Day 1&2: Python Basics Documentation

1. Variables

Variables are containers used to store data values. In Python, you don't need to declare the type of a variable explicitly; Python determines the type automatically.

2. Strings

Strings are sequences of characters enclosed in single or double quotes.

String Operations:

- Concatenation: `"Hello" + " " + "World!"`
- Slicing: `greeting[0:5]` (gives "Hello")
- Length: `len(greeting)` (gives 13)

3. If, Elif, Else (Conditional Statements)

Conditional statements allow you to make decisions in your program based on conditions.

If: Runs the code if the condition is `True`.

Elif: Tests another condition if the previous one was `False`.

Else: Runs when all previous conditions were `False`.

4. For Loop

A `for` loop allows you to iterate over a sequence (e.g., list, range, string).

```
for i in range(5):
    print(i)
```

5. While Loop

A `while` loop runs as long as a given condition is `True`.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

6. Use Case Method

A use case describes how a user interacts with a system to achieve a goal. In programming, we use methods (functions) to handle specific tasks. Here's an example use case:

- **Use Case**: Calculate the area of a rectangle.

1. Input the length and width.
2. Compute the area using the formula `Area = length * width`.
3. Return the result.

7. Functions

A function is a block of code that only runs when it is called. You can pass data (parameters) into a function, and it can return data.

Parameters: Values passed into the function.

Return: The value that the function outputs.

8. Methods of Functions

Functions can have various methods or types of return behaviors.

1. **Positional Arguments**:
   The order of arguments matters when calling a function.
2. **Keyword Arguments**:
   Arguments can be passed by name, not by position.
3. **Default Arguments**:
   If no argument is provided, a default value is used.
4. **Returning Multiple Values**:
   A function can return multiple values, typically in a tuple.

Day 3: Advanced Python Concepts Documentation

1. List

A list is an ordered collection of items that can be of different types (integers, strings, etc.). Lists are mutable, meaning their contents can be modified.

Key Features:

- Ordered: The order of elements is preserved.
- Mutable: You can change the elements after creation.

Common List Methods:

`.append()`: Adds an element to the end of the list.

`.insert()`: Inserts an element at a specific index.

`.remove()`: Removes the first occurrence of a value.

`.pop()`: Removes and returns the last element (or element at a specific index).

`.sort()`: Sorts the list.

`.reverse()`: Reverses the list.

2. Tuple

A **tuple** is similar to a list but immutable. Once created, its elements cannot be changed. Tuples are typically used for fixed collections of items.

Key Features:

- Ordered: The order of elements is preserved.
- Immutable: You cannot modify, add, or remove elements.

Common Tuple Operations:

- **Indexing**: `coordinates[0]` would return `10`.
- **Slicing**: `coordinates[0:1]` would return `(10,)`.

3. Set

A **set** is an unordered collection of unique elements. Sets do not allow duplicate values.

Key Features:

- Unordered: No specific order to the elements.
- Mutable: You can add or remove items, but the elements must be unique.

Common Set Methods:

- `.add()`: Adds a single element to the set.
- `.remove()`: Removes a specific element. Raises an error if the element doesn't exist.
- `.discard()`: Removes an element without raising an error if it doesn't exist.
- `.union()`: Returns a set containing all unique elements from two sets.
- `.intersection()`: Returns a set containing only the elements found in both sets.

4. Dictionary

A **dictionary** is a collection of key-value pairs. Dictionaries are ordered, mutable, and indexed by keys.

Key Features:

- Unordered: No guarantee of order of elements.
- Mutable: You can add, modify, or remove key-value pairs.
- Keys are unique; values can be duplicates.

Common Dictionary Methods:

- `.get()`: Returns the value for a specified key. If the key doesn't exist, it returns `None` (or a default value if specified).
- `.keys()`: Returns a view object displaying all the dictionary's keys.
- `.values()`: Returns a view object displaying all the dictionary's values.
- `.items()`: Returns a view object displaying all key-value pairs.
- `.update()`: Updates the dictionary with new key-value pairs.
- `.pop()`: Removes a key-value pair and returns its value.

5. F-String (Formatted String)

An **f-string** is a way to embed expressions inside string literals using curly braces `{}`. It was introduced in Python 3.6 and makes string formatting more readable.

Benefits:

- More concise and readable than `str.format()` or concatenation.
- Supports complex expressions inside the curly braces.

6. Docstring

A **docstring** is a string that describes the purpose or behavior of a function, class, or module. It is placed at the beginning of the function or class definition.

> Docstrings are important for documentation and are accessible via the `help()` function or the `__doc__` attribute.
>
> **PEP 257** outlines conventions for docstrings, including using triple quotes and starting with a one-line summary of the function's purpose.

7. PEP 8

PEP 8 is the Python Enhancement Proposal that provides guidelines for writing clean, readable, and consistent Python code. It covers naming conventions, indentation, and best practices for writing Python code.

PEP 8 ensures your code remains readable and maintainable for both you and other developers.

Day 4: Object-Oriented Programming (OOP) Concepts

1. Abstraction

Definition:
Abstraction is the concept of hiding the internal workings of a system and exposing only the essential features to the user. In OOP, abstraction is achieved by using abstract classes and methods, allowing you to define a common interface without specifying the exact behavior.

Purpose:

- To simplify complex systems by breaking them into manageable parts.
- To define common functionality that subclasses can implement in their own way.

In Python:

- Abstraction is typically implemented using the `abc` module (Abstract Base Classes).
- Abstract methods in an abstract class must be overridden by any subclass.

---

2. Inheritance

Definition:
Inheritance is the process by which a class (child class) acquires the properties and behaviors (methods) of another class (parent class). This allows for code reuse and a hierarchical relationship between classes.

Purpose:

- To promote code reusability and reduce redundancy.
- To model real-world relationships in a more intuitive way (e.g., a `Dog` is an `Animal`).

In Python:

- A subclass inherits methods and attributes from a parent class using the `class SubclassName(ParentClass)` syntax.

- The `super()` function is often used to invoke the parent class's methods, particularly in initialization (`__init__`).

---

3. Encapsulation

Definition:
Encapsulation is the concept of bundling data (attributes) and the methods that operate on the data into a single unit or class. It also restricts access to some of the object's internal components, which is known as **data hiding**.

Purpose:

- To protect an object's state from unauthorized or unintended modification.

- To ensure that an object's data is modified only through well-defined interfaces (methods).

In Python:

- **Private** attributes and methods (indicated with a `__` prefix) are used to hide implementation details.

- **Public** methods provide controlled access to private data.

---

4. Polymorphism

Definition:
Polymorphism refers to the ability of different classes to provide a method with the same name but implement it differently. It allows objects of different types to be treated as if they were instances of the same class through a common interface.

Purpose:

- To allow different objects to be treated in the same way, enhancing flexibility and scalability in a program.

- To enable method overriding and dynamic behavior based on the object type.

In Python:

- Method overriding allows a subclass to provide a specific implementation of a method that is already defined in its parent class.
- Polymorphism is typically achieved through inheritance, where a parent class defines an interface (method), and subclasses provide specific behavior for that method.

Day 5: Logic Building in Python

1. Check Positive, Negative, or Zero

Description:
This logic takes a number as input and checks whether it's positive, negative, or zero.

Steps:

1. Take user input for a number.
2. Use an `if`, `elif`, and `else` structure to check:
    - If the number is greater than 0 (positive).
    - If the number is 0 (zero).
    - Otherwise, the number is negative.

2. Find the Smallest of Three Numbers

Description:
This logic takes three numbers as input and prints the smallest of the three.

Steps:

1. Take input for three numbers from the user.
2. Use multiple `if` conditions to check which number is the smallest.
3. If two or more numbers are equal, print that they are equal.

3. Check if Number is Divisible by Both 3 and 5

Description:
This logic checks if the input number is divisible by both 3 and 5, only 3, only 5, or neither.

Steps:

1. Take input for a number.
2. Use `if`, `elif` conditions to check:
    - If the number is divisible by both 3 and 5.
    - If it's divisible by only 3.
    - If it's divisible by only 5.
    - Otherwise, it's not divisible by either.

4. Print All Even Numbers from 1 to N

Description:
This logic prints all even numbers from 1 to a number `N` entered by the user.

Steps:

1. Take input for the number `N`.
2. Use a `for` loop to iterate through numbers from 1 to `N`.
3. Use the modulus operator `%` to check if a number is even.
4. Print the even numbers.

5. Count Vowels in a String

Description:
This logic counts the number of vowels in a given string (word or sentence).

Steps:

1. Take input for a string (word or sentence).
2. Use a loop to iterate over each character in the string.
3. Check if the character is a vowel (`a`, `e`, `i`, `o`, `u`).
4. Increment the count if the character is a vowel.
5. Print the total count of vowels.

6. Reverse a String Without Using [::-1]

Description:
This logic reverses a string without using Python's slicing feature `[::-1]`.

Steps:

1. Take input for a string.
2. Use a `for` loop to reverse the string:
    - Append each character to the front of a new string.
3. Print the reversed string.

7. Count Digits, Letters, and Spaces in a String

Description:
This logic counts the number of letters, digits, and spaces in a string.

Steps:

1. Take input for a string (sentence).
2. Initialize counters for letters, digits, and spaces.
3. Use a loop to check each character:
    - If it’s a letter, increment the letter counter.
    - If it’s a digit, increment the digit counter.
    - If it’s a space, increment the space counter.
4. Print the counts of letters, digits, and spaces.