reservation and includes
methods to manage reservation data.

Parameters:
- IDCARD (str): The identification card of the guest.
- creditcardNumb (str): The credit card number used for…

Methods to be called:
- CREDITCARD (property): Get the credit card number.
- CREDITCARD (setter): Set the credit …
"""

class HotelReservation:
```

Visibility

Variables should be private by default, using double underscores '__' to denote privacy.

Example of proper usage:

```
class Example:
    def __init__(self):
        self.__private_instance_variable =  x
    __private_static_variable =  y
```

Example of improper usage:

Not including underscores “__” to denote privacy.

```
class Example:
    def __init__(self):
        self.private_instance_variable =  x
    private_static_variable =  y
```

Code Formatting

Each statement should be on a separate line to increase readability. Executive declarations and initializations should be aligned, especially when repeated. Indentation should be used, using 4 spaces to indicate a level of indentation.

Example of proper usage:

```
variable1 = “x”
variable2 = “y”
variable3 = “z”

if condition:
    print("Condition is true")
```

Example of improper usage:

```
variable1 = “x”; variable2 = "y"; variable3 = “z”
if condition: print("Condition is true")
```

Variable Initialization

Initialize class attributes using ‘__init__’ method. Ensure that loop indices are initialized before ‘for’ loops.

Example proper usage:

```
class Example:
    def __init__(self):
        self.variable1 = “y”
        self.variable2 = “x”
        self.index = 0

    def example_method(self):
        for item in range(5):
            self.index += 1
```

Example improper usage:

Loop index is not defined before usage.

```
class Example:
    def __init__(self):
        self.variable1 = “y”
        self.variable2 = “x”

    def example_method(self):
        for item in range(5):
            self.index += 1
```

Use of “self”

The use of the “self” keyword when referencing class attributes in a method, especially with the risk of shadowing local variables.

Example proper use:

```
class Example1:
        def __init__(self):
            self.variable1 = “y”

        def example_method(self):
            print(self.variable1)

class Example2:
        def __init__(self):
            self.variable1 = “y”

        def example_method(self, variable1):
            self.variable1  =  variable1
```

Example improper use:

In this class example, “variable1” is a local variable instead of a class attribute.

```
class Example1:
        def __init__(self):
            variable1 = “y”
```

```
    def example_method(self):
        print(self.variable1)
```

In this class example, there is shadowing of variable 1.

```
class Example2:
    def __init__(self):
        self.variable1 = “y”

    def example_method(self, variable1):
        variable1  =   “x”
```

Standard for Methods

Method Structure

There will not be an explicit maximum method size. Methods should be organized into small, logically organized blocks of code. Descriptive comments are used within the body to explain the purpose and function of each logical block of code

Example of proper use:

A clearly defined method with a descriptive comment to describe its function.

```
class Example:
        def addPrint(x, y):
            “””
            Print two numbers then return the result
            “””

            print(x)
            print(y)

            result = x + y

            return result
```

Example of proper use:

No descriptive comment or logical blocks of code.

```
class Example:
        def addPrint(x, y):
            print(x)
            print(y)
            result = x + y
            return result
```

Indentation and Braces

The code uses 4 spaces for indentation. Always use parentheses to enclose conditions in loops and conditional statements. Parenthesis should be placed immediately after function names, when calling functions, and to enclose the arguments within them. Parenthesis can be used to group mathematical and logical expressions to ensure clarity and ensure correct evaluation.

Example of proper use:

```
def addPrint(x, y):
    print(x)
```

```
          print(y)

          result = x + y

          return result
```

Example of improper use:

Parenthesis do not come directly after the function name. Improper indentation.

```
def addPrint (x, y):
print(x)
print(y)

result = x + y

return result
```

Method Definitions

Parameters are aligned vertically if they don't fit on the same line as the method declaration. Each method includes a descriptive comment.

Example of proper use:

```
def add(x, #first integer
          y ):#second integer

          result = x + y

          return result
```

Example of improper use:

No descriptive comments, return is on the same line as method and parameter declaration.

```
def addPrint(x, y): return result
```

Standard for exception handling

Detecting Errors in Input Parameters

Input parameters should be validated within methods, including checking for missing values. File paths and JSON data are also validated before processing: to ensure the file exists and that it contains valid JSON data.

Example of proper use:

```
class DataProcessor:
    def process_data(self, file_path):
        if not os.path.exists(file_path):
            raise FileNotFoundError("File not found.")

        with open(file_path, 'r') as file:
            try:
                data = json.load(file)
            except json.JSONDecodeError:
                raise ValueError("Invalid JSON data in the
file.")

...

        return data
```

Example of improper use:

There is no validation performed, which could lead to potential errors if the file does not exist or contain valid JSON data.

```
class DataProcessor:
    def process_data(self, file_path):
        with open(file_path, 'r') as file:
            data = json.load(file)

...

        return data
```

Handling Expectations

Expectations should be handled properly either locally or raising them to be handled by higher-level components. For methods that have a high possibility of throwing exceptions, providing specific error messages for each type of expectation.

Example proper use:

```
def divide(self, x, y):
       try:
              res = x  / y
       except ZeroDivisionError:
              raise Exception(“Zero division.)
           return res
```

Example improper use:

Fails to include error handling.

```
def divide(self, x, y):
           return   x  / y
```

Parameter Splitting

Generally, parameters should be split around 80 characters. If the line goes above 80 characters it should be split into multiple lines. Parenthesis should be placed immediately after the function name followed by the parameter names. The parameter names should then be followed by another parenthesis, then a colon.

Example of proper use:

```
def __init__(self,
             IDCARD,
             creditcardNumb,
             nAMeAndSURNAME,
             phonenumber,
             room_type,
             numdays):
```

Example of improper use:

Parameters are not split into multiple logical lines.

```
def __init__(self, IDCARD, creditcardNumb, nAMeAndSURNAME,
phonenumber,room_type, numdays):
```

Standard for the application design

The division of functionality for the application is divided into separate classes. Each class represents its own responsibility in the management of a hotel. Each class will only adhere to a single responsibility in the system. The elements within each class are closely related and will closely contribute to a well-defined purpose.

The design of the application will be driven by domain knowledge of the hospitality industry. Each class represents real world entities within hotel reservation and management.