Unclassified

Python Basics

Python syntax and structure:

- Python syntax is whitespace driven.
- Indentation defines code blocks. Any amount of indentation may be used, however the levels must match as expected or an indentation error will be thrown.
- For example, defining a function or using a flow control statement will require an indentation level for the related block.
    - The following line defines a function that will print hello world. Statements ending in a semi-colon (😃 will expect an indentation level. The following function has a function signature, or a definition, stating it takes no arguments and returns nothing.

```
def print_hello_world():
    print("Hello World!")
```

    - To provide a more personalized message, we will recreate the function to take an argument.

```
def print_hello_world(name:str) -> None:
    print(f"Hello {name}!")
```

    - The new function takes a string and returns nothing, but will print the given name.

Variables and data types (int, float, str, bool)

Primitive Types

1. Integer (int): It represents whole numbers without any fractional part, like 1, 10, or -5. Integers are used to perform mathematical operations like addition, subtraction, and multiplication.
2. Float: It represents numbers with a decimal point, like 3.14 or -2.5. Floats are used when you need to work with numbers that can have a fractional part, such as in calculations involving measurements or scientific values.
3. String (str): It represents a sequence of characters enclosed in single quotes ('') or double quotes (""). Strings are used to store text or any combination of letters, numbers, symbols, or spaces. For example, "Hello, World!" or "12345".
4. Boolean (bool): It represents one of two values: True or False. Booleans are used to express logical states or conditions. They are often used in conditional statements and comparisons. True indicates a positive or affirmative condition, and False indicates a negative or false condition.

```
# Variable declaration and assignment
name = "John"
age = 25
pi = 3.14
is_student = True

# Printing the variables
print("Name:", name)
print("Age:", age)
print("Pi:", pi)
print("Is student?", is_student)
```

- The variable `name` is declared and assigned the string value "John".
- The variable `age` is declared and assigned the integer value 25.
- The variable `pi` is declared and assigned the float value 3.14.
- The variable `is_student` is declared and assigned the boolean value True.

After assigning values to the variables, we can use the `print()` function to display their values on the console. In this example, we print the values of all the variables.

When you run this program, the output will be:

```
Name: John
Age: 25
Pi: 3.14
Is student? True
```

Variable Naming and Conventions

Python follows the PEP 8 style guide, which provides guidelines for writing clean and readable code. Here are some key rules and conventions for variable naming:

1. Variable Names:
    - Use lowercase letters for variable names.
    - Separate words with underscores (_). This is called "snake_case". For example, my_variable, first_name, or item_count.
2. Avoid Reserved Keywords:
    - Do not use Python reserved keywords (e.g., if, while, for, def, class, etc.) as variable names.
3. Clarity and Readability:
    - Choose descriptive and meaningful names that reflect the purpose or content of the variable.
    - Avoid single-character names unless they represent a well-known convention (e.g., i for loop counters).
4. Constants:
    - Use ALL_CAPS for constants, indicating that their values should not be changed.
    - Separate words with underscores. For example, MAX_VALUE, PI, or TOTAL_COUNT.
5. Avoid Ambiguous Names:
    - Use names that are specific and unambiguous to avoid confusion.

- Choose names that accurately describe the variable's purpose or data it represents.

6. Avoid Leading Underscores:
   - A single leading underscore (_) indicates that a variable or method is intended for internal use within a class or module. It is a convention to indicate that the identifier is private, although it is still accessible.
7. Use Proper Spacing and Alignment:
   - Use spaces around operators (e.g., =, +, -, *, /, etc.) for clarity.
   - Maintain consistent indentation (typically four spaces) to indicate code blocks and improve readability.
   - Following these naming conventions and guidelines helps make your code more consistent and readable, allowing others to understand and collaborate on your code more easily. Adhering to PEP 8 standards is considered a best practice in the Python community.

Basic operations: arithmetic, string concatenation, and comparison:

```
# Arithmetic operations
num1 = 10
num2 = 3

# Addition
sum_result = num1 + num2
print("Sum:", sum_result)

# Subtraction
difference = num1 - num2
print("Difference:", difference)

# Multiplication
product = num1 * num2
print("Product:", product)

# Division
quotient = num1 / num2
print("Quotient:", quotient)

# Modulo (Remainder)
remainder = num1 % num2
print("Remainder:", remainder)

# Exponentiation
exponent = num1 ** num2
print("Exponent:", exponent)
```

In the example above:

- The variables num1 and num2 are assigned the values 10 and 3, respectively.
- The addition operation num1 + num2 calculates the sum and assigns it to the variable sum_result.
- The subtraction operation num1 - num2 calculates the difference and assigns it to the variable difference.

- The multiplication operation num1 * num2 calculates the product and assigns it to the variable product.
- The division operation num1 / num2 calculates the quotient and assigns it to the variable quotient.
- The modulo operation num1 % num2 calculates the remainder and assigns it to the variable remainder.
- The exponentiation operation num1 ** num2 raises num1 to the power of num2 and assigns it to the variable exponent.

```
Sum: 13
Difference: 7
Product: 30
Quotient: 3.3333333333333335
Remainder: 1
Exponent: 1000
```

Unclassified