Module 6) Python Fundamentals

Introduction to Python

Question-1: Introduction to Python and its Features (simple, high-level, interpreted language).

Answer: Python is a high-level, interpreted programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages due to its versatility and ease of learning.

Features in Python

In this section we will see what are the features of Python programming language:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword. Download Python Since it is open-source, this means that source code is also available to the public. So, you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in Python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Large Community Support

Python has gained popularity over the years. Our questions are constantly answered by the enormous Stack Overflow community. These websites have already provided answers to many questions about Python, so Python users can consult them as needed.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for Windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an integrated language

Python is also an integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called **bytecode**.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.x

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like

javascript. Backend is the strong forte of Python it’s extensively used for this work cause of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

In Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

Question 2: History and evolution of Python.

Answer: Python is a widely used general-purpose, high-level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed to emphasize code readability, and its syntax allows programmers to express concepts in fewer lines of code.

In the late 1980s, history was about to be written. It was that time when working on Python started. Soon after that, Guido Van Rossum began doing its application-based work in December of 1989 at Centrum Wiskunde & Informatica (CWI) which is situated in the Netherlands. It was started as a hobby project because he was looking for an interesting project to keep him occupied during Christmas.

The programming language in which Python is said to have succeeded is ABC Programming Language, which had interfacing with the Amoeba Operating System and had the feature of exception handling. He had already helped create ABC earlier in his career and had seen some issues with ABC but liked most of the features. After that what he did was very clever. He had taken the syntax of ABC, and some of its good features. It came with a lot of complaints too, so he fixed those issues completely and created a good scripting language that had removed all the flaws.

Question 3: **Advantages of using Python over other programming languages.**

Answer:

1. Python is Simple to Learn and Use
2. Perfect for Rapid Deployment
3. Python has High-Performance
4. Strong Support for Data Analysis and Scientific Computing
5. User-Friendly Built-in Data Types
6. Python Needs Less Coding
7. Python Has OS Portability
8. Python is Flexible in Deployment
9. Support from Renowned Corporate Sponsors
10. Hundreds of Community Driven Python Libraries and Frameworks

Question 4: Installing Python and setting up the development environment (Anaconda, PyCharm, or VS Code).

Answer:

To install Python and set up a development environment, you can use Anaconda, PyCharm, or Visual Studio Code (VS Code):

Python:

You can download the latest version of Python from Python.org. The installer is available for Windows, macOS, Linux/Unix, and other platforms.

Anaconda:

You can download the installer for Anaconda from the Anaconda website. Anaconda is available for Windows, macOS, and Linux. It comes with pre-installed packages and software, but it can be more resource-intensive than a lean Python installation.

VS Code:

You can download the installer for VS Code from code.visualstudio.com. You can install the Python extension from the VS Code Marketplace. You can also use the Python profile template to get started with Python development.

Here are some tips for setting up Python and the development environment:

Set a Python interpreter in VS Code

You can use the Python: Select Interpreter command:

1. Open the Command Palette (Ctrl+Shift+P)
2. Select the Python: Select Interpreter command
3. Select the Enter interpreter path... option
4. Enter the full path of the Python interpreter

Set up PyCharm with an Anaconda virtual environment

You can:

1. Open Configure > Settings
2. Search for “Project Interpreter”
3. Click on Add local
4. Select “conda environment”
5. Click on “Existing environment” and navigate to the environment that you want to use
6. Select the bin/python file inside the conda environment

Click the “Make available to all projects” if you want the interpreter to be used by multiple projects

7. How to setup PyCharm with an anaconda virtual environment .

Question 5: Writing and executing your first Python program

Answer: Installation of Python

- Download the current production version of Python (2.7.1) from the Python Download site.
- Double click on the icon of the file that you just downloaded.
- Accept the default options given to you until you get to the *Finish* button. Your installation is complete.

Setting up the Environment

- Starting at *My Computer* go to the following directory C:\Python27. In that folder you should see all the Python files.
- Copy that address starting with *C:* and ending with *27* and close that window.
- Click on *Start*. Right Click on *My Computer*.
- Click on *Properties*. Click on *Advanced System Settings* or *Advanced*.
- Click on *Environment Variables*.
- Under *System Variables* search for the variable *Path*.
- Select *Path* by clicking on it. Click on *Edit*.
- Scroll all the way to the right of the field called *Variable value* using the right arrow.
- Add a semi-colon (;) to the end and paste the path (to the Python folder) that you previously copied. Click *OK*.

Writing Your First Python Program

- Create a folder called *PythonPrograms* on your *C:*\ drive. You will be storing all your Python programs in this folder.

- Go to *Start* and either type *Run* in the *Start Search* box at the bootom or click on *Run*.
- Type in *notepad* in the field called *Open*.
- In *Notepad* type in the following program exactly as written:

```
# File: Hello.py


print "Hello World!"
```

- Go to *File* and click on *Save as*.
- In the field *Save in* browse for the *C:* drive and then select the folder *PythonPrograms*.
- For the field *File name* remove everything that is there and type in *Hello.py*.
- In the field *Save as type* select *All Files*
- Click on *Save*. You have just created your first Python program.

Running Your First Program

- Go to *Start* and click on *Run*.
- Type *cmd* in the *Open* field and click OK.
- A dark window will appear. Type *cd C:* and hit the key Enter.
- If you type *dir* you will get a listing of all folders in your *C:* drive. You should see the folder *PythonPrograms* that you created.
- Type *cd PythonPrograms* and hit Enter. It should take you to the PythonPrograms folder.
- Type *dir* and you should see the file *Hello.py*.

- To run the program, type *python Hello.py* and hit Enter.
- You should see the line *Hello World!*
- Congratulations, you have run your first Python program.

Getting Started with Python Programming for Mac Users

Python comes bundled with Mac OS X. But the version that you have is quite likely an older version. Download the latest binary version of Python that runs on both Power PC and Intel systems and install it on your system.

Writing Your First Python Program

- Click on *File* and then *New Finder Window*.
- Click on *Documents*.
- Click on *File* and then *New Folder*.
- Call the folder *PythonPrograms*. You will be storing all class related programs there.
- Click on *Applications* and then *TextEdit*.
- Click on *TextEdit* on the menu bar and select *Preferences*.
- Select *Plain Text*.
- In the empty TextEdit window type in the following program, exactly as given:

```
# File: Hello.py


print "Hello World!"
```

- From the *File* menu in TextEdit click on *Save As*.
- In the field *Save As:* type *Hello.py*.

- Select *Documents* and the file folder *PythonPrograms*.
- Click *Save*.

Running Your First Program

- Select *Applications*, then *Utilities* and *Terminal*.
- In your *Terminal* window type *ls* and Return. It should give a listing of all the top level folders. You should see the *Documents* folder.
- Type *cd Documents* and hit Return.
- Type *ls* and hit Return and you should see the folder *PythonPrograms*.
- Type *cd PythonPrograms* and hit Return.
- Type *ls* and hit return and you should see the file *Hello.py*.
- To run the program, type *python Hello.py* and hit Return.
- You should see the line *Hello World!*
- Congratulations, you have run your first Python program.

Starting IDLE on Mac

- In a *Terminal* window, type *python*. This will start the Python shell. The prompt for that is *>>>*
- At the Python shell prompt type *import idlelib.idle*
- This will start the IDLE IDE

Using IDLE on either Windows or Mac

- Start IDLE
- Go to File menu and click on *New Window*
- Type your program in

- Go to File menu and click on Save. Type in *filename.py* This will save it as a plain text file, which can be opened in in any editor you choose (like Notepad or TextEdit).
- To run your program go to *Run* and click *Run Module*

2. Programming Style

Question 1: Understanding Python's PEP 8 guidelines.

Answer: This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

he Python Enhancement Proposal (PEP) 8 style guide provides guidelines for writing Python code that is consistent and easy to read. Here are some of the PEP 8 guidelines:

- **Indentation**: Use four spaces for indentation and avoid tabs.
- **Line length**: Limit lines to 79 characters for code and 72 for docstrings and comments.
- **Imports**: Import one module per line, in alphabetical order.
- **Whitespace**: Use spaces around operators and after commas, but not directly inside parentheses or brackets.
- **Comments**: Use regular and updated comments that form complete sentences.

- **Naming conventions**: Use lowercase with underscores (snake_case) for variables and functions, and CamelCase for classes.
- **Constants**: Write constants in all capital letters with underscores separating words.
- **Docstrings**: Use single or multi-line docstrings to define a program or function.
- **Wrapping lines**: Use parenthesis, brackets, and braces to wrap lines, rather than backslashes.
- **Blank lines**: Use blank lines to separate logical sections and functions.

Question 2: Indentation, comments, and naming conventions in Python.

Answer:

Indentation

Indentation is used to define blocks of code, such as loops, conditionals, and function bodies. Here are some indentation rules:

- Use spaces for indentation, not tabs.
- Use four spaces by default for each indentation level.
- Maintain consistent indentation throughout your code.
- Do not indent the first line of Python code.
- Use a colon (:) to indicate the start of an indented block.

Comments

Comments are lines of text in a program that are ignored by the interpreter. There are three types of comments in Python: single-line, multi-line, and documentation comments. To add a single-line comment, put the # symbol at the beginning of your line.

Naming conventions

Here are some naming conventions for Python:

- Use lowercase for methods.
- Use CapWords for type variables.
- Use UPPERCASE for constants.
- Use lowercase for package.
- When using acronyms in CapWords, capitalize all the letters of the acronym.

Question 3: Writing readable and maintainable code.

Answer:

1. Use Meaningful Variable Name

Choose descriptive names for variables, functions, classes, and modules. This helps make the code self-documenting and easier to understand.

2. Follow PEP 8 Guidelines:

Adhere to the Python Enhancement Proposal (PEP) 8 guidelines, which define the coding style for Python. Consistency in formatting and naming conventions is crucial for clean code.

3. Limit Line Length:

Keep lines of code reasonably short (recommended: 79 characters or less) to improve readability and avoid horizontal scrolling.

4. Proper Indentation:

Use consistent indentation (typically four spaces) to enhance code structure and make it more readable.

5. Add Comments:

Include comments to explain complex logic, tricky parts, or any important information. Avoid excessive comments on obvious code.

6. Modularize Code:

Break down your code into smaller functions or classes, each with a single responsibility. This promotes reusability and maintainability.

7. Avoid Global Variables:

Minimize the use of global variables as they can lead to unintended side effects and make the code harder to reason about.

3. Core Python Concepts

Question-1: Understanding data types: integers, floats, strings, lists, tuples, dictionaries, sets.

Answer:

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either True or False
Set	set, frozenset	hold collection of unique items

Question-2: Python variables and memory allocation.

Answer: In Python, variables are stored differently based on their type and scope either it can be stored in the heap or in the stack which are the two main memory regions utilized for a variable storage. Python, being a high-level programming language, abstracts away many low-level memory management details from the programmer. However, it's still essential to understand how variables are stored in Python to write efficient and optimized code.

Question-3: Python operators: arithmetic, comparison, logical, bitwise.

Answer:

Arithmetic Operators:

Python Arithmetic Operators are used on two operands to perform basic mathematical operators like addition, subtraction, multiplication, and division. There are different types of arithmetic operators available in Python including the '+' operator for addition, '-' operator for subtraction, '*' for multiplication, '/' for division, '%' for modulus, '**' for exponent and '//' for floor division.

Let us consider the following table of arithmetic operators for a detailed explanation.

S. No.	Operator	Syntax	Description
1	**+ (Addition)**	r = a + b	This operator is used to add two operands. For example, if a = 15, b = 10 => a + b = 15 + 10 = 25
2	**- (Subtraction)**	r = a - b	This operator is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 5 => a - b = 20 – 5 = 15
3	**/ (divide)**	r = a / b	This operator returns the quotient after dividing the first operand by the second operand. For example, if a = 15, b = 4 => a / b = 15 / 4 = 3.75
4	*** (Multiplication)**	r = a * b	This operator is used to multiply one operand with the other. For

			example, if a = 20, b = 4 => a * b = 20 * 4 = 80
5	**% (reminder)**	r = a % b	This operator returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a % b = 20 % 10 = 0
6	**** (Exponent)**	r = a ** b	As this operator calculates the first operand's power to the second operand, it is an exponent operator. For example, if a = 2, b = 3 => a**b = 2**3 = 2^3 = 2*2*2 = 8
7	**// (Floor division)**	r = a // b	This operator provides the quotient's floor value, which is obtained by dividing the two operands. For example, if a = 15, b = 4 => a // b = 15 // 4 = 3

Comparison Operators:

Python Comparison operators are mainly used for the purpose of comparing two values or variables (operands) and return a Boolean value as either True or False accordingly. There are various types of comparison operators available in Python including the '==', '!=', '<=', '>=', '<', and '>'.

Let us consider the following table of comparison operators for a detailed explanation.

S. No.	Operator	Syntax	Description
1	==	a == b	Equal to: If the value of two operands is equal,

			then the condition becomes true.
2	!=	a != b	Not Equal to: If the value of two operands is not equal, then the condition becomes true.
3	<=	a <= b	Less than or Equal to: The condition is met if the first operand is smaller than or equal to the second operand.
4	>=	a >= b	Greater than or Equal to: The condition is met if the first operand is greater than or equal to the second operand.
5	>	a > b	Greater than: If the first operand is greater than the second operand, then the condition becomes true.
6	<	a < b	Less than: If the first operand is less than the second operand, then the condition becomes true.

Logical Operators:

The assessment of expressions to make decisions typically uses logical operators. Python offers different types of logical operators such as and, or, and not. In the case of the logical AND, if the first one is 0, it does not depend upon the second one. In the case of the logical OR, if the first one is 1, it does not depend on the second one.

S. No.	Operator	Syntax	Description
1	and	a and b	**Logical AND:** The condition will also be true if the expression is true. If the two expressions a and b are the same, then a and b both must be true.
2	or	a or b	**Logical OR:** The condition will be true if one of the phrases is true. If a and b are the two expressions, then either a or b must be true to make the condition true.
3	not	not a	**Logical NOT:** If an expression a is true, then not (a) will be false and vice versa.

Bitwise Operators:

The two operands' values are processed bit by bit by the bitwise operators. There are various Bitwise operators used in Python, such as bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>).

S. No.	Operator	Syntax	Description
1	&	a & b	Bitwise AND: 1 is copied to the result if both bits in two operands at the same location are 1. If not, 0 is copied.

2	|	a | b	Bitwise OR: The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
3	^	a ^ b	Bitwise XOR: If the two bits are different, the outcome bit will be 1, else it will be 0.
4	~	~a	Bitwise NOT: The operand's bits are calculated as their negations, so if one bit is 0, the next bit will be 1, and vice versa.
5	<<	a <<	Bitwise Left Shift: The number of bits in the right operand is multiplied by the leftward shift of the value of the left operand.
6	>>	a >>	Bitwise Right Shift: The left operand is moved right by the number of bits present in the right operand.

4. Conditional Statements

Question-1: Introduction to conditional statements: if, else, elif

Answer:

if Statement in Python

If the simple code of block is to be performed if the condition holds true then the if statement is used. Here the condition mentioned holds then the code of the block runs otherwise not.

Python if Statement Syntax

Syntax*: if condition:*

 # Statements to execute if

 # condition is true

if else Statement in Python

In conditional if Statement the additional block of code is merged as else statement which is performed when if condition is false.

Python if-else Statement Syntax

Syntax*: if (condition): # Executes this block if # condition is trueelse: # Executes this block if # condition is false*

if-elif Statement in Python

The if-elif statement is shortcut of if..else chain. While using if-elif statement at the end else block is added which is performed if none of the above if-elif statement is true.

Python if-elif Statement Syntax:-

Syntax*: if (condition): statementelif (condition): statement..else: statement*

Question-2: Nested if-else conditions.

Answer:

if statement can also be checked inside other if statement. This conditional statement is called a nested if statement. This means that inner if condition will be checked only if outer if condition is true and by this, we can see multiple conditions to be satisfied.

Python Nested If Statement Syntax

Syntax: *if (condition1): # Executes when condition1 is true if (condition2): # Executes when condition2 is true # if Block is end here# if Block is end here*

5. Looping (For, While)

Question-1: Introduction to for and while loops.

Answer:

For loop

In Python, a 'for loop' is used to iterate over a sequence of items, such as a Python tuple, list, string, or range. The loop will execute a block of statements for each item in the sequence.

Syntax of Python for loop

In the below syntax for is a keyword, var is the variable name, and iterable is an object which can be looped over or iterated over with the help of a for a loop. Objects like tuples, lists, sets, dictionaries, strings, etc. are called iterable. We can also use the range() function in place of iterable.

*for **var** in **iterable**:*

 # statements

While Loop

In Python, a while loop is used to repeatedly execute a block of statements while a condition is true. The loop will continue to run as long as the condition remains true.

Syntax of Python While loop

In the while loop condition is written just after the **'while'** keyword and then we write the set of statements to perform some task.

*while **condition**:*

 # Set of statements

Question-2: How loops work in Python.

Answer: Python programming language provides two types of Python loopshecking time. In this article, we will look at Python loops and understand their working with the help of examp – For loop and While loop to handle looping requirements. Loops in Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition-checking time.

Question-3: Using loops with collections (lists, tuples, etc.).

Answer:

Lists in Python:

Lists are one of the most powerful data structures in python. Lists are sequenced data types. In Python, an empty list is created using list() function. They are just like the arrays declared in other languages. But the most powerful thing is that list need not be always homogeneous. A single list can contain strings, integers, as well as other objects. Lists can also be used for implementing stacks and queues. Lists are mutable, i.e., they can be altered once declared. The elements of list can be accessed using indexing and slicing operations.

Tuples in Python:

A tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists.

Iterations in Python:

Iterations or looping can be performed in python by 'for' and 'while' loops. Apart from iterating upon a particular condition, we can also iterate on strings, lists, and tuples.

6. Generators and Iterators

Question-1: Understanding how generators work in Python.

Answer: A generator function is a special type of function that returns an iterator object. Instead of using return to send back a single value, generator functions use yield to produce a series of results over time. This allows the function to generate values and pause its execution after each yield, maintaining its state between iterations.

```
def fun(max):
    cnt = 1
    while cnt <= max:
        yield cnt
        cnt += 1


ctr = fun (5)
for n in ctr:
    print(n)
```

Output

```
1
2
3
4
5
```

Question-2: Difference between yield and return.

Answer:

S.NO.	YIELD	RETURN
1	Yield is generally used to convert a regular Python function into a generator.	Return is generally used for the end of the execution and “returns” the result to the caller statement.
2	It replaces the return of a function to suspend its execution without destroying local variables.	It exits from a function and handing back a value to its caller.
3	It is used when the generator returns an intermediate result to the caller.	It is used when a function is ready to send a value.
4	Code written after yield statement execute in next function call.	while, code written after return statement won’t execute.
5	It can run multiple times.	It only runs single time.
6	Yield statement function is executed from the last state from where the function get paused.	All function calls run the function from the start.

Question-3: Understanding iterators and creating custom iterators.

Answer: An iterator in Python is an object that holds a sequence of values and provide sequential traversal through a collection of items such as lists, tuples and dictionaries. The Python iterators object is initialized using the iter() method. It uses the next() method for iteration.

1. __iter__(): __iter__() method initializes and returns the iterator object itself.
2. __next__(): the __next__() method retrieves the next available item, throwing a Stop Iteration exception when no more items are available.

Difference between Iterator and Iterable

Iterables are objects that can return an iterator. These include built-in data structures like lists, dictionaries, and sets. Essentially, an iterable is anything you can loop over using a for loop. An iterable implements the __iter__() method, which is expected to return an iterator object.

Iterators are the objects that actually perform the iteration. They implement two methods: __iter__() and __next__(). The __iter__() method returns the iterator object itself, making iterators iterable as well.

7. Functions and Methods

Question-1: Defining and calling functions in Python.

Answer: Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Types of Functions in Python

Below are the different types of functions in Python:

- **Built-in library function:** These are Standard functions in Python that are available to use.
- **User-defined function:** We can create our own functions based on our requirements.

Creating a Function in Python

We can define a function in Python, using the **def** keyword. We can add any type of functionalities and properties to it as we require. By the following example, we can understand how to write a function in Python. In this way we can create Python function definition by using def keyword.

```
# A simple Python function

def fun():

    print("Welcome to GFG")
```

Calling a Function in Python

After creating a function in Python we can call it by using the name of the functions Python followed by parenthesis containing parameters of that particular function. Below is the example for calling def function Python.

```
# Driver code to call a function

fun()
```

Question-2: Function arguments (positional, keyword, default).

Answer:

Keyword-Only Arguments

Keyword-only arguments mean whenever we pass the arguments(or value) by their parameter names at the time of calling the function in Python in which if you change the position of arguments then there will be no change in the output.

Benefits of using Keyword arguments over positional arguments

- On using keyword arguments you will get the correct output because the order of argument doesn't matter provided the logic of your code is correct. But in the case of positional arguments, you will get more than one output on changing the order of the arguments.

Positional-Only Arguments

Position-only arguments mean whenever we pass the arguments in the order we have defined function parameters in which if you change the argument position then you may get the unexpected output. We should use positional Arguments whenever we know the order of argument to be passed. So now, we will call the function by using the position-only arguments in two ways and In both cases, we will be getting different outputs from which one will be correct and another one will be incorrect.

Question-3: Scope of variables in Python.

Answer: A variable is only available from inside the region it is created. This is called scope.

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

Question-4: Built-in methods for strings, lists, etc.

Answer:

Python String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found

List Methods

Let's look at different list methods in Python:

- append(): Adds an element to the end of the list.
- copy(): Returns a shallow copy of the list.
- clear(): Removes all elements from the list.
- count(): Returns the number of times a specified element appears in the list.

8. Control Statements (Break, Continue, Pass)

Question-1: Understanding the role of break, continue, and pass in Python loops.

Answer:

Break Statement in Python

The break statement in Python is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available. If the break statement is present in the nested loop, then it terminates only those loops which contain the break statement.

Syntax of Break Statement

The break statement in Python has the following syntax:

```
for / while loop:
   # statement(s)
   if condition:
      break
   # statement(s)
# loop end
```

Continue Statement in Python

Continue is also a loop control statement just like the break statement. continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop. As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

Syntax of Continue Statement

The continue statement in Python has the following syntax:

```
for / while loop:
   # statement(s)
   if condition:
      continue
   # statement(s)
```

Pass Statement in Python

As the name suggests pass statement simply does nothing. The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like a null operation, as nothing will happen if it is executed. Pass statements

can also be used for writing empty loops. Pass is also used for empty control statements, functions, and classes.

Syntax of Pass Statement

The pass statement in Python has the following syntax:

```
function/ condition / loop:
  pass
```

9. String Manipulation

Question-1: Understanding how to access and manipulate strings.

Answer: A string is known as a sequence of characters, and string manipulation is something we programmers do all the time. In Python, Strings are arrays of bytes that represent Unicode characters.

Python does not contain any character variables compared to other programming languages. In Python, characters are instead strings of unit length.

String manipulation is the process of manipulating and analyzing strings. It involves modification and parsing of strings to use and change their data.

Question-2: Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

Answer:

Concatenation of Strings

Concatenation means to join two or more strings into a single string. **+** operator is used to concatenate strings. For example,

```
word1 = 'Coding'

word2 = ' Ninjas'

print(word1 + word2)
```

Output:
Coding Ninjas

String repetition

In Python, you can multiply strings to create a new string that repeats the original string a certain number of times. This technique is known as string repetition or string replication.

The multiplication operator (*) is used to repeat a string a specified number of times.

```
string = "hello "

result = string * 3

print(result)
```

output:

hello hello hello

lower() Method

The **lower()** method in Python converts all uppercase letters in a string to their lowercase. This method does not alter non-letter characters (e.g., numbers, punctuation).

```
s = "HELLO, WORLD!

res = s.lower()

print(res)
```

Output

hello, world!

upper() Method

The **upper()** is a method of string objects in Python. It creates a new string with all lowercase letters changed to uppercase. This method does not change the original string instead it simply returns a new one.

```
s = "hello, world!"

res = s.upper()

print(res)
```

Output

```
HELLO, WORLD!
```

Question-3: String slicing.

Answer: String slicing in Python is a way to get specific parts of a string by using start, end, and step values. It's especially useful for text manipulation and data parsing.

```
s = "Hello, Python!"

s2 = s[0:5]

print(s2)
```

Output

```
Hello
```

10. Advanced Python (map(), reduce(), filter(), Closures and Decorators)

Question-1: How functional programming works in Python.

Answer: Functional programming is a programming paradigm in which we try to bind everything in a pure mathematical functions style. It is a declarative type of programming style. Its main focus is on " what to solve" in contrast to an imperative style where the main focus is "how to solve". It uses expressions instead of statements. An expression is evaluated to produce a value whereas a statement is executed to assign variables.

Concepts of Functional Programming

Any Functional programming language is expected to follow these concepts.

- Pure Functions: These functions have two main properties. First, they always produce the same output for the same arguments irrespective of anything else. Secondly, they have no side-effects i.e. they do modify any argument or global variables or output something.
- Recursion: There are no "for" or "while" loop in functional languages. Iteration in functional languages is implemented through recursion.
- Functions are First-Class and can be Higher-Order: First-class functions are treated as first-class variables. The first-class variables can be passed to functions as a parameter, can be returned from functions, or stored in data structures.
- Variables are Immutable: In functional programming, we can't modify a variable after it's been initialized. We can create new variables – but we can't modify existing variables.

Question-2: Using map(), reduce(), and filter() functions for processing data.

Answer:

Map Function in Python

The map () function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple, etc.).

Syntax: map(fun, iter)

Parameters:

- fun: It is a function to which map passes each element of given iterable.
- iter: iterable object to be mapped.

Reduce Function in Python

The reduce function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in "functools" module.

Syntax: reduce(func, iterable[, initial])

Parameters:

- fun: It is a function to execuate on each element of the iterable objec
- iter: It is iterable to be reduced

Filter Function in Python

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

Syntax: filter(function, sequence)

Parameters:

- function: function that tests if each element of a sequence is true or not.
- sequence: sequence which needs to be filtered, it can be sets, lists, tuples, or containers of any iterators.

Question-3: Introduction to closures and decorators.

Answer: Closures and decorators are powerful features in Python that allow for more advanced and flexible code patterns. Understanding these concepts can greatly enhance your ability to write clean, efficient, and reusable code.

Closures in Python

A closure in Python occurs when a nested function captures the local variables from its enclosing scope. This allows the nested function to access these variables even after the outer function has finished executing.

How Closures Work

Closures are created when:

1. There is a nested function.
2. The nested function references a value in its enclosing scope.
3. The enclosing function returns the nested function.

Decorators in Python

Python Decorators are a powerful and expressive tool in Python that allows you to modify the behavior of a function or method. They are often used to add "wrapping" functionality to existing functions in a clean and readable way.

How Decorators Work

A decorator is a function that takes another function as an argument, adds some kind of functionality, and returns a new function.