Python Programming Design Guide for CMSC 1203

Introduction

This guide serves as a resource for CMSC 1203 Structured Programming Logic. Following these guidelines will not only improve the readability of your code but also make it easier to work on group projects, and allow instructors to better understand your work for more effective grading and feedback.

This guide provides rules, good and bad examples, and additional tips to guide you in writing clean, efficient, and easy-to-understand code. Although some of these practices may seem trivial at first glance, following them consistently can greatly enhance your coding skills and prepare you for more advanced courses and real-world projects.

Please note that violating the guidelines may result in a reduction of your assignment grades, so it's crucial to familiarize yourself with the following best practices. Your instructors and future self will thank you for the disciplined coding habits you develop today.

1. Variable Naming and Data Types

Rules:

- Descriptiveness: Variables must have descriptive names. For example, use age instead of a, student_name instead of sn.
- Case Style: Use snake_case for variable names. Words should be separated by an underscore (_).
    - Correct: my_age
    - Incorrect: myAge, MyAge, m

Good:

```
student_name = "Alice"
total_score = 95
```

Bad:

```
s = "Alice"
t = 95
```

2. File and Class Naming

Rules:

- File Names: Use snake_case for file names and separate words by an underscore (_).
    - Correct: hello_world.py
    - Incorrect: HelloWorld.py

- Class Names: Use snake_case for class names.
    - Correct: my_class
    - Incorrect: MyClass

Examples:

- File: input_validation.py
- Class: input_parser

3. Methods and Functions

Rules:

- Descriptive Action Words: Names should start with a verb to indicate the action.
- Case Style: Use snake_case for method and function names.

Good:

```
def calculate_sum(a, b):
    return a + b
```

Bad:

```
def sum(a, b):
    return a + b
```

4. Constants

Rules:

- All Caps: Constants should be in ALL CAPS.
- Underscore Separation: Use an underscore to separate words as needed.

Good:

```
MAX_LIMIT = 100
```

Bad:

```
MaxLimit = 100
```

5. Indentation and Whitespace

Rules:

- Use 4 spaces for each indentation level.
- Whitespace around Operators: Use single spaces around operators and after commas to improve readability.

Importance:

Correct indentation and proper use of whitespace are not just a matter of aesthetics. In Python, indentation is syntactically significant, and failing to indent your code correctly will result in errors. Whitespace around operators and after commas improves code readability.

Correct Indentation:

```
def calculate_sum(a, b):
    if a > 0 and b > 0:
        return a + b
    else:
        print("Input numbers must be positive.")
```

Incorrect Indentation:

```
def calculate_sum(a,b):
if a > 0 and b > 0:
return a+b
else:
print("Input numbers must be positive.")
```

Whitespace around Operators:

```
# Correct
total = a + b

# Incorrect
total=a+b
```

Whitespace after Commas:

```
# Correct
my_list = [1, 2, 3, 4]

# Incorrect
my_list = [1,2,3,4]
```

6. Commenting

Rules:

- Use both documentation and implementation comments.
- Avoid excessive commenting.
- Comments are mandatory in this class.

Types of Comments:

- Documentation Comments: These should allow someone to use your code without having to read the source code.

Block Comment:

```
#
# Here is a block comment.
#
```

Single Line Comment:

```
# This is a single line comment explaining the following code.
```

End-of-Line Comment:

```
if a == 2:
    return True  # special case
else:
    return is_prime(a)  # works only for odd a
```

Example of Good Commenting:

```
# Holds item prices, subtotal, tax, and total
item1 = 0.0
item2 = 0.0
subtotal = 0.0
tax = 0.0
total = 0.0

# Constant for sales tax rate
TAX_RATE = 0.07

# User input for item prices
item1 = float(input('Enter the price of item #1: '))
item2 = float(input('Enter the price of item #2: '))

# Calculations
subtotal = item1 + item2
tax = subtotal * TAX_RATE
total = subtotal + tax

# Display results
print(f'Subtotal: {subtotal:.2f}')
print(f'Sales Tax: {tax:.2f}')
print(f'Total: {total:.2f}')
```

Example of Bad Commenting:

```
# here we have some variables
item1 = 0.0 #variable 1
item2 = 0.0 #variable 2
# ...
subtotal = 0.0 #subtotal variable
tax = 0.0 #tax
total = 0.0 #adding everything
TAX_RATE = 0.07  # this is the tax rate

# now let's get the price of each item
item1 = float(input('Enter the price of item #1: '))
# ...
# now let's add them up
subtotal = item1 + item2
# let's find the tax
tax = subtotal * TAX_RATE
# final total
total = subtotal + tax
# let's print everything
print(f'Subtotal: {subtotal:.2f}')
print(f'Sales Tax: {tax:.2f}')
print(f'Total: {total:.2f}')
```