

Coding Standard Guidelines

Copyright (2025) Madeline and Laura, UC3M Students Software Development Group 16

Created on: 2025-02-05

Guided Exercise 1 Coding Standards

TABLE OF CONTENTS:

Structured Directory Organization	3
Standard for Names and Variables	4
Standard for Methods	5
Standard for Exception Handling	6

Structured Directory Organization

```
G88.2024.TXX.GE1/                                     # Root directory of your project
|— UC3MMoney/                                          # Python package
|  |— __init__.py                                     # Package initialization file
|  |— TransactionManager.py                           # File for managing transactions
|  |— TransactionRequest.py                           # File for handling requests
|  |— TransactionManagementException.py               # Custom exceptions
|— tests/                                              # Directory for test cases - (NOT IMPLEMENTED FOR
NOW)
|— docs/ # Documentation
|  |— CodingStandard.pdf                             # Our coding standard document
|— main.py                                             # Main program entry point
|— README.md                                           # Project overview or documentation
|— test.json                                           # Test data or configuration file
|— .gitignore                                          # Git ignore rules
```

Standard for Names and Variables

- Class names must follow PascalCase.

Accepted: left_vector

Rejected: Leftvector

- Local Variables should be written in snake_case

Accepted: LocalVar

Rejected: Localvar

- Static constants must be in uppercase with underscores

Accepted: MAX_COUNT_POS

Rejected: maxcountpos

- Instance variables should have a leading underscore and follow camelCase

Accepted: _totalNum

Rejected: totalNum

- Instance and Static Variables are allowed to be public.

- There should only be one statement per line

Accepted: x = 5

y = 6

Rejected: x = 5 y = 6

- Initialize Instance in the __init__() method

Accepted: class Colors:

```
def __init__(self, color):  
    self._color = color
```

Rejected: class Colors:

```
    _color = color
```

- For loop indexes must be explicitly initialized using i, j or k.

Accepted: for i in range(10)

```
    print(i)
```

Rejected: for _ in range(10)

```
    print("loop")
```

- Must use self. when referencing class attributes in instance methods.
 - ◆ Use “this” keyword when using other languages such as Java, C#

Accepted: class Test:

```
def __init__(self, value):
    self._value = value
```

Rejected: class Test:

```
def __init__(self, value):
    _value = value
```

Standard for Methods

- Method names must follow snake_case

Accepted: num_of_days

Rejected: numofdays

- Method size limit: Methods should not exceed 60 lines of code excluding comments and blank lines.
- Max Branches for a function/method body: less than or equal to 10.
- Max arguments per function/method is 5 parameters.
- Max number of returns per function/method is less than or equal to 5 returns.
- Logical Block Organization:
 - ◆ Use blank lines to separate related operations within a method.
 - ◆ Use descriptive comments to clarify the purpose of each logical section in the method.
- Indentation and Braces:
 - ◆ Use 2 spaces indentation for all code; do not use a tab.
 - ◆ For conditionals, loops and method calls, the opening “(“ and closing “)” parenthesis should be placed on the same line as the condition, loop or call. There should be no space between the parentheses and argument.

Accepted: method_name(argument)

Rejected: method_name(argument)

 - ◆ For conditionals, loops and method calls with *more than* one argument, arguments should be separated by a comma followed by a space.

Accepted: method_name (argument_one, argument_two, argument_three)

Rejected: method_name(argument_one argument_two,argument_three)

- Method definitions:

- ◆ If a method has *over four* parameters should be declared over multiple lines; parameters should be aligned vertically

Accepted:

```
def total_calc( argument_one,
                argument_two,
                argument_three,
                argument_four)
```

Rejected:

```
def total_calc( argument_one, argument_two, argument_three,
                argument_four)
```

- Each Method, Class or Function should include a description of its function as well as a description of its parameters above the definition/call. Comments should explain the “why” not the “what”.
- Triple - Quoted Function Docstrings are Mandatory below each function, describing what the function does

Example:

```
def multiply(x,y)
```

```
    """ multiplies two numbers together
    args:
    x (int): the first number
    y (int): the second number
    Returns:
    int: the product of x and y """
```

Standard for Exception Handling

- Input Validation should be included at the start of every method to ensure the parameters passed are valid.

Example:

```
def calculate(argument):
```

```
    if argument <= 10:
        raise ValueError("input must be greater than 10")
    #process if input is valid...
```

- Exceptions should be handled close to their source. A component should handle exceptions such as logging, retrying or recovering. Other exceptions that components are capable of handling are

invalid input, arithmetic errors and invalid data types. If a component cannot handle the exception in a meaningful way, it should propagate the exception to the main program.

Example: try: num = int(input("Enter an integer: "))

```
    output = 10/num
```

```
except ZeroDivisionError:
```

```
    print("cannot divide by zero")
```

```
except ValueError:
```

```
    print("please enter an integer")
```

→ Rules for dealing with Methods that have High Possibilities of Throwing Exceptions:

- ◆ Methods with high exception potential should include a try...except block.
- ◆ Use log and raise for each exception; logging records a history of errors which is beneficial for debugging and prevents the loss of information. the term 'raise' stops execution immediately and lets the caller (higher-level code) handle it.
- ◆ Just under the method definition, clearly indicate which exceptions may be thrown.
- ◆ Always use specific exception types to make error handling more precise.
- ◆ The NEW logging format should be utilized using f-strings or {} formatting.

Example: def read(file)

```
#This method opens and reads a file
```

```
#The following exceptions may be thrown:
```

```
#      FileNotFoundError: if the file_path cannot be found
```

```
#      IOError: an issue with the device that prevents communication with Windows
```

```
try:
```

```
    file = open(file_path, 'r')
```

```
    content = file.read()
```

```
    file.close()
```

```
    return content
```

```
except FileNotFoundError as e:
```

```
    print("file not found")
```

```
    logging.error(f"file not found: {e}")
```

```
    raise
```

```
except IOError as e:
```

```
    print(f"IO error details: {e}")
```

```
    logging.error(f"IO error: {e}")
```

raise

Miscellaneous

- Maximum number of characters on a single line is less than or equal to 150 characters.
- “type-check” should be enabled to verify type-correctness within the code; “type-check” detects issues such as invalid assignments, using objects incorrectly, etc.