Division Behavior	Integer division can be forced	Division always returns float
Exceptions Handling	Less consistent chaining	Consistent exception chaining
Syntax and Method Changes	Method name inconsistencies	Improved consistency and clarity
End of Life	Python 2 reached end of life in 2020	Python 3 is the actively developed version

Learn more about Python, read our blog – what is Python?

Q2. Explain the key features of Python.

Ans. Following are multiple features that make python different from other languages:

- **Easy to learn:** Python is a programmer-friendly language that it is easy to learn in just a few days. It uses fewer keywords as compared to other object-oriented languages and anyone can learn it
- **Object-oriented:** It is both procedure-oriented and object-oriented programming language. In procedure-oriented languages, procedures can be used again. It allows developers to use an object-oriented approach to develop applications.
- **Open-source and free:** It is an Open-source language that means anyone can easily access the programming and development. There is an open forum online where the number of coders contributes to improving this language.
- **High-level language:** It is defined as a high-level language because of this feature. You do not need to keep an eye on the programming structure, memory management, and architecture of the code.
- **Extendable & Scalable:** It is extendable as it can be extended by using different modules to its interpreters. It helps developers to modify the program. It also provides scalability to the large codes by providing support and good structure to it.

Explore popular courses on Shiksha Online:

Free Python Interview Questions by Great Learning	Popular Data Structures and Algorithms Courses
Programming for Everybody by University of Michigan	Google IT Automation with Python Professional Certificate
Top Python Courses	Top Free Python Courses

Must Read: Introduction to Python – Features, Use Cases, Resources, and Applications

Q3. What is the difference between List and Tuple?

Aspect	Lists	Tuples
Mutability	Mutable (can be changed)	Immutable (cannot be changed)
Syntax	my_list = [1, 2, 3]	my_tuple = (1, 2, 3)
Use Case	For collections of items that may change	For collections that should not change
Modification	Elements can be added, removed, or modified	Cannot modify elements after creation
Performance	Slightly slower than tuples due to added overhead for mutation	Slightly faster and memory-efficient
Hashability	Not hashable (cannot be used as dictionary keys or elements in sets)	Hashable (can be used as dictionary keys or elements in sets)
Parentheses	Square brackets []	Parentheses ()
Iteration	Similar iteration using loops or list comprehensions	Similar iteration methods
Length	Usually longer syntax: len(my_list)	Shorter syntax: len(my_tuple)
Common Methods	append(), extend(), insert(), remove(), etc.	Limited methods: count(), index()
Immutability Use	Used when data should be changed or updated	Used when data should remain constant
Examples	names = [“Alice”, “Bob”, “Charlie”]	coordinates = (10, 20)

Ans. PEP 8, also known as Python Enhancement Proposal 8, is the official style guide for writing Python code. It outlines a set of conventions and recommendations for formatting code to ensure consistency and readability, making it easier for developers to collaborate on projects.

PEP 8 covers various aspects of code style, including indentation, naming conventions, comments, and more. Following these guidelines not only makes your code more pleasant to read but also helps maintain a unified style across projects.

Here's a simple example to illustrate some PEP 8 principles:

Copy code

```
# PEP 8-compliant code

def calculate_square(x):
    """Calculate the square of a number."""
    return x ** 2


class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        """Greet the person."""
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Creating instances of the Person class
alice = Person("Alice", 30)
bob = Person("Bob", 25)

alice.greet()
bob.greet()
```

In this example, the code adheres to several PEP 8 principles:

- **Indentation:** Code is indented using four spaces.
- **Naming Conventions:** Function and variable names are in lowercase with words separated by underscores (calculate_square, alice, bob).
- **Comments:** Comments are used to explain the purpose of functions and classes.
- **Class Definitions:** Class names are in CamelCase (Person).
- **Whitespace in Expressions and Statements:** There is proper spacing around operators and after commas.

By following PEP 8, your code becomes more consistent and easier to read, which is especially helpful when working on collaborative projects or maintaining code over time.

Q5. How is Python different from other scripting languages?

Ans. Python as a scripting language is different from other programming languages in different ways. Let's compare some of the popular scripting language like Perl, Ruby, and Bash with Python.

Feature/Aspect	Python	Perl	Ruby	Bash
Syntax & Readability	Clear, concise, indentation-based	Flexible, multiple ways to do the same	Natural language readability, clean syntax	Command language, suitable for short scripts

Dynamic Typing	Yes, with optional type hints	Yes, with many built-in data types	Yes, strong object-oriented features	Limited to strings and integers
Standard Library	Vast, “batteries-included”	Comprehensive, text processing focus	Rich, especially for OOP	Command execution, text manipulation
Community	Large, active	Was popular in 90s, smaller now	Active, especially around Rails	Unix system administration focus
Performance	Interpreted, can be optimized	Interpreted, fast text processing	Interpreted, comparable to Python	Suitable for short scripts
Platform Independence	Cross-platform	Cross-platform, Unix focus	Cross-platform, some platform-specific gems	Primarily for Unix-like systems
Development Philosophy	“Zen of Python”, simplicity, readability	“There’s more than one way to do it”, flexibility	Influenced by Perl & Smalltalk, natural readability	Command execution, Unix scripting

Q6. Explain memory management in Python.

To understand memory management in Python as imagine your computer’s memory as a giant storage room. When you create something in Python, like a list or a string, you’re essentially asking Python, “Hey, can I have a small space in this room to store my stuff?” Python then takes care of finding the right spot for you.

1. **Private Heap Space**:
   - Think of this as a special section of the storage room where Python keeps all the things (objects, data) you create.
   - Python maintains an internal private heap, where all objects and data structures are stored and managed by the Python memory manager.
2. **Reference Counting**:
   - Imagine you have a toy (an object) in the storage room. Every time you play with it (use the object), you add a sticker to it. When you’re done playing (stop using the object), you remove a sticker. If the toy has no stickers left, Python knows no one is playing with it, so it can be thrown away to free up space. This is what reference counting is.
   - Python uses reference counting as a primary memory management technique. Every object has a count of the number of references pointing to it. When this count drops to zero, the memory occupied by the object can be reclaimed.
3. **Garbage Collection**:
   - Over time, some toys might get forgotten in the corner, even if they have no stickers. Every once in a while, Python sends a cleaner (garbage collector) to find and remove these forgotten toys, ensuring the room doesn’t get too cluttered.

Python’s memory management is a combination of reference counting and garbage collection. While reference counting handles most cases, the garbage collector ensures that circular references don’t lead to memory leaks. This combination ensures efficient memory management in Python applications.

Must Read: Memory Management in Python

Q7. What are the modules in Python? Name some built-in modules.

Ans. In Python, a module is a file containing Python definitions and statements. The file name is the module name with the suffix **.py** is added. Modules are a way to organize related code into a single file, making the code easier to understand and use. They can define functions, classes, and variables, and can also include runnable code.

In other words, a module is like a file that holds related pieces of code. This file ends with “**.py**“. Think of it as a folder on your computer where you store similar photos or documents. Just as you might have a folder for vacation photos, a module groups related functions, classes, and other code. This makes it easier to find, use, and understand that code.

Python modules are the files that contain Python statement and definitions such as –

```
# This is the arithmetic module

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Cannot divide by zero!"
    return a / b
```

Copy code

In the above module (***arithmetic.py***), we have defined four functions: add, subtract, multiply, and divide.

Now, if you want to use these functions in another Python script, you can simply import the arithmetic module and access its functions.

Example usage in another script:

```
import arithmetic

result = arithmetic.add(5, 3)
print(result)  # This will print 8
```

Copy code

Python modules are used to break the large program into small segments that make the code manageable and organized.

Modules serve several purposes:

1. **Code Reusability**: Once you've written a module, you can reuse it across multiple programs.
2. **Namespace Partitioning**: Modules help avoid naming conflicts by creating a separate namespace for identifiers.
3. **Organizing Large Projects**: For bigger projects, organizing code into different modules and packages is essential for maintainability.

Some Built-in Modules in Python:

1. **math**: Provides mathematical functions. For example, **math.sqrt()** computes the square root.
2. **sys**: Provides access to Python interpreter variables. For instance, **sys.argv** gives command-line arguments.
3. **os**: Offers a way to use operating system-dependent functionality like reading or writing to the file system.
4. **datetime**: Supplies classes for manipulating dates and times.
5. **collections**: Implements several specialized container datatypes like named tuples, deques, and Counters.
6. **json**: Allows encoding and decoding JSON data.
7. **random**: Contains functions to generate random numbers.

These are just a few of the many built-in modules in Python. You can use the help('modules')command in the Python interpreter to see a complete list of available modules.

Q8. How do you convert a string to an integer and vice versa in Python?

In Python, converting between strings and integers is straightforward using built-in functions.

1. **Converting a String to an Integer:**
   To convert a string to an integer, you can use the int() function.

```
string_num = "123"
int_num = int(string_num)
print(int_num)  # Outputs: 123
print(type(int_num))  # Outputs: <class 'int'>
```

Note: The string should represent a valid integer. If you try to convert a string that doesn't represent an integer (e.g., "123.45" or "abc"), you'll get a ValueError.

2. **Converting an Integer to a String:**
   To convert an integer to a string, you can use the str() function.

```
int_num = 123
string_num = str(int_num)
print(string_num)  # Outputs: "123"
print(type(string_num))  # Outputs: <class 'str'>
```

These conversions are commonly used in various scenarios, such as reading input from users, processing data from files, or formatting output.

Read More: Data Type in Python

Q9. How would you merge two dictionaries?

In Python, you can merge two dictionaries in several ways. Here are some of the most common methods:

1. **Using the update() method:**
   This method modifies the first dictionary in place by adding keys and values from the second dictionary.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
dict1.update(dict2)
print(dict1)  # Outputs: {'a': 1, 'b': 3, 'c': 4}
```

Note: If the second dictionary has keys that are already present in the first dictionary, their values will be updated in the first dictionary.

Read More: Dictionary in Python

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged_dict = {**dict1, **dict2}
print(merged_dict)  # Outputs: {'a': 1, 'b': 3, 'c': 4}
```

3. **Using the | merge operator (Python 3.9+):**
   This is a new way introduced in Python 3.9 to merge dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged_dict = dict1 | dict2
print(merged_dict)  # Outputs: {'a': 1, 'b': 3, 'c': 4}
```

4. **Using the copy() method and update():**
   This method creates a new dictionary without modifying the original dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged_dict = dict1.copy()
merged_dict.update(dict2)
print(merged_dict)  # Outputs: {'a': 1, 'b': 3, 'c': 4}
```

All of these methods will effectively merge two dictionaries. The choice of method often depends on the specific requirements and the Python version being used.

Q10. What's the difference between append() and extend() methods of a list?

Both append() and extend() are methods of a list in Python, but they serve different purposes:

- **append()** adds its entire argument as a single element to the end of the list.
- **extend()** adds each element of its argument to the list. The argument should be an iterable.

1. **append() method:**
   The append() method adds its entire argument as a single element to the end of the list.
   The new length of the array will be len(list) + 1.
   Example:

```
list1 = [1, 2, 3]
list1.append(4)
print(list1)  # Outputs: [1, 2, 3, 4]

list1.append([5, 6])
```

2. **extend() method:**
   The extend() method takes an iterable (like a list, tuple, string, etc.) and adds each of its elements to the list.
   The new length of the array will be len(list) + len(iterable).
   Example:

```
list1 = [1, 2, 3]
list1.extend([4, 5])
print(list1)  # Outputs: [1, 2, 3, 4, 5]

list1.extend((6, 7))
print(list1)  # Outputs: [1, 2, 3, 4, 5, 6, 7]

list1.extend("89")
print(list1)  # Outputs: [1, 2, 3, 4, 5, 6, 7, '8', '9']
```

Copy code

In the examples above, each element of the iterable passed to extend() is added individually to list1.

Q11. How can you reverse a string in Python?

You can reverse a string in Python using slicing method.

Example:

```
string = "Hello"
reversed_string = string[::-1]
print(reversed_string)  # Outputs: "olleH"
```

Copy code

Q12. What is string slicing? Provide an example.

String slicing is a technique in Python to extract a portion (or slice) of a string. It uses the format:

```
string[start:stop:step]
```

Copy code

- **start:** The beginning index of the slice. It's inclusive, which means the slice starts at this index. If omitted, it defaults to 0 for positive step values and -1 for negative step values.
- **stop:** The ending index of the slice. It's exclusive, which means the slice goes up to, but does not include, this index. If omitted, it defaults to the end of the string.
- **step:** The interval between characters in the slice. If omitted, it defaults to 1.
- **Example:**

```
slice1 = string[1:4]  # "yth"
slice2 = string[::2]  # "Pto"
slice3 = string[-1:-4:-1]  # "noh"
```

Q13. How would you check if a substring exists within a string?

You can use the in keyword to check if a substring exists within a string.

Example:

Copy code

```
string = "Hello, World!"
substring = "World"

if substring in string:
    print(f"'{substring}' exists in '{string}'")
else:
    print(f"'{substring}' does not exist in '{string}'")
```

Q14. What is a tuple? How is it different from a list?

A tuple is an ordered collection of elements, similar to a list. However, unlike lists, tuples are immutable, meaning their contents cannot be modified after they are created. Tuples are defined by enclosing the elements in parentheses ().

Example:

Copy code

```
my_tuple = (1, 2, 3, "Python")
```

How is a Tuple it different from a list?

The primary differences between a tuple and a list are:

- **Mutability**: Lists are mutable (can be modified), while tuples are immutable (cannot be modified).
- **Syntax**: Lists are defined using square brackets [] , whereas tuples are defined using parentheses ().

Must Read: Understanding Tuples in Python

Q15. Can you modify an element inside a tuple? Why or why not?

No, you cannot modify an element inside a tuple. This is because tuples are immutable. Once a tuple is created, you cannot change, add, or remove elements from it. This immutability provides certain advantages in terms of data safety and can also lead to performance improvements in some scenarios.

Q16. When would you use a tuple over a list?

Using a tuple over a list is often a matter of context and the specific requirements of a problem. Here are some scenarios where a tuple might be preferred over a list, along with examples:

1. **Data Integrity**:

When you want to ensure that the data remains constant and doesn't get modified accidentally, tuples are a safer choice because they are immutable.

```
point_3D = (4, 5, 6)
```

Reason: Coordinates of a point are fixed values and should not change once defined. Using a tuple ensures that these values remain constant and cannot be accidentally modified, preserving data integrity.

2. **Using as Dictionary Keys**:

Tuples can be used as keys in dictionaries because they are hashable, whereas lists are not.

Example: Imagine you want to store distances between cities. The pair of cities can be a tuple which acts as a key in a dictionary.

```
distances = {
    ('New York', 'Los Angeles'): 2451,
    ('Houston', 'Chicago'): 1089
}
```

Reason: Dictionary keys need to be hashable and immutable. Lists are mutable and therefore cannot be used as dictionary keys. Tuples, being immutable, can serve as dictionary keys, allowing us to represent a pair of cities as a single key.

3. **Function Multiple Returns**:

When functions return multiple values, they are packed into tuples.

Example: A function that returns both the quotient and remainder of a division operation.

```
def divide(a, b):
    quotient = a // b
    remainder = a % b
    return quotient, remainder

result = divide(10, 3)
print(result)  # Outputs: (3, 1)
```

Reason: When a function returns multiple values, they are packed into a tuple by default. This allows for a clean and efficient way to return multiple pieces of data from a function without needing to create a specific data structure.

4. **Packing and Unpacking**:

Tuples are often used for packing multiple values together and then unpacking them elsewhere in the code.

Example: Swapping two values without a temporary variable.

```
a = 5
b = 10
```

Reason: Tuples allow for a concise way to pack and unpack values. In the swapping example, the values of a and b are packed into a tuple and then immediately unpacked in a swapped order. This makes the code more readable and eliminates the need for a temporary variable.

5. **Performance**:

For read-heavy operations where the data doesn't change, tuples can be slightly faster than lists because of their static nature.

Example: If you have a static set of values that you're iterating over frequently, a tuple might offer better performance.

```
colors = ('red', 'green', 'blue')
for color in colors:
    print(color)
```

Copy code

Reason: Tuples can be slightly faster than lists for read-heavy operations due to their static nature. If the data set is not going to change and is frequently accessed, using a tuple can offer a minor performance advantage.

Q17. How can you remove duplicates from a list?

There are several ways to remove duplicates from a list in Python. Here are some of the most common methods:

1. Using a Set:

Converting a list to a set will automatically remove duplicates because sets cannot have duplicate values. However, this method does not preserve the original order of the list.

```
my_list = [1, 2, 2, 3, 4, 4, 5]
no_duplicates = list(set(my_list))
print(no_duplicates)  # The order might change, e.g., [1, 2, 3, 4, 5]
```

Copy code

2. Using List Comprehension:

This method preserves the original order of the list.

```
my_list = [1, 2, 2, 3, 4, 4, 5]
no_duplicates = []
[no_duplicates.append(x) for x in my_list if x not in no_duplicates]
print(no_duplicates)  # Outputs: [1, 2, 3, 4, 5]
```

Copy code

3. Using dict.fromkeys():

This method also preserves the original order of the list.

```
no_duplicates = list(dict.fromkeys(my_list))
print(no_duplicates)  # Outputs: [1, 2, 3, 4, 5]
```

4. Using a Loop:

This is a more traditional approach and preserves the original order.

Copy code

```
my_list = [1, 2, 2, 3, 4, 4, 5]
no_duplicates = []
for item in my_list:
    if item not in no_duplicates:
        no_duplicates.append(item)
print(no_duplicates)  # Outputs: [1, 2, 3, 4, 5]
```

Each of these methods has its own advantages. The choice of method often depends on the specific requirements of the problem, such as whether the original order of the list needs to be preserved.

Q18. What is the scope of a variable in Python?

In Python, the scope of a variable refers to the region of the code where a variable can be accessed or modified. The scope determines the visibility of a variable in different parts of the code. There are four primary types of variable scopes in Python:

1. **Local Scope**:
   - A variable declared inside a function or method has a local scope.
   - It is only accessible within that function or method and not outside of it.
   - The variable is created when the function is called and destroyed once the function exits.

Example:

Copy code

```
def my_function():
    local_variable = "I'm local"
    print(local_variable)

my_function()  # Outputs: "I'm local"
```

2. **Enclosing (or Non-local) Scope**:

- This scope is specific to nested functions.
- If a variable is defined in an enclosing function (a function containing another function), the inner function has access to the variable, but it's considered non-local to the inner function.

Example:

Copy code

```
def outer_function():
    enclosing_variable = "I'm enclosing"
```

```
    inner_function()

outer_function()  # Outputs: "I'm enclosing"
```

3. Global Scope:

- A variable declared outside of all functions, methods, or classes has a global scope.
- It is accessible throughout the module or file.
- If you need to modify a global variable from within a function, you must declare it using the **global** keyword inside that function.

Example:

Copy code

```
global_variable = "I'm global"

def my_function():
    global global_variable
    global_variable = "Modified global"
    print(global_variable)

my_function()  # Outputs: "Modified global"
print(global_variable)  # Outputs: "Modified global"
```

4. Built-in Scope:

- This scope encompasses all the built-in names in Python, such as functions (print, len, etc.) and exceptions (ValueError, TypeError, etc.).
- These names are always available and don't need to be imported.

Example:

Copy code

```
def my_function():
    print(len([1, 2, 3]))  # Using the built-in len() function

my_function()  # Outputs: 3
```

Q19. Explain the difference between local variables and global variables.

Ans. Global variables are declared outside the function and it can be used both inside and outside the function. It has a global scope.

Local variables are those which are declared inside the function or in the local scope which cannot be used outside the function.

Aspect	Local Variables	Global Variables
Definition	Variables defined inside a function or method.	Variables defined outside of all functions and methods.
Scope	Accessible only within the function they are defined.	Accessible throughout the entire module or file.

		the **global** keyword.

Let's understand this with the help of an analogy. Imagine a house where each room represents a function in Python. The items inside each room are like local variables, and the items outside in the garden or on the porch are like global variables.

Aspect	Local Variables (Items inside a room)	Global Variables (Items in the garden/porch)
Accessibility	You can only use items inside a room while you're in that specific room. Once you leave, you can't access them.	Items in the garden or on the porch can be seen and used from any room with a window or door facing them.
Lifetime	Items in a room (like a toy or book) are only available when the room is open. Once you lock the room, they're out of reach.	The garden gnome or the porch swing is always there, no matter which room you're in or even if all rooms are locked.
Intervention	If you want to use an item from another room, you have to bring it into your current room first.	If you want to change the color of the garden gnome, you can do it from any room, but you need to announce ("declare") you're doing it.

NOTE:

- Just like you can't use a toy from the bedroom while you're in the kitchen (unless you bring it with you), you can't access a local variable outside its function.
- The garden gnome, visible from multiple rooms, is like a global variable. Everyone knows it's there, and with the right declaration, anyone can change its color. But if everyone starts painting the gnome without caution, it can lead to confusion about its current color, just as overusing global variables can lead to unpredictable code behavior.

Example:

Copy code

```
# Global Variable (Garden gnome)
garden_gnome_color = "blue"

def bedroom():
    # Local Variable (Toy inside the bedroom)
    toy_color = "red"
    print(f"In the bedroom, my toy is {toy_color}.")
    print(f"From the bedroom, I see a {garden_gnome_color} garden gnome.")

def kitchen():
    # Local Variable (Item inside the kitchen)
    mug_color = "green"
    print(f"In the kitchen, my mug is {mug_color}.")
    print(f"From the kitchen, I see a {garden_gnome_color} garden gnome.")

def paint_gnome(new_color):
    global garden_gnome_color
    garden_gnome_color = new_color
    print(f"I've painted the garden gnome {new_color}!")

# Let's see the house functions in action
bedroom()
kitchen()
paint_gnome("yellow")
bedroom()
```

In this example:

- The **paint_gnome** function allows us to change the color of the garden gnome, and we declare that we're modifying the global variable using the global keyword.

Must Read: **Variables in Python**

Q20. What is the difference between Python Arrays and Lists?

Aspect	Python List	Python Array (from array module)
Definition	A versatile and general-purpose container that can store elements of mixed data types.	A container that can store elements of a specific data type.
Import Required	No, it's a built-in data type.	Yes, you need to **import** array.
Data Type Homogeneity	Lists can store elements of mixed data types.	Arrays store elements of the same data type.
Functionality	Provides a wide range of methods for manipulation (e.g., **append, remove, insert**).	Offers fewer methods; mainly focused on basic operations like append, pop, and array-specific ones like **tofile** and **fromfile**.
Memory Consumption	Generally consumes more memory due to flexibility in storing different data types.	Consumes less memory as it's more size efficient for a specific data type.
Use Case	Suitable for general purposes where the data type of elements can vary.	Suitable when working with a large amount of data of a single data type, especially when memory efficiency is a concern.
Example	my_list = [1, "hello", 3.14]	python import array my_array = array.array('i', [1, 2, 3, 4])

It's important to note that while Python's list is more commonly used due to its flexibility, the array module provides a more space-efficient way to store data of a single type, especially useful in scenarios where performance and memory usage are crucial.

Must Read: Lists in python

Must Read: Array manipulation using NumPy

Q21. What's the difference between arguments and parameters?

- **Parameters** are the names listed in the function definition. They act as placeholders for the values you'll pass into the function when you call it.
- **Arguments** are the actual values that you pass into a function when you call it. These values get assigned to the parameters and are used within the function.
- **Example:**

Copy code

```
def greet(name):  # 'name' is a parameter
    print(f"Hello, {name}!")

greet("Alice")  # "Alice" is an argument
```

Q22. How can you define a function with default argument values?

You can provide default values to parameters by using the assignment operator (=) in the function definition. If an argument for that parameter is not provided when calling the function, the default value will be used.

Example:

```
    print(f"Hello, {name}!")

greet()        # Outputs: Hello, Guest!
greet("Bob")   # Outputs: Hello, Bob!
```

Must Read: Functions in Python

Q23. What are positional arguments and keyword arguments? Provide examples.

- **Positional Arguments**: These are arguments that are passed in order and are assigned to parameters based on their position.
- **Keyword Arguments**: These are arguments passed by explicitly naming the parameter along with its value. They can be provided in any order.
- Example:

Copy code

```
def display_info(name, age):
    print(f"{name} is {age} years old.")

display_info("Charlie", 30)      # Positional arguments
display_info(age=25, name="Eve")  # Keyword arguments
```

Q24. How can you pass a variable number of arguments to a function?

- You can use ***args** to pass a variable number of positional arguments. Inside the function, args will be a tuple containing all the passed positional arguments.
- For a variable number of keyword arguments, you can use ****kwargs**. Inside the function, kwargs will be a dictionary containing all the passed keyword arguments.

Example:

Copy code

```
def display_names(*names):
    for name in names:
        print(name)

display_names("Anna", "Brian", "Catherine")

def display_data(**data):
    for key, value in data.items():
        print(f"{key}: {value}")

display_data(name="David", age=40, profession="Engineer")
```

In the first function, **display_names**, any number of positional arguments can be passed, and they'll be accessible as a tuple named names. In the second function, **display_data**, any number of keyword arguments can be passed, and they'll be accessible as a dictionary named data.

Q25. Explain the use and purpose of *args and **kwargs. How do they differ?

the number of scoops: “1 scoop, 2 scoops, 3 scoops…”. You don’t specify anything else, just the number of scoops.

- **Simple Explanation:** *args lets you send any number of single items (like scoops of ice cream) to a function.
- **Example:**

```
def order_scoops(*scoops):
    print(f"Ordering {len(scoops)} scoops of ice cream!")
    for scoop in scoops:
        print(f"One scoop of {scoop} flavor.")

order_scoops("vanilla", "chocolate", "strawberry")
```

Copy code

****kwargs:**

- **Analogy:** Now, think of **kwargs as the “sundae” option. For a sundae, you don’t just count scoops. You specify: “1 scoop vanilla, 1 scoop chocolate, with nuts, no cherry on top”. Each part of your order has a name (like “vanilla” or “chocolate”) and a corresponding value (like “1 scoop” or “with nuts”).
- **Simple Explanation:** **kwargs lets you send items with specific names and values (like ingredients in a sundae) to a function.
- **Example:**

```
def build_sundae(**ingredients):
    print("Building a custom sundae with:")
    for ingredient, value in ingredients.items():
        print(f"{value} of {ingredient}")

build_sundae(vanilla="1 scoop", chocolate="1 scoop", topping="nuts", cherry="no cherry on top")
```

Copy code

Difference between *args and **kwargs:

With *args, you’re just counting items (like scoops).
With **kwargs, you’re naming each item and giving it a value (like building a custom sundae).
So, when you think of *args and **kwargs, think of scoops of ice cream vs. building a sundae!

NOTE:

- ***args** allows you to pass any number of positional arguments, which are then accessed as a tuple.
- ****kwargs** allows you to pass any number of keyword arguments, which are then accessed as a dictionary.

Q26. What is a Lambda Function in Python?

Ans: A lambda function in Python is a small, anonymous function that can have any number of arguments but can only have one expression. The expression is evaluated and returned when the lambda function is called. Lambda functions are used for creating quick functions without the need to formally define a function using the def keyword.

Analogy: Imagine you’re at a café, and instead of ordering a regular menu item (like a sandwich or a salad), you just want a quick snack. Instead of going through the whole process of looking at the menu, deciding on an item,

- The regular menu items (sandwich, salad) are like standard functions defined using the **def** keyword. They have a name, they can be complex, and they can perform multiple tasks.
- The piece of fruit from the counter is like a lambda function. It's quick, nameless, serves a single purpose, and doesn't require a lot of formalities.

Example in Python: Instead of defining a function in the standard way:

```
def add(x, y):
    return x + y
```

You can use a lambda function for the same purpose:

```
add = lambda x, y: x + y
```

Q27. How would you write a lambda function to square a number?

A lambda function to square a number can be written as:

```
square = lambda x: x**2
```

This lambda function can be used as:

```
print(square(5))  # Outputs: 25
```

Q28. Can lambda functions have multiple inputs? If so, provide an example.

Yes, lambda functions can have multiple inputs.

Example: A lambda function to add two numbers:

```
add = lambda x, y: x + y
print(add(3, 4))  # Outputs: 7
```

Similarly, you can have more than two inputs if needed. For instance, to find the product of three numbers:

```
print(product(2, 3, 4))  # Outputs: 24
```

Must Read: Lambda Function in Python

Q29. How does the return statement work in Python functions?

The **return** statement in Python is used to send a value back from a function to the place where the function was called. When a function encounters the return statement, it immediately exits the function, and no further code inside that function is executed. If you don't use a return statement, the function will return **None** by default.

Example:

Imagine a simple vending machine function. You give it some money, and it returns a snack to you.

Copy code

```
def vending_machine(money):
    if money == 1:
        return "chips"
    elif money == 2:
        return "soda"
    else:
        return "Invalid amount"
```

Here's how it works:

- If you give the vending machine 1 unit of money (money = = 1), it returns "chips".
- If you give it 2 units of money (money = = 2), it returns "soda".
- For any other amount, it returns "Invalid amount".

Now, when you use this function:

Copy code

```
snack = vending_machine(1)
print(snack)  # Outputs: chips
```

In this example, you gave the **vending_machine** function 1 unit of money, and it returned "chips" to you. The return statement in the function sent the value "chips" back, which was then stored in the snack variable and printed out.

In simple terms, The ***return*** statement in Python functions is like getting an item back after you've made a request. It sends a specific value from the function back to the place where the function was called.

Q30. Can a function return multiple values? If so, how?

Yes, a function in Python can return multiple values. When it does, the values are typically packed into a tuple and returned as a single tuple object. You can then unpack these values into multiple variables when you call the function.

Example:

Let's say you have a function that calculates both the quotient and remainder of two numbers:

```
    quotient = a // b
    remainder = a % b
    return quotient, remainder
```

When you call this function, it will return both the quotient and the remainder:

Copy code

```
result = divide_and_remainder(10, 3)
print(result) # Outputs: (3, 1)
```

The function returned a tuple (3, 1), where 3 is the quotient and 1 is the remainder.

You can also unpack the returned values into separate variables:

Copy code

```
quot, rem = divide_and_remainder(10, 3)
print(quot)  # Outputs: 3
print(rem)   # Outputs: 1
```

Here, **quot** holds the quotient, and **rem** holds the remainder.

In short, while a function technically returns a single object, by using tuples (or other data structures like lists or dictionaries), you can effectively return multiple values from a function and then unpack them as needed.

Q31. What are recursive functions? Why do we use it?

A **recursive function** is a function that calls itself in its definition. It's a method used in programming to solve problems by breaking them down into smaller and smaller sub-problems until the sub-problem is simple enough to be solved directly.

Scenario: File System Search

Imagine you're building a program to search for a specific file within a computer's file system. Directories can have files and other sub-directories inside them, and those sub-directories can have their own files and further sub-directories, and so on. This nested structure is inherently recursive.

To search for a file, you'd:

1. Look at the current directory's files.
2. If you find the file, great!
3. If not, you'd then go into each sub-directory and repeat the process.

This is a natural fit for a recursive approach. Here's a simplified example:

Copy code

```
def find_file(target, current_directory):
    for item in current_directory:
        if item is a file:
            if item.name == target:
                return item.path
        elif item is a directory:
```

In this scenario, the **find_file** function calls itself to dive into sub-directories, making the problem manageable by breaking it down level by level.

In essence, recursive functions are particularly useful when the problem itself has a recursive nature, like traversing tree structures, solving puzzles like the **Tower of Hanoi**, or calculating **factorials** and **Fibonacci sequences**.

Q32. What is a decorator in Python? Provide an example of a simple decorator with a scenario. based example.

In Python, a decorator is a special type of function that allows you to add or modify the behavior of another function or method without changing its source code. It's like wrapping a gift; the core gift remains the same, but the wrapping can change its appearance or add some additional touches.

Scenario: Imagine you're developing a web application, and you want to measure the time it takes for certain functions to run, so you can identify any performance bottlenecks.

Instead of adding timing code to every function you want to measure, you can use a decorator to add this timing behavior.

Example:

Here's a simple **timer_decorator** that measures and prints the time a function takes to execute:

Copy code

```
import time

# This is the decorator function
def timer_decorator(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()

        result = func(*args, **kwargs)  # Call the original function

        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"{func.__name__} took {elapsed_time:.2f} seconds to run.")

        return result
    return wrapper

# Applying the decorator to a function using the "@" syntax
@timer_decorator
def process_data(data):
    time.sleep(2)  # Simulating some data processing that takes 2 seconds
    print(f"Processed {len(data)} items.")

# Calling the decorated function
process_data(list(range(1000)))
```

Output:

```
process_data took 2.00 seconds to run.
```

In this scenario, the **timer_decorator** allows us to easily measure the execution time of the **process_data** function (or any other function we decorate) without cluttering its code with timing logic. Decorators provide a clean and reusable way to extend or modify the behavior of functions or methods in Python.

Must Read: Python Decorators

Q33. How can you import a specific function from a module in Python?

Answer: You can use the **from … import …** statement.

Example:

Copy code

```
# Assuming there's a function called 'my_function' in a module named 'my_module'
from my_module import my_function
```

Intermediate Python Interview Question

Q36. How do Python's list comprehensions work under the hood? Can you provide an example where using a list comprehension would be more efficient than a traditional loop?

Ans: How List Comprehensions Work Under the Hood:

1. **Creation of a New List**: When a list comprehension is executed, Python creates a new list in memory to store the results.
2. **Iterating Over the Input Sequence**: Python then iterates over the input sequence (or sequences, in the case of nested comprehensions) from left to right.
3. **Evaluation of the Expression**: For each item in the input sequence, Python evaluates the expression on the left side of the comprehension and appends the result to the new list.
4. **Condition Evaluation**: If there's a conditional in the comprehension (using if), Python checks it for each item. If the condition is **True**, the item is processed; otherwise, it's skipped.
5. **Return the New List**: Once all items in the input sequence have been processed, Python returns the new list.

Under the hood, list comprehensions are implemented using a loop, but they are optimized in C (since Python is implemented in C), making them faster in many cases than equivalent Python loops written out long-form.

Efficiency of List Comprehensions:

List comprehensions can be more efficient than traditional loops in several scenarios:

1. **Memory Usage**: List comprehensions can be more memory-efficient because they generate the output list in one go, without the need for intermediate storage or append operations.
2. **Speed**: Due to their optimization in C, list comprehensions can be faster than equivalent for-loops in Python, especially for simple operations.
3. **Readability**: For simple transformations and filtering, list comprehensions can be more concise and readable than traditional loops.

Example:

Let's consider an example where we want to square all even numbers in a list:

Using a traditional loop:

```
squared_evens = []
for num in numbers:
    if num % 2 == 0:
        squared_evens.append(num ** 2)
```

Using a list comprehension:

Copy code

```
squared_evens = [num ** 2 for num in numbers if num % 2 == 0]
```

In this example, the list comprehension is more concise and arguably more readable. Additionally, the list comprehension avoids the overhead of the append method, making it more efficient for large lists.

However, it's essential to note that for more complex operations or when side effects are required (like printing or updating other variables), traditional loops might be more appropriate. Also, for very large datasets, generator expressions (using () instead of []) can be more memory-efficient than list comprehensions.

Q23. What is Monkey Patching? How can you do it in Python?

Ans. Monkey Patching is the process of making changes to a module or class while the program is running. A Monkey Patch is a piece of code that extends or modifies other code at runtime (typically at startup).

Q24. What is a unittest?

Ans. The unit testing framework of Python is known as unittest. It has similar features with unit testing frameworks in other languages.

Unittest supports some important concepts of object-oriented Programming:

- Test fixture
- Test case
- Test suite
- Test runner

Example:

Copy code

```
import unittest

class ABC(unittest.TestCase):

def xyz():

…

if __name__ == "__main__":

unittest.main()
```

Q25. What is a negative index?

Q26. What is the difference between Xrange() and range()?

Ans. Range() returns a list and Xrange() returns an Xrange object, which is kind of like an iterator and generates the numbers on demand.

Example:

```
Copy code

a = range(1,10000)


# initializing a with xrange()

x = xrange(1,10000 testing the type of a

i (“The return type of range() is : “)

print (type(a)

# testing the type of x

print (“The return type of xrange() is : “)

print (type(x))
```

Output:

The return type of range() is :

<type ‘list’>

The return type of xrange() is :

<type ‘xrange’>

Q27. Define module and package.

Ans. A module is a Python object with arbitrarily named attributes that you can bind and reference.

A Python package is simply a directory of Python module(s).

Q28. Why don’t lambda forms have statements?

Ans. It is because a lambda form is used to make a new function object and then return at runtime. Also, the syntactic framework of Python is unable to handle statements nested inside expressions.

Q29. What is Flask?

Ans. Flask (source code) is a Python micro web framework and it does not require particular tools or libraries. It is used for deploying python code into web apps.

Must Check: Python Flask Online Courses and Certifications

Q30. How will you perform static analysis in a Python application or find bugs?

Ans. PyChecker can be helpful as a static analyzer to identify the bugs in the Python project. This also helps to find out the complexity-related bugs. Pylint is another tool that helps check if the Python module is at par with the coding standards.

Ans. Python Decorator is used to adjusting the functions in Python syntax quickly.

Q32. Are Python strings immutable or mutable?

Ans. This is among the very commonly asked Python interview questions.

Python strings are immutable. Ironically, it is not a string, but a variable with a string value.

Q33. What is a pass in Python?

Ans. Pass stands for no-operation Python statement. It means that pass is a null operation; nothing happens when it is executed.

You might come across some confusing Python interview questions, MCQ is one of them, but do not get stumped. You should be thorough with your study and be well prepared for the Python interview questions.

Must Read: Pass Statement in Python

Q34. Now, choose the right answer –When "else" in try-except-else is executed?

1. In case of any exception
2. When no exception is there
3. When an exception occurs in the except block
4. Always

Ans. c) when no exception occurs

Q35. What is slicing?

Ans. Slicing is a computationally fast way to methodically access a range of items from sequence types like list, tuple, strings, etc.

Must Read: Slicing in Python

Q36. What is the output of the following code?

Ans.

```
list1 = [5,4,3,2,1]

list2 = list1

list2[0] = 0;

print "list1= : ", list1
```

Copy code

Output:

list1= : [5,4,3,2,1]

Q37. How is the last object from a list removed?

Ans. list.pop(obj=list[-1]) − Removes and returns last object from the list.

Q38. What is docstring?

Ans. We can delete a file in Python by using a command os.remove (filename) or os.unlink(filename).

Q40. What is the output of the following code?

```
x = True
y = False
z = False
if x or y and z:
print "HELLOWORLD"
else:
print "helloworld"
```

Copy code

Ans. Output:

HELLOWORLD

Reason: Here, in Python AND operator has a higher preference than OR operator. So, (y and z) are evaluated first.

Q41. How many kinds of sequences does Python support? Name them.

Ans. Python supports seven sequence types –

- Str
- List
- Tuple
- Unicode
- byte array
- xrange
- buffer

Q42. How will you reload a Python module?

Ans. reload() is used to reload a previously imported module.

Q43. What is a set?

Ans. A Python set is an unordered collection of iterable and mutable data, and it has no duplicate elements.

Must Read: Sets in Python

Q44. Name some standard Python errors.

Ans. Some standard errors are –

- TypeError
- ValueError

- KeyError

We can use dir(__builtin__) to list all the errors.

Q45. What is Tkinter?

Ans. Tkinter is the de-facto standard GUI (Graphical User Interface) package of Python.

Q46. What is Multithreading?

Ans. Multithreading stands for running a number of programs simultaneously by invoking multiple threads.

Example:

Copy code

```
from threading import *
print(current_thread().getName())
def mt():
    print("Child Thread")
child=Thread(target=mt)
child.start()
print("Executing thread name :",current_thread().getName())
```

Output:

MainThread

Child Thread

Executing thread name : MainThread

Q47. How is a list reversed?

Ans. To reverse lists, one can use list.reverse()

Q48. How to capitalize the first letter of string?

Ans. To capitalize the first letter of the string, capitalize() method is used. If the string is already capitalized then it will return the original value.

Q49. Which python library is used for Machine learning?

Ans. Scikit-learn python Library is used for Machine learning.

Must Read: Top 10 python libraries for Data Science and Machine Learning

Q50. What is the role of len() in python?

Ans. len() is used to determine the length of an array, list and string in the program.

Example:

str1='1234'

len(str1)

Q51. What is the output of the following code?

class Demo:

def __initl__(self, id):

self.id = id

id = 777

acc = Acc(222)

print acc.id

Ans. Output: 222

Reason: “Demo” class automatically calls the method ”initl” and passes the object and “222” is assigned to the object called id. So, the value “777” cannot be called and the output will be “222”.

Q52. How to delete a file in Python?

Ans. OS Module needs to be installed to delete any file. After installing the module, os.remove() function is used to delete a file.

Q53. Write a code to test whether the number is in the defined range or not?

Ans.

def test_range(n1):

if n1 in range(0, 555):

print(”%s is in range”%str(n1))

else:

print(”%s is not in range”%str(n1))

Output:

test_range(555)

555 is not in the range

Q54. Write a code to convert a string into lowercase?

Ans. lower() is used to convert the string into lower case

str=’XYZ’

print(str.lower())

Output: xyz

Must Read: Strings in Python

Q55. What is the output of the following code?

nameList = [‘Joe’, ‘Nick’, ‘Bob’, ‘Harry’]

print nameList[1][-1]

Ans. Output:

k

Reason: [-1] shows the last element or character of the string. In the above code,]1] represents the second string and [-1] represents the last character of the second string, i.e., “k.”

imported to the library to interact with the database.

Q57. What is the output of the following code?

demoCodes = [1, 2, 3, 4]

demoCodes.append([5,6,7,8])

print len(demoCodes)

Ans. Output: 5

Reason: 'append' method is used in the code, which has to append the existing object into the list. But the append method does not merge the list, which is added as an element. So, the output will be'5'.

Q58. What is the use of the '#' symbol in Python?

Ans. '#' symbol is used to symbolize the comments

print ("I am a quick learner")

#print ("I am a quick learner")

Q59. Suppose a list1 is [2, 44, 191, 86], what would be the output for list1[-1]?

Ans. Output: 86

List1[-1] shows the last integer of the list

Q60. What is the maximum length of an identifier?

Ans. The maximum possible length of an identifier in python is 79 characters.

Explore the differences between Python and Java. Read our blog – Python Vs Java – Which One is Better to Learn?

Q61. Can you tell me the generator functions in Python?

Ans. Generator functions help to declare a function that behaves like an iterator in a fast, easy, and neat way.

Q62. Write a code to display the current time.

Ans. Here is the code to represent the current time:

import datetime

now = datetime.datetime.now()

print ("Current date and time : ")

print (now.stgftime("%Y-%m-%d %H:%M:%S"))

Q63. Is Python a case-sensitive programming language?

Ans. Yes, it is a case-sensitive language like other languages such as Java, C, and C++.

Q64. Write a code to sort a numerical list in Python.

Ans. To sort a numerical list, use the following code:

list = ["2", "7", "3", "5", "1"]

list = [int(i) for i in list]

list.sort()

print (list)

Q64. Write a code to display the contents of a file in reverse.

Ans. To reverse the content, use the following code:

Q65. How to add array elements in programming?

Ans. We can add elements to an array with the help of append(), insert (i,y) and extend() functions.

Example:

x=arr.array('d', [1.2 , 2.2 ,3.2])

x.append(3.3)

print(x)

x.extend([4.5,6.2,6.3])

print(x)

x.insert(2,3.8)

print(x)

Output:

array('d', [1.2, 2.2, 3.2, 3.3])

array('d', [1.2, 2.2, 3.2, 3.3, 4.5, 6.2, 6.3])

array('d', [1.2, 2.2, 3.8, 3.2, 3.3, 4.5, 6.2, 6.3])

Q66. Why is the split used in Python?

Ans. The split() method is used to separate two strings.

Example:

x="Naukri learning"

print(x.split())

Output: ['Naukri', 'learning']

Q67. How to create classes in Python?

Ans. Classes are user-defined which is defined with a class keyword

Example:

Class Student:

def_init_(self, name) :

self.name = name

S1 = Student ("xyz")

print (S1.name)

Output: xyz

Must Read: Classes and Objects in Python

Q68. How do we create an empty class?

Ans. An empty class is a blank class that does not have any code defined within its block. We can create an empty class using the pass keyword.

Example:

Class x:

 : pass

obj=x()

obj.name="xyz"

Q69. Explain the difference between a shallow copy and a deep copy.

Ans. Shallow copy is used at the time of new instance creation, and it stores the copied values whereas in deep copy, the copying process executes in looping, and copy of an object is copied in other objects. A shallow copy has faster program execution than a deep copy.

Looking for top resources to learn Python? Read our blog – Why Learn Python? Reasons and Top Resources to Learn Python

Q70. Which statement can we use if the statement is required syntactically, but no action is needed for the program?

Ans. Pass statement is used if the statement is required syntactically, but no action is required for the program

Example:

If(x>20)

print("Naukri")

else

pass

Q71. What are the tools required to unit test your code?

Ans. To test units or classes, we can use the "unittest" python standard library. It is the easiest way to test code, and the features required are similar to the other unit testing tools like TestNG, JUnit.

Q72. How to get indices of N maximum values in a NumPy array?

Ans. With the help of below code, we can get the N maximum values in a NumPy array :

Import numpy as nm

arr=nm.array([1, 6, 2, 4, 7])

print (arr.argsort() [-3:] [::-1])

Output:

[4 6 1]

Must Check: NumPy Interview Question

Q73. How can you use ternary operators (Ternary) in Python?

Ans. Ternary operators are used to display conditional statements. This consists of the true or false values.
Syntax :

The ternary operator is indicated as:

[on_true] if [expression] else [on_false] x, y = 25, 50big = x if x <y else y

Example: The expression is evaluated as if x <and else and, in this case, if x <y is true, then the value is returned as big = x and if it is incorrect then big = y will be returned as a result.

Q74. What does this mean? * args, ** kwargs? Why would we use it?

Ans. * Args is used when you are not sure how many arguments to pass to a function, or if you want to pass a list or tuple of stored arguments to a function.

** kwargs is used when we don't know how many keyword arguments to pass to a function, or it is used to pass the values from a dictionary as the keyword argument.

The args and kwargs identifiers are a convention, you can also use * bob and ** billy but that would not be wise

creating an object model. However, Python can also be treated as a procedural and structural language.

Must Read: OOPs Concept in Python

Q76. What are compilation and linking process in Python?

Ans. Compilation and binding in Python allow new extensions to compile without any errors and binding can only be done when the build procedure passes. If dynamic loading is used, then it depends on the style supplied with the system. The Python interpreter can be used to provide dynamic loading of configuration files and rebuild the interpreter.

For this, the steps required in the process are:

Create a file with a name and in any language, which is compatible with your system compiler, example, file.co file.cpp

Locate the file in the Modules / directory of the distribution you are using.

Add a line in the Setup.local file that is present in the Modules / directory.

Run the file using spam file.o

After successful execution of this rebuild, the interpreter uses the make command in the top-level directory.

If the file is changed, then run rebuildMakefile using command like 'make Makefile'.

Q77. How do I save an image locally using Python whose URL I already know?

Ans. We will use the following code to store an image locally from a URL

import urllib.request

urllib.request.urlretrieve ("URL", "file-name.jpg")

Q78. How can you get the time (age) of the Google cache of any URL or web page?

Ans. Using the following URL format:

http://webcache.googleusercontent.com/search?q=cache:EL-URL-VA-HERE

You should make sure to replace "EL-URL-GO-HERE" with the correct web address of the page or site to get the age of the Google cache.

Example – To check the age of Unipython's Google web cache, you would use the following URL:

http://webcache.googleusercontent.com/search?q=cache:unipython.com

Q79. What is the map function in Python?

Ans. The map function takes two arguments, one is iterable and another is a function and applies the function to each element of the iterable. If the given function accepts more than 1 argument then many iterables are given.

Check out the best Python Courses online

Q80. How are percentages calculated with Python / NumPy?

Ans. Percentages can be calculated using the following code:

import numpy as np

a = np.array ([1,2,3,4,5])

p = np.percentile (a, 50) #Returns 50%.

print (p)

Q81. What is the difference between NumPy and SciPy?

Ans. Both NumPy and SciPy are modules of Python, and they are used for various operations of the data. NumPy stands for Numerical Python while SciPy stands for Scientific Python. The main differences are –

Used for efficient operation on homogeneous data that are stored in arrays	Support operations like integration, differentiation, gradient optimization, etc.
Multi-dimensional array of objects, used for basic operations such as sorting, indexing, and elementary functioning on the array data type	No related array or list concepts as it is more functional
Suitable for computation of data and statistics, and basic mathematical calculation	Suitable for complex computing of numerical data

Q82. How 3D graphics/visualizations are made using NumPy / SciPy?

Ans. Like 2D plotting, 3D graphics are beyond the scope of NumPy and SciPy, but there are packages that can be integrated with NumPy. However, Matplotlib supplies basic 3D plotting in the mplot3d sub-package, while Mayavi offers a host of high-quality 3D viewing features, using the powerful VTK engine.

Explore the concept of Data Science, read our post – what is Data Science?

Q83. What is PYTHONPATH?

Ans. It is an environment variable and is used when importing a module. In addition, PYTHONPATH is used to check the presence of imported modules in some directories. The interpreter uses it to determine which module to load.

Q84. How to install Python on Windows and set a path variable?

Ans. – Install Python from this link: https://www.python.org/downloads/

– After that, install it on your PC. Find the location where Python has been installed on your PC using the following command on the command line: cmd python

– Go to advanced system settings and add a new variable and name it PYTHON_NAME, and paste the copied path

– Find the path variable, select its value and select ‘edit’

– Add a semicolon after the value if it is not present and then write% PYTHON_HOME%

Q85. Is indentation required in Python?

Ans. Indentation is very important in Python. It specifies a block of code. All the code within classes, functions, loops, etc., is specified within an indented block. Generally, this is done using four space characters. If your code is not indented, it will not execute accurately and will throw errors.

Q86. What is the Self in Python?

Ans. The Self in Python is an instance or object of a class. It is explicitly included as the first parameter. This helps distinguish between methods and attributes of a class with local variables.

The variable self in the init method refers to the newly created object while in other methods; it refers to the object whose method was called.

Q87. How does break, continue and pass work?

Ans. Break – It allows the termination of the loop when some condition is met and control is transferred to the next instruction.

Continue – This lets you skip some part of a loop when a specific condition is met and control is transferred to the beginning of the loop.

Pass – It is used when a block of code is needed syntactically, however, its execution needs to be skipped. This is a null operation. Nothing happens when this runs.

Q89. How can you generate random numbers in Python?

Ans. The random module is the standard module for generating a random number. The method is defined as:

import random

random.random

The random.random () declaration returns the floating-point number that is in the range of [0, 1]. The function generates random floating numbers. The methods used with the random class are the bound methods of the hidden instances.

Random instances can be made to display multithreading programs that create a distinct instance of individual threads. The other random generators used are:

randrange (a, b): choose an integer and define the range between [a, b). Returns elements by randomly picking them from the specified range. It does not construct a range object.

Uniform (a, b): select a floating-point number that is defined in the range of [a, b). It returns the floating-point number.

normalvariate (mean, sdev): used for normal distribution where mu is mean and sdev is sigma used for standard deviation.

Q90. What is the Dogpile effect?

Ans. This is one of those difficult Python interview questions to memorize at first, so give it a few tries.

The Dogpile effect occurs when a website's cache has expired and is hit by numerous requests at the same time. This causes a variety of problems, from increased large lag to massive errors. A system called traffic light blocking is used to prevent the effect of Dogpiles.

Q91. Explain what is encapsulation?

Ans. Encapsulation is one of the characteristics of the Python language because it is an object-oriented programming language. Encapsulation is the process of grouping data sets in one and only place. Along with members, encapsulation also returns functions.

Q92. When does Abnormal Termination occur?

Ans. First of all, I should mention that abend or abnormal termination is bad. You don't want it to happen during your programming experience. However, it is practically unavoidable, in one way or another especially when you are a beginner.

Abend is an error in your program during its execution, while the main tasks continue to perform processes. This is caused by a code error or some software problem.

Q93. Does the Python language have a compiler?

Ans. This is one of the most difficult Python interview questions, especially since so many people don't pay attention to it. Python clearly has a compiler, but it is very easy to miss. This is because it works automatically, you won't even notice.

Q94. What is polymorphism in Python?

Ans. Polymorphism is the ability to take many forms, for example, if the parent class has a method called ABC, it means that the child class can have a method with the same name ABC. This contains its own parameters and variables. Polymorphism is allowed in Python.

Q95. How is data abstraction done in Python?

Ans. It can be achieved in Python using abstract classes and interfaces. Data abstraction only supplies the necessary details and hides the implementation.

Q96. Does Python use access specifiers?

Ans. Python does not have access modifiers. Rather it establishes the concept of prefixing the variable, method, or function name with a single or double underscore to mimic the behavior of the private and protected access specifiers.

Q97. Explain the bytes() function in Python.

Ans. The bytes() function in Python returns a bytes object. It converts objects into bytes objects. It also creates empty bytes objects of the specified size.

Q98. Explain the 'with statement'.

Ans. In Python, the 'with statement' is used for exception handling and resource management. It makes the code cleaner and readable as it allows a file to be opened and closed while executing a block of code containing the 'with statement'.

Q99. What is Pandas?

Ans. Pandas is an open-source data analysis and manipulation tool built on top of the Python programming language. It is a Python library that provides fast and high-performance data structures and data analysis tools. It is widely used for data science and machine learning tasks.

Pandas is built on top of Numpy that offers flexibility in creating data structures for data science, enabling you to create multidimensional, tabular, heterogeneous, data structures. It also lets users perform data manipulation and time series. Python with Pandas is used in different domains like analytics, finance, economics, statistics, etc.

Also explore the commonly asked Data Science Interview Questions and Answers

Q100. What is a Pandas Series?

Ans. Pandas Series is a one-dimensional array that can hold data of any type, like integer, string, float, python objects. A Pandas Series is like a column in an excel sheet. It represents a single column in memory, which can either belong to or be independent of a DataFrame.

We can create a Pandas Series using the below constructor −

pandas.Series(data, index, dtype, copy)

Example:

The below code will create a Series from ndarray. We will import a numpy module and use array() function.

```
# import pandas as pd

import pandas as pd

# import numpy as np

import numpy as np

data = np.array(['n','a','u','k','r','i'])

s = pd.Series(data)

print s
```

Output:

```
0  n

1  a

2  u

3  k

4  r
```

Q101. What are Pandas DataFrames?

Ans. Pandas DataFrame is a two-dimensional tabular data structure with labeled axes. The data is aligned in a tabular manner in rows and columns. DataFrames are widely used in data science, machine learning, scientific computing, etc.

Here are some features of Dataframes:

1. 2-dimensional
2. Labeled axes (rows and columns)
3. Size-mutable
4. Arithmetic operations can be performed on rows and columns

Example:

The below code will create a DataFrame using a single list or a list of lists.

```
# import pandas as pd

import pandas as pd

data = [1,2,3,4,5]

df = pd.DataFrame(data)

print df
```

Output:

```
   0
0  1
1  2
2  3
3  4
4  5
```

Must Read: Series vs DataFrame in Pandas

Q102. How to combine DataFrames in Pandas?

Ans. We can combine DataFrames using the following functions:

1. **concat() function:** It is used for vertical stacking.

```
pd.concat([data_frame1, data_frame2])
```

2. **append():** It is used for horizontal stacking of DataDrames.

```
data_frame1.append(data_frame2)
```

3. **join():** It is used to extract data from different DataFrames which have one or more columns common.

```
data_frame1.join(data_frame2)
```

Must Read: Adding Columns to Pandas DataFrame

Q103. How to access the top n rows of a dataframe?

Ans. To access the top n rows of a dataframe, we will use df.head(n).

Q104. How to access the last n rows of a dataframe?

Ans. We will use df.tail(n) to access the last n rows of a dataframe

program are unique and can be used without any conflict. Python maintains a namespace in the form of a Python dictionary. These namespaces are implemented as dictionaries with 'name as key' mapped to its respective 'object as value'. Namespaces have different lifetimes as they are often created at different points in time.

Some of the namespaces in a Python program are:

- **Local Namespace** – it contains local names inside a function. The local namespace is created for a function call and lasts until the function returns.
- **Global Namespace** – It consists of the names from various imported modules that are being used in the ongoing project. This namespace is created when the package is imported into the script and lasts until the script ends.
- **Built-In Namespace** – This namespace contains built-in functions and built-in exception names.

Q106. What is Inheritance in Python?

Ans. Inheritance is the capability of classes in Python to inherit the properties or attributes of another class. A new class is defined with little or no modification to an existing class. The new class is called derived or child class and the class from which it inherits is called the base or parent class. Inheritance provides the code reusability feature. It is transitive, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Python supports the following types of inheritance:

- **Single Inheritance:** A class inherits only one superclass.
- **Multiple Inheritance:** A class inherits multiple superclasses.
- **Multilevel Inheritance:** Features of the base class and the derived class are further inherited into the new derived class. A class inherits a superclass, and then another class inherits the derived class forming a 'parent, child, and grandchild' class structure.
- **Hierarchical inheritance:** More than one derived class are created from a single base class.

Q107. What are Python literals?

Ans. Literals refer to the raw value or data given in a variable or constant. They represent a fixed value in the source code. In Python, there are various types of literals:

1. String literals
2. Numeric literals
3. Boolean literals
4. Literal Collections
5. Special literals
6. **String Literals:** These are created by enclosing text in single or double-quotes. There are two types of Strings Single-line and Multi-line String.

Example: "Literal" , '12345'

2. **Numeric Literals:** They support three types of literals:

- Integer:I=20
- Float: i=3.6
- Complex:2+7j

3. **Boolean Literals:** There are two boolean literals – true and false.
4. **Literal Collections:** There are four different types of literal collections:

- List literals
- Tuple literals

Must Read: Literals in Python

Q108. Write a Python program to produce half pyramid using *.

Ans. Below is the code to print half pyramid using *

```
n = int(input("Enter the number of rows"))
for i in range(0, n):
    for j in range(0, i + 1):
        print("* ", end="")
    print()
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Q109. Write a python program to check if the number given is a palindrome or not

Ans. Below is the code to check if the given number is palindrome or not:

```
num=input("Enter a number:")
if num==num[::-1]
    print ("It is a Palindrome!")
else:
    print("It is not a Palindrome!")
```

Output:

Case 1:

Enter a number: 12321

It is a Palindrome!

Case 2:

Enter number: 5678

It is not a Palindrome!

Must Read: Palindrome in Python

Q110. Write a sorting algorithm for a numerical dataset in Python.

Ans. Below is the sorting algorithm for a numerical dataset in Python

```
list = ["9", "5", "2", "0", "8"]
list = [int(i) for i in list]
list.sort()
print (list)
```

Q111. Explain how can you make a Python Script executable on Unix?

Ans: Python Script file must begin with **#!/usr/bin/env python**