Unit 2: Python Programming

A comprehensive and detailed Q&A guide from fundamentals to advanced concepts.

2.1 Introduction to Python

Why is Python considered a good language for beginners?

Python's straightforward syntax and clear structure make it an excellent choice for beginners, allowing them to focus on learning programming concepts rather than dealing with complex syntax rules.

What are the four basic steps of the programming life cycle?

1. Write Code: Create instructions in a programming language.
2. Compile/Interpret: Translate the code into a form the computer can understand.
3. Execute: Run the code to perform the task.
4. Output: Display results or perform actions based on the code.

During Python installation, why is it recommended to check "Add Python to PATH"?

Checking "Add Python to PATH" makes it easier to run Python scripts from any directory in the command line or terminal without having to specify the full path to the Python executable.

2.2 Basic Syntax & Structure

What is the purpose of comments in code, and how are they created?

Comments are used to provide explanations or notes within the code for human readers. They are ignored by the Python interpreter. Single-line comments start with `#`, and multi-line comments are enclosed in triple quotes (`"""..."""`).

What is "snake_case" and why is it a good practice in Python?

"Snake_case" is a naming convention where words in a variable name are separated by underscores (e.g., `student_name`). It is a good practice in Python because it improves code readability, making multi-word variable names easy to understand.

How does Python handle different data types in the `print()` function?

The `print()` function can accept multiple arguments of different types. It automatically converts non-string types (like integers or floats) to strings before displaying them. You can separate arguments with commas.

```
age = 30
print("The age is:", age) # Python converts 'age' to a string for printing
```

2.3 Operators and Expressions

Explain the difference between regular division (`/`), floor division (`//`), and modulus (`%`).

- **Regular Division (`/`):** Performs standard division and always returns a float. Example: `10 / 3` results in `3.333...`
- **Floor Division (`//`):** Performs division and rounds the result down to the nearest whole number (integer). Example: `10 // 3` results in `3`.
- **Modulus (`%`):** Returns the remainder of a division. Example: `10 % 3` results in `1`.

What type of value do comparison operators return?

Comparison operators (like `>`, `<`, `==`, `!=`) always return a Boolean value: either `True` or `False`, based on the result of the comparison.

Explain the order of precedence for arithmetic operators in Python.

The order of precedence from highest to lowest is:

1. **Parentheses `()`:** Operations inside parentheses are always evaluated first.
2. **Exponentiation `**`:** Power operations.
3. **Multiplication `*`, Division `/`, Floor Division `//`, Modulus `%`:** These have equal precedence and are evaluated from left to right.
4. **Addition `+` and Subtraction `-`:** These have the lowest precedence and are also evaluated from left to right.

2.4 Control Structures

What is a "shorthand if-else" statement (ternary operator)?

It is a compact, single-line way to write an if-else statement. The syntax is: `value_if_true if condition else value_if_false`.

```
# Example:
result = "Pass" if score >= 50 else "Fail"
print(result)
```

When would you use a nested conditional statement?

A nested conditional is used when you need to check a secondary condition only if a primary condition is met. For example, you might first check if the weather is "rainy," and only if that is true, you then check if the temperature is below 15 degrees to decide between an umbrella and a raincoat.

Explain the three arguments of the `range()` function.

The `range()` function can take up to three arguments: `range(start, stop, step)`.

- **start:** The first number in the sequence (inclusive). Defaults to 0 if not provided.
- **stop:** The number to stop before (exclusive). This argument is mandatory.
- **step:** The increment between each number in the sequence. Defaults to 1 if not provided.

2.5-2.6 Functions, Modules, and Data Structures

What is the difference between defining a function and invoking it?

Defining a function means creating it using the `def` keyword. This sets up the function's name, parameters, and the code it will execute, but it doesn't run the code.
Invoking (or calling) a function means actually executing the code inside it by writing the function's name followed by parentheses, e.g., `my_function()`.

Explain how default parameters work in a function.

Default parameters are given a value when the function is defined. If a value for that parameter is not provided when the function is called, the default value is used automatically.

```
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet("Alice")  # Output: Hello, Alice!
greet()         # Output: Hello, Guest! (uses default)
```

Why is it beneficial to use keyword arguments when calling a function?

Keyword arguments (e.g., `function(name="Alice", age=30)`) improve code readability and clarity. They make it explicit which value is being assigned to which parameter, and they allow you to pass arguments in any order.

What is the module search path in Python?

When you import a module, Python searches for it in a specific order:

1. **Current Directory:** The directory where the main script is running.
2. **Standard Library:** The directories where Python's built-in modules are installed.
3. **Third-Party Libraries:** Directories where packages installed via tools like `pip` are located.

Explain why one might choose to use a tuple over a list.

A tuple might be chosen over a list for two main reasons:

- **Immutability:** Because tuples cannot be changed, they are useful for storing data that should not be modified, ensuring data integrity.
- **Performance:** Tuples can be slightly faster than lists for iteration because of their fixed size. They can also be used as keys in a dictionary, whereas lists cannot.

Explain list slicing with both positive and negative indices.

Slicing creates a new list from a portion of an existing one. **Positive indices** count from the beginning (0, 1, 2...). **Negative indices** count from the end (-1, -2, -3...).

```
fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"]

  # Positive Slicing: from index 1 up to (not including) 4
 print(fruits[1:4])  # Output: ['Banana', 'Cherry', 'Date']

# Negative Slicing: from the 4th-to-last item up to (not including) the last
                                    item
 print(fruits[-4:-1]) # Output: ['Banana', 'Cherry', 'Date']
```

2.7-2.8 Object-Oriented Programming (OOP)

What is the role of the `self` parameter in a class method?

The `self` parameter is a reference to the current instance (object) of the class. It is automatically passed as the first argument to instance methods and is used to access the object's attributes and other methods. For example, `self.color` accesses the `color` attribute of that specific object.

Describe the three access modifiers in Python (Public, Protected, Private).

- **Public:** Attributes are accessible from anywhere, both inside and outside the class. This is the default. (e.g., `self.name`)
- **Protected:** Attributes are intended for internal use within the class and its subclasses. They are prefixed with a single underscore. (e.g., `self._speed`)
- **Private:** Attributes are intended to be accessible only from within the class itself. They are prefixed with a double underscore, which triggers name mangling. (e.g., `self.__serial_number`)

Explain the difference between Association and Composition in OOP.

Both describe relationships between classes, but they differ in ownership and lifetime.

- **Association:** A "uses-a" relationship where objects have their own lifecycles. For example, a `Professor` and a `Student` can exist independently.
- **Composition:** A "part-of" relationship where the "part" object cannot exist without the "whole" object. For example, a `Room` is part of a `House`; if the `House` is destroyed, the `Room` is too.

How does Polymorphism work in Python?

Polymorphism (meaning "many forms") allows objects of different classes to be treated as objects of a common superclass. It means different classes can have methods with the same name, and Python will call the correct one based on the object's actual type. This allows for writing flexible and generic code.

```
    # Dog and Cat both have a make_sound() method
animal_sound(my_dog_object) # Calls the Dog's version
animal_sound(my_cat_object) # Calls the Cat's version
```

2.9-2.10 Advanced Concepts & Testing

What is the purpose of the `else` block in a `try...except` statement?

The `else` block in a `try...except` statement contains code that will be executed only if **no exceptions** were raised in the `try` block. It is useful for separating the code that might raise an error from the code that should run upon success.

Why is it beneficial to use the `with` statement when working with files?

The `with` statement is highly recommended for file handling because it ensures that the file is automatically and properly closed after the block of code is executed, even if an error occurs within the block. This prevents resource leaks.

Describe the main file modes for opening a file in Python.

- `'r'` **(Read):** Opens a file for reading only. The file must exist. This is the default mode.
- `'w'` **(Write):** Opens a file for writing. If the file exists, it is truncated (emptied). If it does not exist, it is created.
- `'a'` **(Append):** Opens a file for appending. New data is added to the end. If the file does not exist, it is created.
- `'r+'` **(Read and Write):** Opens a file for both reading and writing. The file must exist.

What are the different types of testing mentioned in the document?

- **Unit Testing:** Tests individual parts of the code (e.g., functions, classes) in isolation.
- **Integration Testing:** Checks how different parts of the code work together.
- **Functional Testing:** Validates that the software behaves as expected from the user's perspective.
- **Regression Testing:** Ensures that new changes or bug fixes do not break existing functionality.