Company Wise Python Interview Questions

Capegemini:

1. Is python interpreted language?

Ans: Yes, Python is an interpreted language. This means that Python code is not compiled into machine code before it is run. Instead, the Python interpreter reads and executes the code directly from the source code files. When you run a Python program, the interpreter parses the code and executes it one line at a time. This makes it easier to write and debug Python code, as you can see the results of your code immediately. However, interpreted languages are generally slower than compiled languages, as the interpreter has to translate the code into machine code at runtime.

2. Difference between List and Tuple.

Ans:

Mutability:
Lists are mutable, which means you can add, remove, or modify elements in a list after it has been created. Tuples, on the other hand, are immutable, which means that once a tuple is created, its contents cannot be changed.
Syntax:
Lists are created using square brackets [], while tuples are created using parentheses ().
Performance:
Tuples are generally faster than lists because they are immutable, so they can be optimized by the interpreter more efficiently. Lists require more memory allocation and deallocation, which makes them slower.
Usage:
Lists are typically used when you need to store a collection of items that may change over time, such as a list of to-do items. Tuples, on the other hand, are typically used when you need to store a collection of items that won't change, such as the coordinates of a point in space.

3. Int to string conversion.

Ans: In Python, you can convert an integer to a string using the str() function.
Example:

```
x = 10
print(str(x))
O/P: “10”
```

4. Difference between is and == in python.

Ans: The == operator is used for value equality. It compares the values of two objects to determine whether they are equal or not.
Example:

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x == y)

# Output: True
```

The is operator, on the other hand, is used for object identity. It checks whether two variables point to the same object in memory.
Example:

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x is y)

# Output: False
```

5. First class object in python.

Ans: In Python, functions are considered first-class objects. This means that they can be treated like any other object, such as an integer, a string, or a list. Specifically, in Python, a function can be: Assigned to a variable Passed as an argument to another function Returned as a value from a function.

Example:

```
# Define a function
def square(x):
    return x ** 2

# Assign the function to a variable
f = square

# Call the function through the variable
print(f(5))  # Output: 25

# Define another function that takes a function as an argument
def apply(func, x):
    return func(x)

# Call the apply function with the square function as an argument
result = apply(square, 6)
print(result)  # Output: 36

# Define yet another function that returns a function
def get_function():
    return square

# Call the get_function function and use the returned function
```

```
f = get_function()
print(f(7))  # Output: 49
```

In this example, the square function is defined and assigned to the variable f. Then, the apply function is defined to take a function as its first argument, and the square function is passed to it. Finally, the get_function function is defined to return the square function, which is then used through the f variable to calculate the square of 7. In summary, functions in Python are first-class objects, which means that they can be assigned to variables, passed as arguments to other functions, and returned as values from functions. This makes Python a very flexible and powerful language for functional programming.

6. Object initialization in python.

Ans: In Python, you can initialize an object using a special method called the constructor. The constructor method is defined with the __init__ method and it is automatically called when a new object is created.

Example:

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

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

# Create a new Person object
person = Person("Alice", 25)

# Call a method on the Person object
person.greet()  # Output: "Hello, my name is Alice and I am 25 years old."
```

To initialize an object of a class, you can simply call the class constructor, passing any necessary arguments. The constructor returns a new object, which you can then assign to a variable. In summary, in Python, you can initialize an object using a constructor method defined in the class with the __init__ method. When a new object is created, the constructor is automatically called to initialize the object's attributes.

7. How to create an empty class in python?

Ans: In Python, you can create an empty class by using the pass statement in the class body. The pass statement is used as a placeholder where code is expected, but no code needs to be executed.

Example:

```
class MyClass:
    pass
```

8. Does python support multiple inheritance?

Ans: Yes, Python supports multiple inheritance, which means that a class can inherit from more than one parent class. To specify multiple inheritance in Python, you simply list the parent classes in the class declaration, separated by commas.

Example:

```
class A:
    def method_a(self):
        print("A")

class B:
    def method_b(self):
        print("B")

class C(A, B):
    def method_c(self):
        print("C")

# Create an object of class C
c = C()

# Call methods from the parent classes
c.method_a()  # Output: "A"
c.method_b()  # Output: "B"
c.method_c()  # Output: "C"
```

In this example, the classes A and B are defined with their own methods, and the class C is defined to inherit from both A and B. The C class defines its own method method_c, which is not present in its parent classes. When an object of class C is created, it inherits the methods from both A and B, in addition to its own methods. The object can call any of these methods, regardless of which parent class they were defined in. Multiple inheritance can be a powerful tool in Python, but it can also make code more complex and difficult to understand. Careful design and organization is necessary when using multiple inheritance to avoid potential conflicts and ambiguities.

9. Django vs flask.

Ans:

Architecture:
Django follows the Model-View-Controller (MVC) pattern, while Flask follows the Model-View-Template (MVT) pattern.

Built-in features:
Django includes many built-in features for handling common web development tasks, while Flask is more lightweight and flexible, and does not include as many built-in features.

Database support:
Django has built-in support for Object-Relational Mapping (ORM) and many popular databases, while Flask supports many databases, but does not include built-in ORM support.

Community:
Django has a large and active community, which means that it is well-documented and has many third-party packages and plugins available. Flask also has a large community, but it is not as large as Django's.

10. Multithreading in python.

Ans: Multithreading is the ability of a program to perform multiple tasks concurrently. In Python, multithreading can be achieved using the threading module. The threading module provides a way to create and manage threads in a Python program.

Example:

```
import threading

# Define a function to be executed in a thread
def print_numbers():
    for i in range(1, 11):
        print(i)

# Create a thread for the function
thread = threading.Thread(target=print_numbers)

# Start the thread
thread.start()

# Do other things in the main thread
for i in range(11, 21):
    print(i)

# Wait for the thread to finish
thread.join()
```

```
print("Done")
```

In this example, a function print_numbers is defined that prints the numbers from 1 to 10. A thread is created for this function using the Thread class from the threading module. The start method is called on the thread object to start the thread, and the join method is called to wait for the thread to finish before continuing. While the thread is running, the main thread continues to execute other code. In this example, the main thread prints the numbers from 11 to 20. When the thread finishes, the program prints "Done".

Note that in Python, multithreading is limited by the Global Interpreter Lock (GIL), which means that only one thread can execute Python bytecode at a time. This can limit the effectiveness of multithreading in certain situations, such as CPU-bound tasks that require a lot of computation. However, multithreading can still be useful for tasks that are I/O-bound, such as network or file I/O, where threads can be used to perform non-blocking I/O operations. In cases where CPU-bound tasks are a concern, multiprocessing or asynchronous programming may be better options.

11. Counter in python.

Ans: Counter is a built-in class in Python's collections module that provides a convenient way to count occurrences of elements in an iterable (e.g., list, tuple, string, etc.). The Counter class creates a dictionary-like object where keys are elements from the iterable and values are the count of those elements.

Example:

```
from collections import Counter

# Create a list of colors
colors = ["red", "blue", "green", "red", "yellow", "blue", "red"]

# Create a Counter object
color_counts = Counter(colors)

# Print the count of each color
print(color_counts)
```

Output:

```
Counter({'red': 3, 'blue': 2, 'green': 1, 'yellow': 1})
```

12. Reverse a list in python.

Ans:

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

```
my_list.reverse()
print(my_list)
```

O/P: [5,4,3,2,1]

Or,

```
x = my_list[::-1]
print(x)
```

O/P: [5,4,3,2,1]

13. Global and local variables.

Ans: A global variable is a variable that is defined outside of any function and can be accessed from anywhere in the program. A local variable, on the other hand, is a variable that is defined inside a function and can only be accessed within that function (i.e., it has local scope).

Example:

```
# Define a global variable
global_var = 10

def my_function():
    # Access the global variable from within the function
    print("The global variable is:", global_var)

my_function()  # Output: The global variable is: 10
```

```
def my_function():
    # Define a local variable
    local_var = 5
    # Access the local variable from within the function
    print("The local variable is:", local_var)

my_function()  # Output: The local variable is: 5
```

In this example, local_var is a local variable that is defined inside the my_function() function. The my_function() function is defined to print the value of the local_var variable. When my_function() is called, it prints the value of local_var, which is 5.

Note that if you try to access a local variable outside of its function, you'll get a NameError:

```
def my_function():
    # Define a local variable
    local_var = 5

my_function()
print(local_var)  # Output: NameError: name 'local_var' is not defined.
```

14. Dynamic typing.

Ans: Dynamic typing is a feature of Python that allows a variable to change its type at runtime. This means that you don't need to declare the type of a variable before using it, and you can reassign a variable to a different type.

15. What is PEP ?

Ans: PEP stands for "Python Enhancement Proposal." It's a document that outlines proposed changes or additions to Python's core functionality or standard library.

PEPs are used by the Python community to suggest new features or changes to the language, and to provide a structured process for discussing and evaluating these proposals.

PEPs cover a wide range of topics, from syntax and language features to library modules and development processes. Some examples of PEPs include:

- PEP 8: Style Guide for Python Code
- PEP 257: Docstring Conventions
- PEP 484: Type Hints
- PEP 3101: Advanced String Formatting

Each PEP goes through a review process that involves discussion and feedback from the community. If a proposal is accepted, it is assigned a number and becomes part of the official Python specification.

PEPs are an important part of the Python ecosystem, as they help to ensure that the language evolves in a coherent and community-driven way. They also provide a way for developers to propose and discuss new ideas, and to shape the future of the language.

16. List comprehension.

Ans: List comprehension is a concise way to create a new list in Python by applying an expression to each element of an existing list or iterable. It is a powerful and efficient way to write code, especially when working with large datasets.

Example:

```
# Create a new list by squaring each element in an existing list
numbers = [1, 2, 3, 4, 5]
squares = [x**2 for x in numbers]

# Print the new list
print(squares)  # Output: [1, 4, 9, 16, 25]
```

17. Lambda function in python.

Ans: In Python, a lambda function is a small anonymous function that can take any number of arguments, but can only have one expression. It is a way to create a function without using the def keyword, and instead uses the lambda keyword.

Example:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# filter the list to only include even numbers
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))

# print the filtered list
print(filtered_list)
```

O/P:
[2, 4, 6, 8, 10]

18. Range vs xrange.

Ans:
range() returns a list of numbers from the starting value to the ending value, incrementing by the step size. The list is generated in memory and stored in its entirety, which means that it can consume a lot of memory if the range is large.

Example:

```
range(0, 10, 2)  # returns [0, 2, 4, 6, 8]
```

xrange() generates the numbers in the sequence on-the-fly as they are needed, rather than generating them all at once and storing them in memory. This makes it much more memory efficient than range(), especially for large ranges.

Example:

```
xrange(0, 10, 2)  # generates 0, 2, 4, 6, 8 on-the-fly
```

In Python 2, there are two built-in functions for generating a sequence of numbers: range() and xrange(). In Python 3, xrange() is no longer available, and the range() function behaves like the xrange() function used to in Python 2.

19. Memory management in python.

Ans:

20. Garbage collector in python.

Ans: In Python, the garbage collector is a mechanism that automatically frees memory that is no longer being used by the program. The garbage collector works by periodically examining the objects in memory and determining which ones are still in use and which ones can be deleted.

Python's garbage collector uses a generational approach to memory management. This means that it divides objects into three generations based on how long they have been in memory. Newly created objects are in the youngest generation, and as they survive longer, they are moved to older generations.

The garbage collector runs more frequently on younger generations, since they are more likely to contain objects that are no longer needed. The older generations are examined less frequently, since they tend to contain objects that have been in use for a longer period of time and are more likely to still be needed.

When the garbage collector runs, it first marks all objects that are still in use. It then frees memory for any objects that are no longer marked. The garbage collector is able to handle circular references, where two or more objects refer to each other in a way that prevents them from being garbage collected.

While the garbage collector is usually very effective at freeing memory that is no longer being used, it is not perfect. In some cases, the garbage collector may not be able to free all unused memory, especially if there are circular references or other complex memory structures in the program. In these cases, it may be necessary to use manual memory management techniques or to modify the program's design to reduce its memory footprint.

Accenture:

1. What is python?

Ans: Python is a high-level, interpreted programming language that was first released in 1991 by Guido van Rossum. It is known for its simple and easy-to-read syntax, as well as its vast collection of libraries and frameworks that make it a popular choice for a wide range of applications, from web development to data science.

Python is an interpreted language, which means that programs written in Python are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, since you can run programs without needing to compile them first.

Python is also a dynamically-typed language, which means that you don't need to declare variable types explicitly. Instead, Python infers the types of variables based on their values at runtime. This can make it faster to write and modify code, since you don't need to worry about type declarations or casting.

One of the strengths of Python is its large and active community of users and developers. There are thousands of libraries and frameworks available for Python, covering everything from scientific computing to web development to machine learning. This makes it easy to find and use third-party code to build complex applications quickly.

Overall, Python is a powerful and versatile language that is well-suited to a wide range of applications, from simple scripting tasks to large-scale software development projects.

2. Why Python?

Ans: There are many reasons why Python has become such a popular programming language, and why it may be a good choice for your next project:

1. Easy to Learn: Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

2. Large Community: Python has a large and active community of developers and users who contribute to a wide range of open-source projects, libraries, and frameworks. This makes it easy to find support, get help with problems, and learn from other people's code.

3. Versatile: Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

4. Interpreted: Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, without needing to worry about compilation.

5. Platform-independent: Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

6. High-level: Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

Overall, Python is a versatile, powerful, and easy-to-learn language that has a large and active community of developers and users. It can be a great choice for a wide range of applications, from small scripts to large-scale software projects.

3. What are the applications of python?

Ans: Python is a versatile programming language that can be used for a wide range of applications, including:

1. Web Development: Python has many popular web frameworks, such as Django and Flask, that make it easy to build complex web applications.

2. Data Science and Machine Learning: Python has become a popular language for data analysis and machine learning, with libraries such as NumPy, Pandas, and TensorFlow.

3. Scripting and Automation: Python is well-suited for writing scripts and automating tasks, such as system administration, file management, and web scraping.

4. Desktop GUI Applications: Python can be used to build graphical user interfaces (GUIs) for desktop applications using libraries such as PyQt and wxPython.

5. Game Development: Python has several game development libraries, such as Pygame and Panda3D, which make it easy to create games.

6. Scientific Computing: Python has a rich collection of scientific computing libraries, such as SciPy, that make it useful for scientific applications.

7. Education: Python's simple and easy-to-read syntax makes it a great language for teaching programming to beginners.

Overall, Python's versatility and large collection of libraries and frameworks make it useful for a wide range of applications, from small scripts to large-scale software projects.

4. What are the advantages of python?

Ans:

1. Easy to Learn: Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

2. Large Community: Python has a large and active community of developers and users who contribute to a wide range of open-source projects, libraries, and

frameworks. This makes it easy to find support, get help with problems, and learn from other people's code.

3. Versatile: Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

4. Interpreted: Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, without needing to worry about compilation.

5. Platform-independent: Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

6. High-level: Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

5. What is PEP8?

Ans: PEP 8 (Python Enhancement Proposal 8) is a style guide for Python code. It was created to promote consistent, readable, and maintainable code across different projects and developers. The style guide covers topics such as naming conventions, whitespace, comments, and formatting.

The goal of PEP 8 is to make Python code more readable and understandable, which can help make it easier to maintain, debug, and modify in the future. By following PEP 8 guidelines, developers can create code that is easier for themselves and others to understand, which can lead to fewer errors and faster development times.

Some of the key guidelines outlined in PEP 8 include:

- Use four spaces to indent code blocks.
- Use underscores for variable and function names, and use camelCase for class names.
- Limit lines to 79 characters.
- Use whitespace to separate logical sections of code.
- Use comments to explain complex sections of code.
- Use docstrings to document functions, modules, and classes.

Overall, PEP 8 is an important resource for Python developers who want to write clean, consistent, and maintainable code. By following these guidelines, developers can create code that is easier to read, understand, and modify.

6. What do you mean by python literals?

Ans: In Python, literals are used to represent fixed values in the source code. They are constant values that are assigned to variables or used in expressions. Python supports several types of literals, including:

1. Numeric literals: Numeric literals represent numbers and can be integer, floating-point, or complex. For example, 10, 3.14, and 2 + 3j are numeric literals.

2. String literals: String literals represent a sequence of characters enclosed in quotes, either single or double. For example, "hello" and 'world' are string literals.

3. Boolean literals: Boolean literals represent either True or False. For example, True and False are boolean literals.

4. Sequence literals: Sequence literals represent a group of values of the same type, such as lists, tuples, and sets. For example, [1, 2, 3] is a list literal, (1, 2, 3) is a tuple literal, and {1, 2, 3} is a set literal.

5. Dictionary literals: Dictionary literals represent key-value pairs enclosed in curly braces. For example, {'a': 1, 'b': 2} is a dictionary literal.

Literals are used throughout Python code and are a fundamental part of the language. They are used to represent values that do not change during the execution of the program, and are a simple and efficient way to define and use fixed values in your code.

7. Explain python functions.

Ans: In Python, a function is a reusable block of code that performs a specific task. It can take input arguments and return output values, and can be called from anywhere in the code. Functions are a fundamental building block of modular programming, which promotes code reuse, simplicity, and maintainability.

In Python, functions are defined using the "def" keyword, followed by the function name and a set of parentheses containing the input arguments. The body of the function is indented and can contain any number of statements. The "return" keyword is used to return a value from the function.

Example:

```
def add_numbers(a, b):
    sum = a + b
    return sum

add_numbers(5,6)
```

O/P: 11

8. What is zip() function in python?

Ans: In Python, the built-in "zip()" function is used to combine two or more iterables into a single iterator of tuples. The resulting iterator contains tuples where the i-th tuple contains the i-th element from each of the input iterables.

Example:

```
list1 = [1, 2, 3]
list2 = ['a', 'b', 'c']
result = zip(list1, list2)

print(list(result))  # Output: [(1, 'a'), (2, 'b'), (3, 'c')]
```

9. What is python's parameter passing mechanism?

Ans:

10. How to overload constructors or methods in python?

Ans: Python does not have the concept of method or constructor overloading like some other languages, such as Java or C++. However, you can achieve similar functionality in Python by using default argument values and variable-length argument lists.

Here's an example of how to use default argument values to simulate constructor overloading:

```python
class MyClass:
    def __init__(self, arg1=None, arg2=None):
        if arg1 is not None and arg2 is not None:
            # Handle case when both arguments are provided
            self.arg1 = arg1
            self.arg2 = arg2
        elif arg1 is not None:
            # Handle case when only arg1 is provided
            self.arg1 = arg1
            self.arg2 = 'default_value'
        else:
            # Handle case when neither argument is provided
            self.arg1 = 'default_value'
            self.arg2 = 'default_value'
```

In this example, the `__init__()` constructor has two arguments, `arg1` and `arg2`, both of which have default values of `None`. If both arguments are provided, the constructor sets the `arg1` and `arg2` attributes of the object to the provided values. If only `arg1` is provided, the constructor sets `arg1` to the provided value and `arg2` to a default value. If neither argument is provided, both `arg1` and `arg2` are set to default values.

Here's an example of how to use variable-length argument lists to simulate method overloading:

```python
class MyClass:
    def my_method(self, *args):
        if len(args) == 0:
            # Handle case when no arguments are provided
            pass
        elif len(args) == 1:
            # Handle case when one argument is provided
            arg1 = args[0]
            # Do something with arg1
        elif len(args) == 2:
            # Handle case when two arguments are provided
            arg1 = args[0]
            arg2 = args[1]
            # Do something with arg1 and arg2
        else:
            # Raise an error for unsupported number of arguments
            raise TypeError('Unsupported number of arguments')
```

In this example, the `my_method()` method takes a variable-length argument list `*args`, which allows it to accept any number of arguments. The method then checks the length of the argument list to determine which code path to take, based on the number of arguments provided.

Note that while this approach can simulate method overloading, it can also make code harder to read and maintain, so it should be used sparingly.

11. What is the difference between remove() function and dell statement?

Ans: In Python, both the `remove()` function and the `del` statement are used to remove elements from a list, but there are some differences between them:

`remove()` function:

The `remove()` function is a list method that removes the first occurrence of the specified element from the list. It modifies the list in-place and does not return any value. If the element is not found in the list, it raises a `ValueError` exception.

Example:

```
>>> lst = [1, 2, 3, 4, 2, 5]
>>> lst.remove(2)
>>> print(lst)
[1, 3, 4, 2, 5]
```

`del` statement:

The `del` statement is a general statement in Python that can be used to remove an element or a slice of elements from a list. It can also be used to remove variables and other objects from the current namespace. The `del` statement does not return any value.

Example:

```
>>> lst = [1, 2, 3, 4, 2, 5]
>>> del lst[1]
>>> print(lst)
[1, 3, 4, 2, 5]
```

So, the main difference between `remove()` function and `del` statement is that `remove()` removes the first occurrence of a specified element from the list, whereas `del` removes the element at a specified index or a slice of elements from the list.

12. What is swapcase() function in python?

Ans: The `swapcase()` function is a built-in function in Python which returns a new string by swapping the case of all the characters in the given string. The uppercase characters are converted to lowercase and the lowercase characters are converted to uppercase.

Syntax: `string.swapcase()`

Example:

```python
string = "Hello, World!"
new_string = string.swapcase()
print(new_string)
```

Output:

```
hELLO, wORLD!
```

13. How to remove whitespace from a string in python?

Ans:

In Python, there are several ways to remove whitespace from a string:

1. Using `strip()` function: This function removes whitespace (spaces, tabs, and newlines) from the beginning and end of the string.

```python
string = "  Hello, World!   \n"
new_string = string.strip()
print(new_string)  # Output: "Hello, World!"
```

14. How to remove leading whitespace from a string in python?

Ans: In Python, you can remove leading whitespace (spaces, tabs, and newlines) from a string using the `lstrip()` method. Here's an example:

```python
string = "   Hello, World!  "
new_string = string.lstrip()
print(new_string)  # Output: "Hello, World!  "
```

The `lstrip()` method returns a new string with all leading whitespace characters removed. If you want to remove only specific characters from the beginning of the string, you can pass those characters as an argument to the `lstrip()` method. Here's an example:

```python
string = "***Hello, World!***"
new_string = string.lstrip("*")
print(new_string)  # Output: "Hello, World!***"
```

In this example, the `lstrip("*")` method removes all the asterisks from the beginning of the string.

15. Why do we use join() function in python?

Ans: We use the `join()` function in Python to concatenate strings together. It takes an iterable (e.g., a list or tuple) of strings as an argument and returns a single string where each item in the iterable is separated by the string on which `join()` is called.

```python
words = ["apple", "banana", "orange"]
joined_string = ", ".join(words)
print(joined_string)  # Output: "apple, banana, orange"
```

In this example, the `join()` function is used to concatenate the words in the list `words` into a single string, separated by commas. The resulting string is then stored in the variable `joined_string`.

16. GIve an example of shuffle() method?

Ans: The `shuffle()` method in Python is used to shuffle a list randomly. Here is an example:

```python
import random

my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)
```

Output:
```
[4, 2, 5, 1, 3]
```

In this example, we imported the `random` module and defined a list `my_list` with five elements. Then we used the `shuffle()` method of the `random` module to shuffle the list randomly. Finally, we printed the shuffled list.

17. What is the use of a break statement?

Ans: The break statement in Python is used to terminate the loop statement in which it is present. When the break statement is encountered inside a loop, the loop is terminated and the program execution continues with the statement immediately following the loop.

Example:

```
i = 1
while i <= 10:
    if i > 5:
        break
```

```
    print(i)
    i += 1
```

O/P:
1
2
3
4
5

18. What is tuple in python?

Ans: In Python, a tuple is an ordered, immutable sequence of values, similar to a list. However, unlike a list, a tuple cannot be modified once it is created. Tuples are defined using parentheses () or the tuple() constructor, with elements separated by commas. For example:

```python
my_tuple = (1, 2, 3, 'a', 'b', 'c')
```

Tuples can contain elements of different data types and can be nested. Tuples are commonly used for returning multiple values from a function or for data that should not be changed, such as coordinates or constant values. Tuples support indexing and slicing like lists.

19. Which are the file related libraries/modules in python?

Ans:

Python has several built-in modules and libraries for file-related operations. Some of the commonly used ones are:

1. `os` module: This module provides a way of interacting with the operating system. It can be used to perform operations like creating, deleting, and renaming files and directories.

2. `os.path` module: This module provides functions to manipulate paths in a platform-independent way. It can be used to get the current working directory, join two paths, split a path, and more.

3. `glob` module: This module is used to retrieve files and directories matching a specified pattern. It supports Unix-style pathname patterns and can be used to search for files with a specific extension or matching a particular string.

4. `shutil` module: This module provides a higher level interface for file operations like copying, moving, and deleting files and directories. It supports copying entire directory trees and also provides functions to work with file permissions.

5. `csv` module: This module provides functionality to read from and write to CSV files. It supports various dialects of CSV files and provides options for handling headers and quoting characters.

6. `json` module: This module provides functions to read and write data in JSON format. It supports encoding and decoding of Python objects to and from JSON format.

7. `pickle` module: This module is used for object serialization and deserialization. It can be used to convert Python objects into a stream of bytes, which can then be stored in a file or transmitted over a network.

20. What are the different file processing modes supported by python?

Ans: Python supports several file processing modes, which determine how the file will be opened and accessed. The most commonly used file processing modes in Python are:

1. **Read mode ("r")**: This mode is used to read data from an existing file. The file pointer is placed at the beginning of the file, and if the file does not exist, an error is raised.

2. **Write mode ("w")**: This mode is used to write data to a file. If the file does not exist, it is created. If the file already exists, its contents are truncated and overwritten.

3. **Append mode ("a")**: This mode is used to append data to an existing file. The file pointer is placed at the end of the file, so any data written is added to the end of the file.

4. **Binary mode ("b")**: This mode is used to read or write binary data, such as images or audio files. It can be used with any of the above modes.

5. **Read and write mode ("r+")**: This mode is used to read and write data to an existing file. The file pointer is placed at the beginning of the file, and any data written will overwrite the existing data.

6. **Write and read mode ("w+")**: This mode is used to create a new file for reading and writing. If the file already exists, its contents are truncated and overwritten. The file pointer is placed at the beginning of the file.

7. **Append and read mode ("a+")**: This mode is used to open a file for both reading and appending. The file pointer is placed at the end of the file, so any data written is added to the end of the file, and any data read is read from the beginning of the file.

TCS:

1. Is there scanf() or sscanf() equivalent.

   Ans:

2. What is negative index?

   Ans: In Python, negative index refers to accessing elements of a sequence (such as a string, list, tuple, etc.) from the end instead of the beginning. It means that the last element has an index of -1, the second last has an index of -2, and so on. For example, if we have a list `my_list = [1, 2, 3, 4, 5]`, then `my_list[-1]` will return `5`, `my_list[-2]` will return `4`, and so on. Negative indexing can be useful when we need to access the last few elements of a sequence without knowing the exact length.

3. How do you make an array in python?

   Ans: In Python, you can create an array using the `array` module. Here's an example:

   ```python
   import array

   # Create an array of integers
   arr_int = array.array('i', [1, 2, 3, 4, 5])

   # Create an array of floats
   arr_float = array.array('f', [1.0, 2.0, 3.0, 4.0, 5.0])
   ```

   In the example above, we create two arrays, one of integers and one of floats. The first argument to `array` is the type code, which determines the type of the elements in the array. Here are some of the type codes you can use:

   - `b`: signed integer of size 1 byte
   - `B`: unsigned integer of size 1 byte
   - `i`: signed integer of size 2 bytes
   - `I`: unsigned integer of size 2 bytes
   - `l`: signed integer of size 4 bytes
   - `L`: unsigned integer of size 4 bytes
   - `f`: floating point number of size 4 bytes
   - `d`: floating point number of size 8 bytes

You can also create arrays of other types by using the appropriate type code. Once you have created an array, you can access its elements using indexing, just like with a list:

```python
print(arr_int[0])  # Output: 1
print(arr_float[2])  # Output: 3.0
```

4. What is self?

Ans: In Python, `self` is a reference to the current instance of a class. It is the first parameter of any instance method defined in a class. When an instance method is called on an object, Python automatically passes the reference to the object as the first argument, which is then captured by the `self` parameter.

Using the `self` parameter, you can access the attributes and methods of the current instance, and modify or retrieve their values. The `self` parameter is used to differentiate between the instance variables of a class and the local variables of a method with the same name.

For example, consider the following class definition:

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

    def get_name(self):
        return self.name

    def get_age(self):
        return self.age
```

In the `__init__` method, the `self` parameter refers to the instance of the `Person` class that is being created. The `name` and `age` parameters are used to initialize the `name` and `age` instance variables of the object, respectively.

In the `get_name` and `get_age` methods, the `self` parameter is used to access the `name` and `age` instance variables of the object, respectively.

5. Where is the math.py (socket.py, regex.py, etc) source file?

Ans: The source code for built-in Python modules such as math, socket, regex, etc. is typically written in C or C++, not in Python. These modules are compiled and included in the Python standard library. You can view the source code for these

modules in the CPython GitHub repository, which is the reference implementation of Python. However, it is worth noting that the implementation of these modules may vary between different Python implementations, such as PyPy or Jython.

6. How do i read (or write) binary data?

Ans: To read or write binary data in Python, you can use the `open()` function with the appropriate mode, either "rb" for reading binary data or "wb" for writing binary data.

Here is an example of reading binary data from a file:

```
with open('file.bin', 'rb') as f:
    data = f.read()
```

And here is an example of writing binary data to a file:

```
data = b'\x00\x01\x02\x03'
with open('file.bin', 'wb') as f:
    f.write(data)
```

Note that binary data is represented as a byte string (`b'...'`) in Python, and the `read()` and `write()` methods of a file handle work with byte strings when the file is opened in binary mode.

7. Are there any interfaces to database packages in python?

Ans: Yes, Python provides various interfaces to work with databases. Some popular database interfaces in Python include:

1. Python DB-API: It is a standard Python module that defines a common interface to relational databases. It provides a consistent way to access different database management systems, such as MySQL, PostgreSQL, SQLite, Oracle, etc.

2. SQLAlchemy: It is a popular Object-Relational Mapping (ORM) library that allows users to interact with databases using high-level Python objects.

3. Django ORM: It is a part of the Django web framework and provides an easy-to-use ORM layer for database interactions.

4. PyMongo: It is a Python driver for MongoDB, a NoSQL database system.

These interfaces allow you to connect to a database, execute SQL queries, and retrieve data from the database.

8. How can I execute arbitrary python statement from C?

   Ans:
9. How can I evaluate an arbitrary python statement from C?

   Ans:

10. How do I debug an extension?

    Ans: Debugging extensions in Python can be done using various tools and techniques. Here are a few options:

    1. `gdb` (GNU debugger): You can attach a debugger to your Python process and use `gdb` to inspect the extension's code and variables. Here is an example command to start a Python process with `gdb`: `gdb --args python myscript.py`. Once you have attached to the process, you can use various `gdb` commands like `break`, `step`, `continue`, `print`, and `backtrace` to debug the extension.

    2. `pdb` (Python debugger): You can also use `pdb` to debug Python extensions. To do this, you need to modify the C code of your extension to include the line `import pdb; pdb.set_trace()` at the point where you want to start debugging. This will start a `pdb` session when your extension is executed, and you can use `pdb` commands like `n` (next), `s` (step), `c` (continue), `p` (print), and `q` (quit) to debug the code.

    3. `valgrind`: If you suspect that your extension has memory issues, you can use a tool like `valgrind` to detect memory leaks, buffer overflows, and other memory-related errors. You can run your Python process under `valgrind` using the command `valgrind --tool=memcheck python myscript.py`.

    4. Logging: You can also use logging to debug your Python extension. You can use the `logging` module in Python to log messages at various levels of severity, and then inspect the log files to understand what's happening in your code. You can use `logging.debug()`, `logging.info()`, `logging.warning()`, `logging.error()`, and `logging.critical()` to log messages at different levels.

    5. Print statements: Finally, you can also use print statements to debug your Python extension. You can use `printf` statements in your C code to print out values of variables, and then inspect the output to understand what's happening in your code. This technique is not as powerful as using a debugger, but it can be useful in certain situations.

Infosys:

1. What is Python?

Ans: Python is a high-level, interpreted programming language that was first released in 1991 by Guido van Rossum. It is known for its simple and easy-to-read syntax, as well as its vast collection of libraries and frameworks that make it a popular choice for a wide range of applications, from web development to data science.

Python is an interpreted language, which means that programs written in Python are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, since you can run programs without needing to compile them first.

Python is also a dynamically-typed language, which means that you don't need to declare variable types explicitly. Instead, Python infers the types of variables based on their values at runtime. This can make it faster to write and modify code, since you don't need to worry about type declarations or casting.

One of the strengths of Python is its large and active community of users and developers. There are thousands of libraries and frameworks available for Python, covering everything from scientific computing to web development to machine learning. This makes it easy to find and use third-party code to build complex applications quickly.

Overall, Python is a powerful and versatile language that is well-suited to a wide range of applications, from simple scripting tasks to large-scale software development projects.

2. Why can’t I use an assignment in an expression?

Ans: In Python, an assignment statement is used to assign a value to a variable. The statement has the form `variable = expression`, where `expression` can be any valid Python expression that evaluates to a value. The assignment statement assigns the value of `expression` to `variable`.

However, an assignment statement is not a valid expression in Python. This means that you cannot use an assignment statement inside another expression. For example, the following code is not valid:

```
x = 2
y = (x = 3)   # invalid syntax
```

The reason for this is that an assignment statement does not return a value in Python. Instead, it simply assigns a value to a variable. Therefore, using an assignment statement inside an expression does not make sense, as it does not produce a meaningful result.

If you want to modify the value of a variable inside an expression, you can use the augmented assignment operators such as `+=`, `-=` etc. These operators combine an arithmetic operation with an assignment, so that you can modify the value of a variable in a single expression. For example:

```
x = 2
x += 3   # equivalent to x = x + 3
```

In this example, the `+=` operator adds 3 to the value of `x` and assigns the result back to `x`.

3. Is there a tool to help find bugs or perform static analysis?

Ans: Yes, there are several tools available in Python for finding bugs and performing static analysis. Here are some popular ones:

1. PyLint: PyLint is a widely used tool for static analysis of Python code. It analyzes the code for potential bugs, errors, and style issues, and generates a report with suggestions for improvements. PyLint checks the code against a set of predefined coding standards and can also be customized to fit specific project requirements.

2. Flake8: Flake8 is a popular tool for static analysis of Python code that combines several other tools, including PyFlakes, PEP8, and McCabe complexity checker. Flake8 checks the code for potential bugs, errors, and style issues and generates a report with suggestions for improvements. It also includes a McCabe complexity checker to identify functions that may be too complex.

3. PyCharm: PyCharm is an Integrated Development Environment (IDE) for Python that includes several built-in tools for finding and fixing errors in code. PyCharm includes a code inspection tool that analyzes the code for potential bugs, errors, and style issues, and generates a report with suggestions for improvements. It also includes a debugger and profiler for identifying and resolving runtime issues.

4. Bandit: Bandit is a tool for finding security vulnerabilities in Python code. It checks the code for potential security issues, such as cross-site scripting (XSS) and SQL injection, and generates a report with suggestions for improvements.

These are just a few examples of the many tools available for finding bugs and performing static analysis in Python. Depending on your specific needs, there may be other tools that are more suitable for your project.

4. Pychecker and Pylint.

Ans: PyChecker and Pylint are both Python tools that can be used to check code for potential errors and style violations. However, they use different methods to accomplish this.

PyChecker is a static analysis tool that scans Python code and checks for common programming errors such as undefined variables and unused imports. It does this by analyzing the source code without actually running it. PyChecker produces a list of warnings and errors, which can be used to identify potential problems in the code. PyChecker is lightweight and easy to use, but it has some limitations, such as not being able to catch all runtime errors.

Pylint, on the other hand, is a more comprehensive code analysis tool that not only checks for potential errors but also enforces a set of coding conventions and style guidelines. It uses a combination of static analysis and dynamic analysis to identify problems in the code. Pylint checks for issues such as syntax errors, variable naming conventions, and code complexity, and generates a detailed report with suggestions for improvement. Pylint is highly configurable, and its output can be customized to match the specific needs of a project.

Overall, PyChecker and Pylint are both valuable tools for Python developers, but they serve slightly different purposes. PyChecker is a lightweight tool for quickly identifying potential programming errors, while Pylint is a more comprehensive code analysis tool that can help enforce coding standards and improve code quality.

5. How do you set a global variable in a function?

Ans: A global variable is defined outside of any function, and its scope is not limited to a particular function but extends throughout the entire program. Any function can access and modify the global variable. To modify a global variable within a function, you must use the global keyword to indicate that the variable is global and not local. If a variable is referenced in a function, and it is not defined within the function, Python will look for it in the global scope.

Example:

```
x = 10  # global variable

def my_func():
    global x
    x = 20  # modify the global variable
    print(x)

my_func()   # prints 20
print(x)    # prints 20
```

6. What are the rules for local and global variable in python?

Ans: In Python, a variable's scope determines where it can be accessed from within a program. There are two main types of variable scopes in Python:

Local Variables:

A local variable is defined within a function, and its scope is limited to that function. The variable can be accessed only within that function and not from outside of it. Once the function execution is completed, the memory allocated to the local variable is released.

Example:

```
def my_func():
    x = 10   # local variable
    print(x)

my_func()   # prints 10
```

Global Variables:
A global variable is defined outside of any function, and its scope is not limited to a particular function but extends throughout the entire program. Any function can access and modify the global variable. To modify a global variable within a function, you must use the global keyword to indicate that the variable is global and not local. If a variable is referenced in a function, and it is not defined within the function, Python will look for it in the global scope.

Example:

```
x = 10  # global variable

def my_func():
    global x
    x = 20  # modify the global variable
    print(x)

my_func()   # prints 20
print(x)    # prints 20
```

7. How do i share global variables across modules?

Ans: In Python, you can share global variables across modules by importing the module that contains the global variables into the module that needs to access them. Here's an example:

module1.py:
```
global_var = 10
```

module2.py:
```
```

```
from module1 import global_var

def print_global_var():
    print(global_var)
```

In this example, we have a global variable called `global_var` defined in `module1.py`. We then import this variable into `module2.py` using the `from module1 import global_var` statement. We can then access the `global_var` variable from within `module2.py` by referencing it directly.

Note that it is generally considered best practice to limit the use of global variables as much as possible, as they can make your code more difficult to read and maintain. If possible, consider passing variables between modules as function arguments or using other methods of encapsulation to limit the scope of your variables.

8. How can I pass optional or keywords parameters from one function to another?

Ans: In Python, you can pass optional or keyword parameters from one function to another using the `*args` and `**kwargs` syntax.

The `*args` syntax allows you to pass a variable number of non-keyword arguments to a function. Here's an example of how to pass `*args` from one function to another:

```
def foo(*args):
    bar(*args)

def bar(*args):
    for arg in args:
        print(arg)
```

In this example, the `foo` function accepts any number of arguments using the `*args` syntax. It then calls the `bar` function and passes the `*args` parameter to it. The `bar` function simply loops over each argument and prints it.

The `**kwargs` syntax allows you to pass a variable number of keyword arguments to a function. Here's an example of how to pass `**kwargs` from one function to another:

```
def foo(**kwargs):
    bar(**kwargs)

def bar(**kwargs):
    for key, value in kwargs.items():
        print(key, value)
```

```

In this example, the `foo` function accepts any number of keyword arguments using the `**kwargs` syntax. It then calls the `bar` function and passes the `**kwargs` parameter to it. The `bar` function loops over each key-value pair in the `kwargs` dictionary and prints it.

Note that you can also use both `*args` and `**kwargs` in the same function definition to accept both non-keyword and keyword arguments.

9. How do you make a higher order function in python?

Ans: In Python, a higher-order function is a function that takes one or more functions as arguments, or returns a function as its result. Here's an example of how to create a higher-order function in Python:

```
def double(func):
    def inner(x):
        return 2 * func(x)
    return inner

def add(x):
    return x + 1

new_func = double(add)
result = new_func(3) # result will be 8
```

In this example, the `double` function is a higher-order function that takes a function as its argument. It returns a new function, `inner`, which takes a single argument `x`, and applies the input function `func` to it, doubling its result. The `add` function is a simple function that returns its input value plus 1.

To create a new function that doubles the result of `add`, we call `double(add)`, which returns the `inner` function that doubles its input. We then call `new_func(3)`, which first calls `add(3)` (resulting in 4), and then doubles the result using `inner`, resulting in a final result of 8.

Note that higher-order functions can be used to create powerful and flexible code by abstracting away common patterns and allowing functions to be composed and customized in various ways.

10. How do I copy an object in python?

Ans: In Python, there are two ways to copy an object: shallow copy and deep copy.
```

1. Shallow Copy: In shallow copy, a new object is created which is a reference to the original object. Any changes made to the original object will also be reflected in the copied object.

There are two ways to create a shallow copy of an object:

a. Using the slicing operator [:]

Example:

```
original_list = [1, 2, 3, 4]
copied_list = original_list[:]
```

b. Using the copy() method

Example:

```
import copy
original_list = [1, 2, 3, 4]
copied_list = original_list.copy()
```

2. Deep Copy: In deep copy, a new object is created which is a completely independent copy of the original object. Any changes made to the original object will not be reflected in the copied object.

To create a deep copy of an object, we can use the deepcopy() function from the copy module.

Example:

```
import copy
original_list = [[1, 2], [3, 4]]
copied_list = copy.deepcopy(original_list)
```

In this example, a deep copy of the original list is created, which means that any changes made to the original list will not be reflected in the copied list.

11. How can I find the methods or attributes of an object?
12. How do I convert a string to a number?

Ans: To convert a string to a number in Python, you can use one of the following built-in functions:

1. int(): to convert a string to an integer.

   Example: `num = int("42")`

2. float(): to convert a string to a floating-point number.

   Example: `num = float("3.14")`

3. complex(): to convert a string to a complex number.

   Example: `num = complex("2+3j")`

Note that if the string contains characters that cannot be converted to a number, you will get a ValueError. Therefore, it is a good practice to use a try-except block to handle such cases.

Example:

```
str_num = "123"
try:
    num = int(str_num)
    print(num)
except ValueError:
    print("Invalid number")
```

In this example, the string "123" is converted to an integer using the `int()` function. If the string cannot be converted to a number, a ValueError is raised and caught by the `except` block.

IBM:

1. What is python?

   Ans: Python is a high-level, interpreted programming language that was first released in 1991 by Guido van Rossum. It is known for its simple and easy-to-read syntax, as well as its vast collection of libraries and frameworks that make it a popular choice for a wide range of applications, from web development to data science.

   Python is an interpreted language, which means that programs written in Python are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, since you can run programs without needing to compile them first.

Python is also a dynamically-typed language, which means that you don't need to declare variable types explicitly. Instead, Python infers the types of variables based on their values at runtime. This can make it faster to write and modify code, since you don't need to worry about type declarations or casting.

One of the strengths of Python is its large and active community of users and developers. There are thousands of libraries and frameworks available for Python, covering everything from scientific computing to web development to machine learning. This makes it easy to find and use third-party code to build complex applications quickly.

Overall, Python is a powerful and versatile language that is well-suited to a wide range of applications, from simple scripting tasks to large-scale software development projects.

2. What are the benefits of Python?

Ans:

1. Easy to Learn: Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

2. Large Community: Python has a large and active community of developers and users who contribute to a wide range of open-source projects, libraries, and frameworks. This makes it easy to find support, get help with problems, and learn from other people's code.

3. Versatile: Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

4. Interpreted: Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, without needing to worry about compilation.

5. Platform-independent: Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

6. High-level: Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

3. What are the key features of python?

Ans:

1. Easy to Learn: Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

2. Large Community: Python has a large and active community of developers and users who contribute to a wide range of open-source projects, libraries, and frameworks. This makes it easy to find support, get help with problems, and learn from other people's code.

3. Versatile: Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

4. Interpreted: Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, without needing to worry about compilation.

5. Platform-independent: Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

6. High-level: Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

4. What type of language is python? Programming or scripting?

Ans: Python is both a programming language and a scripting language.

As a programming language, Python provides a rich set of features and tools for building complex software applications. It supports object-oriented, imperative, and functional programming styles, and has a large standard library and an active ecosystem of third-party libraries and frameworks.

As a scripting language, Python is often used for rapid prototyping, automation, and glue code that connects different software components. Python scripts can be run directly from the command line or executed within a larger software system. Python's interactive interpreter also allows developers to experiment with code snippets and explore data interactively.

Python's flexibility as both a programming language and a scripting language has contributed to its popularity and widespread adoption in a variety of domains, including scientific computing, web development, data analysis, machine learning, and more.

5. What are the applications of python?

Ans:

1. Web Development: Python has many popular web frameworks, such as Django and Flask, that make it easy to build complex web applications.

2. Data Science and Machine Learning: Python has become a popular language for data analysis and machine learning, with libraries such as NumPy, Pandas, and TensorFlow.

3. Scripting and Automation: Python is well-suited for writing scripts and automating tasks, such as system administration, file management, and web scraping.

4. Desktop GUI Applications: Python can be used to build graphical user interfaces (GUIs) for desktop applications using libraries such as PyQt and wxPython.

5. Game Development: Python has several game development libraries, such as Pygame and Panda3D, which make it easy to create games.

6. Scientific Computing: Python has a rich collection of scientific computing libraries, such as SciPy, that make it useful for scientific applications.

6. What is the difference between List and Tuple in python?

Ans:

Mutability:
Lists are mutable, which means you can add, remove, or modify elements in a list after it has been created. Tuples, on the other hand, are immutable, which means that once a tuple is created, its contents cannot be changed.
Syntax:
Lists are created using square brackets [], while tuples are created using parentheses ().
Performance:
Tuples are generally faster than lists because they are immutable, so they can be optimized by the interpreter more efficiently. Lists require more memory allocation and deallocation, which makes them slower.
Usage:
Lists are typically used when you need to store a collection of items that may change over time, such as a list of to-do items. Tuples, on the other hand, are typically used when you need to store a collection of items that won't change, such as the coordinates of a point in space.

7. What are global and local variables in python?

Ans: In Python, a variable's scope determines where it can be accessed from within a program. There are two main types of variable scopes in Python:

Local Variables:
A local variable is defined within a function, and its scope is limited to that function. The variable can be accessed only within that function and not from outside of it. Once the function execution is completed, the memory allocated to the local variable is released.

Example:

```
def my_func():
    x = 10   # local variable
    print(x)

my_func()   # prints 10
```

Global Variables:
A global variable is defined outside of any function, and its scope is not limited to a particular function but extends throughout the entire program. Any function can access and modify the global variable. To modify a global variable within a function, you must use the global keyword to indicate that the variable is global and not local. If a variable is referenced in a function, and it is not defined within the function, Python will look for it in the global scope.

Example:

```
x = 10  # global variable

def my_func():
    global x
    x = 20  # modify the global variable
    print(x)

my_func()   # prints 20
print(x)    # prints 20
```

8. Define Python path?
9. What are the two major loop statements?

Ans: The two major loops in Python are the `for` loop and the `while` loop.

`for` loop:

A `for` loop is used to iterate over a sequence (such as a list, tuple, or string) or other iterable objects (such as a dictionary, set, or generator) and perform some operations on each item in the sequence.

The syntax of the `for` loop is as follows:

```python
for variable in sequence:
    # do something with variable
```

`while` loop:
A `while` loop is used to repeatedly execute a block of code as long as a particular condition is true.

The syntax of the `while` loop is as follows:

```python
while condition:
    # do something
```

In a `while` loop, the condition is evaluated before each iteration. If the condition is true, the loop body is executed. If the condition is false, the loop terminates.

Both `for` and `while` loops are powerful constructs that are commonly used in Python programming to control the flow of execution and automate repetitive tasks.

10. What is PEP8?

Ans: PEP 8 (Python Enhancement Proposal 8) is a style guide for Python code. It was created to promote consistent, readable, and maintainable code across different projects and developers. The style guide covers topics such as naming conventions, whitespace, comments, and formatting.

The goal of PEP 8 is to make Python code more readable and understandable, which can help make it easier to maintain, debug, and modify in the future. By following PEP 8 guidelines, developers can create code that is easier for themselves and others to understand, which can lead to fewer errors and faster development times.

Some of the key guidelines outlined in PEP 8 include:

- Use four spaces to indent code blocks.
- Use underscores for variable and function names, and use camelCase for class names.
- Limit lines to 79 characters.
- Use whitespace to separate logical sections of code.
- Use comments to explain complex sections of code.
- Use docstrings to document functions, modules, and classes.

Overall, PEP 8 is an important resource for Python developers who want to write clean, consistent, and maintainable code. By following these guidelines, developers can create code that is easier to read, understand, and modify.

11. How is memory management done in python?

Ans: Memory management in Python is handled automatically by the Python runtime environment. The process is done through a mechanism called "garbage collection," which is the process of automatically freeing up memory that is no longer being used by the program.

The Python runtime environment uses a technique called "reference counting" to manage memory. Every object in Python has a reference count, which is the number of references or pointers to that object. When an object is created, its reference count is set to 1. When a new reference to the object is created, such as when an object is assigned to a variable or passed as an argument to a function, the reference count is incremented. When a reference is deleted, the reference count is decremented. When the reference count reaches 0, the object is no longer being used by the program and is automatically deleted by the garbage collector.

Python also has a memory manager that handles larger blocks of memory, called "memory arenas." Memory arenas are used to allocate and deallocate memory for Python objects. The memory manager is responsible for requesting memory from the operating system and dividing it into smaller blocks as needed. When an object is deleted, its memory is returned to the memory manager and can be reused for other objects.

In summary, Python's memory management is automated through the use of garbage collection and reference counting, which ensures that memory is allocated and deallocated efficiently and automatically.

12. Define modules in python.

Ans: In Python, a module is a file containing Python code that defines functions, classes, and variables that can be used in other Python programs. A module can also contain executable code that is run when the module is imported.

A module can be created simply by writing Python code in a text file and saving it with a `.py` extension. To use the functions and variables defined in a module, it must be imported into another Python script using the `import` statement.

For example, suppose you have a file `my_module.py` containing the following Python code:

```python
def add_numbers(x, y):
    return x + y
```

```
def subtract_numbers(x, y):
    return x - y
```

To use the `add_numbers` and `subtract_numbers` functions in another Python script, you can import the `my_module` module using the following code:

```python
import my_module

result1 = my_module.add_numbers(3, 5)
result2 = my_module.subtract_numbers(7, 2)

print(result1)  # Output: 8
print(result2)  # Output: 5
```

In this example, the `my_module` module is imported using the `import` statement, and the `add_numbers` and `subtract_numbers` functions are called using the dot notation `my_module.function_name()`.

13. What are the built in types available in python?

Ans: Python has several built-in data types, including:

1. Numeric types:
   - `int` (integer)
   - `float` (floating-point number)
   - `complex` (complex number)

2. Boolean type:
   - `bool` (boolean)

3. Sequence types:
   - `str` (string)
   - `list` (list)
   - `tuple` (tuple)
   - `range` (range)

4. Mapping type:
   - `dict` (dictionary)

5. Set types:
   - `set` (set)
   - `frozenset` (frozen set)

6. Binary types:
   - `bytes` (bytes)

- `bytearray` (byte array)
- `memoryview` (memory view)

Each of these built-in types has its own set of operations and methods that can be used to manipulate and work with data of that type.

14. What are python decorators?

Ans: In Python, a decorator is a special type of function that can modify the behavior of another function. Decorators are used to add functionality to an existing function without modifying its source code. They are a powerful and flexible feature of Python that allows for the creation of reusable code components.

A decorator is defined using the `@decorator_name` syntax, where `decorator_name` is the name of the decorator function. The decorator function takes the function it decorates as an argument, modifies its behavior in some way, and returns the modified function.

For example, suppose you have a function `my_function()` that takes some input, performs some processing, and returns an output. You can define a decorator function `my_decorator()` that modifies the behavior of `my_function()` by adding some additional processing before or after the original function call.

Here is an example of a simple decorator that adds a greeting message before and after the function call:

```python
def my_decorator(func):
    def wrapper():
        print("Hello, this is before the function call.")
        func()
        print("This is after the function call.")
    return wrapper

@my_decorator
def my_function():
    print("This is my function.")

my_function()
```

In this example, the `my_decorator()` function takes a function `func` as an argument and defines a new function `wrapper()` that adds the greeting message before and after the original function call. The `wrapper()` function is returned by the decorator and replaces the original function. The `@my_decorator` syntax is used to apply the decorator to the `my_function()` function.

When you call `my_function()`, the output will be:

```
Hello, this is before the function call.
This is my function.
This is after the function call.
```

In this way, decorators allow you to modify the behavior of a function without changing its source code, which can be useful for adding functionality such as logging, authentication, or caching to your functions.

15. How do we find bugs and statistical problems in python?

Ans:

Finding bugs and statistical problems in Python can be done using various tools and techniques. Here are some common methods:

1. Debugging: Python provides built-in debugging tools such as `pdb` and `ipdb`, which allow you to step through your code line by line, inspect variables, and find errors in your code.

2. Testing: Python has a built-in testing framework called `unittest` that allows you to write automated tests for your code. You can use this framework to ensure that your code is behaving as expected and catching errors before they make it into production.

3. Code analysis: There are several code analysis tools available for Python, such as `pylint`, `flake8`, and `mypy`, that can help you find errors, inconsistencies, and style violations in your code. These tools can also help you identify potential performance issues and memory leaks.

4. Statistical analysis: For statistical problems, Python has many libraries such as `pandas`, `numpy`, and `scipy`, which offer a variety of statistical functions and analysis tools. These libraries can help you identify outliers, trends, correlations, and other patterns in your data. Additionally, there are visualization libraries such as `matplotlib` and `seaborn`, which can help you create graphs and charts to visualize your data.

In summary, Python provides a variety of built-in and third-party tools that can help you find and fix bugs and statistical problems in your code. It's important to use a combination of these tools to ensure that your code is robust, efficient, and accurate.

16. What is the difference between .py and .pyc?

Ans: The `.py` and `.pyc` extensions are both used in Python, but they have different purposes:

1. `.py` files: These are the source code files in Python. They contain the actual Python code that you write, and can be edited and executed directly by the Python interpreter. When you run a Python program using the `python` command, you're telling the interpreter to read and execute the code in a `.py` file.

2. `.pyc` files: These are compiled bytecode files that are created by the Python interpreter when you run a `.py` file. The interpreter compiles the source code into bytecode, which is a lower-level representation of the code that can be executed more efficiently. The bytecode is then saved to a `.pyc` file, which can be executed directly by the interpreter the next time you run the program. This can help speed up the execution of your Python code, especially for larger programs.

In summary, `.py` files contain the source code that you write, while `.pyc` files contain compiled bytecode that is created by the Python interpreter. You don't need to manually create `.pyc` files; the interpreter will automatically generate them for you when you run a `.py` file.

17. Define string in python.

Ans: In Python, a string is a sequence of characters enclosed in quotation marks (either single or double). Strings are one of the built-in data types in Python and are used to represent text and other types of data that can be represented as a sequence of characters.

In Python, strings are immutable, which means that once a string is created, it cannot be changed. However, new strings can be created by concatenating existing strings or using string formatting. String concatenation is done using the `+` operator, and string formatting is done using various string formatting methods, such as the `format()` method or f-strings.

For example, the following code creates a string variable and assigns it the value "Hello, World!":

```python
my_string = "Hello, World!"
```

Strings can be indexed and sliced like other sequences in Python. For example, the first character of a string can be accessed using an index of 0:

```python
first_char = my_string[0]  # 'H'
```

Slicing can be used to extract a substring from a string:

```python
```

```
substring = my_string[0:5]  # 'Hello'
```

There are also many built-in string methods in Python that can be used to perform various operations on strings, such as converting the case of the string, searching for substrings, replacing substrings, and more.

18. What do you understand about the term namespace in python?

Ans: In Python, a namespace is a mapping between names and objects. Namespaces are used to avoid naming collisions and to provide a way to organize code and variables. Every name that is defined in a Python program belongs to a specific namespace.

There are several types of namespaces in Python:

1. Local namespace: This is the namespace of a function or method. It contains the local variables and parameters defined within the function.

2. Global namespace: This is the namespace of a module. It contains the global variables and functions defined within the module.

3. Built-in namespace: This is the namespace of the Python interpreter. It contains the built-in functions and modules that are available to all Python programs.

When you use a name in a Python program, the interpreter looks for that name in the local namespace first. If the name is not found in the local namespace, it looks for it in the global namespace. If the name is not found in either the local or global namespace, it looks for it in the built-in namespace.

You can access the contents of a namespace using the `dir()` function or by accessing the namespace as a dictionary. For example, `globals()` returns a dictionary containing the contents of the global namespace.

Understanding namespaces is important for writing maintainable and organized code in Python. By using different namespaces for different parts of your code, you can avoid naming collisions and make your code easier to understand and modify.

19. How do you create a python function?

Ans: In Python, a function is defined using the `def` keyword followed by the name of the function and a set of parentheses that may or may not contain parameters. The body of the function is indented below the function definition.

Here's a basic example of a Python function that takes two arguments and returns their sum:

```python
def add_numbers(x, y):
    sum = x + y
    return sum
```

In this example, the function is named `add_numbers` and takes two parameters, `x` and `y`. The body of the function calculates the sum of `x` and `y` and assigns it to a variable named `sum`. Finally, the function returns the value of `sum`.

To call a function in Python, you simply write the name of the function followed by a set of parentheses containing the arguments to be passed to the function. For example:

```python
result = add_numbers(3, 4)
print(result)  # Output: 7
```

This code calls the `add_numbers` function with the arguments 3 and 4, and assigns the return value (7) to the variable `result`. The `print` function is then used to display the value of `result`.

Functions can also have optional parameters, default parameter values, variable-length argument lists, and other features. However, the basic syntax for defining and calling a function in Python is as described above.

DELL:

1. Difference between List and Tuple.

Ans:

Mutability:
Lists are mutable, which means you can add, remove, or modify elements in a list after it has been created. Tuples, on the other hand, are immutable, which means that once a tuple is created, its contents cannot be changed.
Syntax:
Lists are created using square brackets [], while tuples are created using parentheses ().
Performance:
Tuples are generally faster than lists because they are immutable, so they can be optimized by the interpreter more efficiently. Lists require more memory allocation and deallocation, which makes them slower.
Usage:

Lists are typically used when you need to store a collection of items that may change over time, such as a list of to-do items. Tuples, on the other hand, are typically used when you need to store a collection of items that won't change, such as the coordinates of a point in space.

2. Explain ternary operator in python.

   Ans: The ternary operator in Python is a shorthand way of writing an if-else statement in a single line of code. It's a convenient way of writing simple conditional expressions without having to write a full if-else statement.

   Example:

```
def is_even(num):
    return True if num % 2 == 0 else False
```

3. What are negative indices?

   Ans: In Python, negative indices refer to the position of an element in a sequence (such as a string or list) starting from the end instead of the beginning.

   For example, in a string "Hello, World!", the first character "H" is at position 0, while the last character "!" is at position -1. Similarly, the second last character is at position -2, the third last at position -3, and so on.

   Using negative indices can be helpful in cases where you want to access elements from the end of a sequence, without having to know its length. It can also simplify certain operations like extracting the last element of a list or string, or reversing the order of a sequence.

4. Is python case sensitive?

   Ans: No

5. How long can an identifier be in python?

   Ans:

6. What is the pass statement in python?

   Ans: The `pass` statement in Python is a null operation. It is a placeholder statement in a block of code, indicating that there is no action to be taken. It is often used as a placeholder for code that is not yet implemented, or for a code block that needs to be empty but still needs to exist syntactically.

   The `pass` statement is a way to make sure that the interpreter does not raise a syntax error due to an empty block of code. It can be used in a variety of contexts, such as in empty function definitions or in conditional statements where no action is required for a particular condition. For example, consider the following code:

```
if x < 0:
    pass  # Placeholder for handling negative numbers
else:
    print("Positive")
```

Here, the `pass` statement is used as a placeholder for handling negative numbers. If `x` is less than zero, the code block under the `if` statement would be empty without the `pass` statement. The `pass` statement makes sure that the interpreter does not raise a syntax error, and the code will continue to run without any action taken for negative numbers.

7. Explain help() and dir() functions in python.

Ans: In Python, the `help()` and `dir()` functions are built-in functions used for introspection and to access the documentation of an object.

The `help()` function displays the documentation string and other useful information about a module, function, class, or method. It can be called in two ways:

1. `help(object)` - displays help information for the specified object.
2. `help()` - starts the help interpreter, where you can enter the name of the object to get help for.

For example, to get help information for the `print()` function, you can use `help(print)` or `help('print')`.

The `dir()` function, on the other hand, returns a list of valid attributes for the specified object. It can be called in two ways:

1. `dir([object])` - returns a list of valid attributes for the specified object. If no object is specified, it returns a list of valid attributes for the current namespace.
2. `dir()` - returns a list of valid attributes for the current namespace.

For example, to get a list of valid attributes for the `math` module, you can use `dir(math)` or `dir('math')`.

Both `help()` and `dir()` functions are useful for exploring and understanding Python modules, functions, classes, and objects.

8. How does a function return values?

Ans: In Python, a function can return a value using the `return` statement. The `return` statement is followed by an expression that is evaluated and returned as the result of the function.

Here is an example of a function that returns a value:

```
def add_numbers(a, b):
    result = a + b
    return result
```

In this example, the function takes two parameters `a` and `b`, calculates their sum and stores it in the `result` variable. Finally, the function returns the value stored in `result` using the `return` statement.

The returned value can be assigned to a variable or used in an expression:

```
x = add_numbers(5, 10)
print(x)  # Output: 15
```

9. What is the python interpreter prompt?

Ans: The Python interpreter prompt is a command line interface that allows users to interactively enter and execute Python commands and statements. It is also known as the Python shell or REPL (Read-Eval-Print Loop). The prompt displays the version of Python being used and waits for user input. Users can enter Python code directly at the prompt, and the interpreter will execute the code and display the result, if any. The prompt also provides various commands and keyboard shortcuts to help users navigate and interact with the interpreter. The prompt typically looks like this:

```
Python 3.9.2 (default, Feb 24 2021, 13:46:16)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` symbol indicates that the interpreter is ready to receive input.

10. How would you define a block in python?

Ans: In Python, a block is a group of statements that are executed together as a unit. A block is defined by its indentation level. All statements indented to the same level are part of the same block. In Python, indentation is used to define the scope of control statements, loops, functions, and classes.

For example, consider the following code snippet:

```
```

```
if x < 0:
    print('x is negative')
else:
    print('x is non-negative')
```

Here, the if-else statements form a block. Both the `print` statements are indented at the same level and belong to the same block. If the condition `x < 0` is true, the first statement will be executed, and if it is false, the second statement will be executed.

11. Why do we need break and continue in python?

Ans: In Python, the `break` and `continue` statements are used to alter the flow of control in loops.

The `break` statement is used to terminate the loop prematurely, based on some condition. When a `break` statement is executed inside a loop, the loop is terminated immediately, and the program execution moves to the statement following the loop.

For example, the following code will iterate over the numbers from 1 to 10, but will terminate the loop prematurely when it encounters the number 5:

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

The `continue` statement, on the other hand, is used to skip the current iteration of a loop based on some condition, and move on to the next iteration. When a `continue` statement is executed inside a loop, the current iteration of the loop is terminated immediately, and the next iteration starts.

For example, the following code will iterate over the numbers from 1 to 10, but will skip the number 5:

```
for i in range(1, 11):
    if i == 5:
        continue
    print(i)
```

Both `break` and `continue` statements can be used with loops such as `for` and `while` loops. They can be very useful in controlling the flow of execution in loops, especially in cases where certain conditions need to be met for the loop to continue or terminate.

12. What are the built-in type does python provides?

Ans: Python provides several built-in data types. Some of the most commonly used built-in data types are:

1. Numeric types: `int`, `float`, `complex`
2. Sequence types: `list`, `tuple`, `range`
3. Text type: `str`
4. Mapping type: `dict`
5. Set types: `set`, `frozenset`
6. Boolean type: `bool`
7. Binary types: `bytes`, `bytearray`, `memoryview`

Each of these data types has its own set of methods and operations that can be performed on them.

13. What is a namespace in python?

Ans: In Python, a namespace is a mapping between names and objects. Namespaces are used to avoid naming collisions and to provide a way to organize code and variables. Every name that is defined in a Python program belongs to a specific namespace.

There are several types of namespaces in Python:

1. Local namespace: This is the namespace of a function or method. It contains the local variables and parameters defined within the function.

2. Global namespace: This is the namespace of a module. It contains the global variables and functions defined within the module.

3. Built-in namespace: This is the namespace of the Python interpreter. It contains the built-in functions and modules that are available to all Python programs.

When you use a name in a Python program, the interpreter looks for that name in the local namespace first. If the name is not found in the local namespace, it looks for it in the global namespace. If the name is not found in either the local or global namespace, it looks for it in the built-in namespace.

You can access the contents of a namespace using the `dir()` function or by accessing the namespace as a dictionary. For example, `globals()` returns a dictionary containing the contents of the global namespace.

Understanding namespaces is important for writing maintainable and organized code in Python. By using different namespaces for different parts of your code, you can avoid naming collisions and make your code easier to understand and modify.

14. What is lambda ?

Ans: In Python, a lambda function is a small anonymous function that can take any number of arguments, but can only have one expression. It is a way to create a function without using the def keyword, and instead uses the lambda keyword.

Example:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# filter the list to only include even numbers
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))

# print the filtered list
print(filtered_list)


O/P:
[2, 4, 6, 8, 10]
```

15. In python what are the iterators.

Ans: In Python, an iterator is an object that can be iterated upon. It is used to implement a sequence of values, and its main advantage is that it saves memory by computing values on the fly, rather than storing them all at once.

An iterator must implement two methods:

1. `__iter__()` method: This method returns the iterator object itself. It is called when the iterator is initialized, and the returned object is used for the iteration.

2. `__next__()` method: This method returns the next value in the sequence. If there are no more items to return, it raises the StopIteration exception.

Iterators are used in many parts of Python, including for loops, generators, and list comprehensions. They provide a powerful and efficient way to work with sequences of data.

CTS:

1. How to remove values from a python array?

Ans:

2. Why do we use the split method in python?

Ans: The `split()` method in Python is used to split a string into a list of substrings based on a delimiter. The delimiter can be specified as an argument to the `split()`

method. By default, if no delimiter is specified, the method will split the string on whitespace.

The `split()` method is useful for parsing and manipulating textual data. For example, if you have a string that contains comma-separated values, you can use the `split()` method to split the string into a list of values.

Here's an example:

```python
my_string = "apple,banana,orange"
my_list = my_string.split(",")
print(my_list)
```

Output:
```
['apple', 'banana', 'orange']
```

In this example, the `split()` method is used to split the `my_string` variable into a list of substrings based on the comma delimiter. The resulting list is then printed to the console.

3. How can we access a module written in python from c?

Ans:

4. How do you copy an object in python?

Ans: In Python, there are two ways to copy an object: shallow copy and deep copy.

1. Shallow Copy: In shallow copy, a new object is created which is a reference to the original object. Any changes made to the original object will also be reflected in the copied object.

There are two ways to create a shallow copy of an object:

a. Using the slicing operator [:]

Example:

```
original_list = [1, 2, 3, 4]
copied_list = original_list[:]
```

b. Using the copy() method

Example:

```
import copy
original_list = [1, 2, 3, 4]
copied_list = original_list.copy()
```

2. Deep Copy: In deep copy, a new object is created which is a completely independent copy of the original object. Any changes made to the original object will not be reflected in the copied object.

To create a deep copy of an object, we can use the deepcopy() function from the copy module.

Example:

```
import copy
original_list = [[1, 2], [3, 4]]
copied_list = copy.deepcopy(original_list)
```

In this example, a deep copy of the original list is created, which means that any changes made to the original list will not be reflected in the copied list.

5. How do we reverse a list in python?

Ans:
```
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list)

O/P: [5,4,3,2,1]

Or,

x = my_list[::-1]
print(x)

O/P: [5,4,3,2,1]
```

6. How can we debug a python program?

Ans: There are several ways to debug a Python program:

1. Print statements: You can insert print statements in your code to print the value of variables at different points in the program and debug it step by step. This is a simple and quick way to debug small programs.

2. Python Debugger (PDB): PDB is a built-in Python debugger that allows you to step through your code, set breakpoints, and examine the values of variables. You can start the debugger by adding the following line in your code:

```python
import pdb; pdb.set_trace()
```

3. Integrated Development Environments (IDEs): There are several IDEs available for Python that have built-in debugging tools, such as PyCharm, VSCode, and Spyder.

4. Logging: You can use Python's logging module to log messages at different levels (debug, info, warning, error, etc.) and examine them to find out where your program is going wrong.

5. Profiling: You can use Python's built-in profiling tools to identify performance bottlenecks in your code and optimize it for speed.

Overall, the choice of debugging method depends on the complexity of your program and your personal preferences.

7. Why do we use join() function?

Ans: We use the `join()` function in Python to concatenate a list of strings into a single string, using a specified delimiter. It takes an iterable object such as a list, tuple, or string and concatenates the elements into a single string, separated by the specified delimiter. This is a very useful function for formatting text or combining multiple pieces of data into a single string.

For example, consider a list of words:

```
words = ["hello", "world", "how", "are", "you"]
```

To join these words into a single string, separated by a space, we can use the `join()` function like this:

```
" ".join(words)
```

This will return a single string:

```
'hello world how are you'
```

We can use any character or string as a delimiter in the `join()` function. For example, to join the words using a hyphen as the delimiter, we can do:

```
"-".join(words)
```

This will return:

```
'hello-world-how-are-you'
```

8. What is the use of a break statement?

Ans: The break statement in Python is used to terminate the loop statement in which it is present. When the break statement is encountered inside a loop, the loop is terminated and the program execution continues with the statement immediately following the loop.

Example:

```
i = 1
while i <= 10:
    if i > 5:
        break
    print(i)
    i += 1

O/P:
1
2
3
4
5
```

9. What is tuple in python?

Ans: In Python, a tuple is a collection of ordered, immutable, and heterogeneous elements enclosed in parentheses (). Tuples are similar to lists, but unlike lists, tuples cannot be modified once they are created.

Tuples can store any type of data, including numbers, strings, and other objects. Tuples can be indexed, sliced, and concatenated, just like lists. However, since tuples are immutable, operations that change the content of a tuple (such as appending or removing elements) are not allowed.

Tuples are commonly used to represent a group of related values that should not be modified, such as the coordinates of a point in space (x, y, z), or the RGB color of a pixel (r, g, b). Tuples are also used as return values of functions, where multiple values need to be returned together.

10. What is an operator in python?

Ans: In Python, an operator is a special symbol or character that performs an operation on one or more operands (variables, values, or expressions) and produces a result. Python provides a wide range of operators, including arithmetic operators, comparison operators, logical operators, identity operators, membership operators, and bitwise operators.

11. Is python interpreted language?

Ans: Yes

12. Which programming language is a good choice between Java and Python?

Ans:

Wipro:

1. How can we make forms in python?

Ans:

2. What are the benefits of python?

Ans:

1. Easy to Learn:

Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

2. Large Community:

Python has a large and active community of developers and users who contribute to a wide range of open-source projects, libraries, and frameworks. This makes it easy to find support, get help with problems, and learn from other people's code.

3. Versatile:

Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

4. Interpreted:

Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This makes it easy to write and test code quickly, without needing to worry about compilation.

5. Platform-independent:

Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

6. High-level:

Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

3. What are the key features of python?

Ans:

Easy to Learn:

Python has a simple and easy-to-learn syntax, which makes it a great choice for beginners. The language's focus on readability and simplicity can also make it faster to write and modify code.

Versatile:

Python can be used for a wide range of applications, from web development to data science to machine learning. It has a vast collection of libraries and frameworks that make it easy to build complex applications quickly.

Interpreted:

Python is an interpreted language, which means that programs are executed directly by the interpreter, rather than being compiled into machine code beforehand. This

makes it easy to write and test code quickly, without needing to worry about compilation.

Platform-independent:

Python code can run on a wide range of platforms, including Windows, macOS, and Linux, as well as many other operating systems. This makes it easy to build applications that can run on different systems without needing to modify the code.

High-level:

Python is a high-level language, which means that it provides a lot of built-in functionality and abstraction, making it faster to write code for complex tasks.

4. What type of language is python?Programming or scripting?

Ans: Python is a high-level programming language that can be used for various purposes such as web development, scientific computing, data analysis, artificial intelligence, and more.

Python is often referred to as a scripting language, because it is interpreted, and can be run without the need for compilation. However, Python is also a full-fledged programming language, as it provides support for various programming paradigms such as procedural, object-oriented, and functional programming.

So, to summarize, Python is a high-level programming language that can be used as both a scripting language and a general-purpose programming language.

5. What are the applications of python?

Ans: Python is a versatile programming language that can be used for a wide range of applications, including:

1. Web Development: Python has many popular web frameworks, such as Django and Flask, that make it easy to build complex web applications.

2. Data Science and Machine Learning: Python has become a popular language for data analysis and machine learning, with libraries such as NumPy, Pandas, and TensorFlow.

3. Scripting and Automation: Python is well-suited for writing scripts and automating tasks, such as system administration, file management, and web scraping.

4. Desktop GUI Applications: Python can be used to build graphical user interfaces (GUIs) for desktop applications using libraries such as PyQt and wxPython.

5. Game Development: Python has several game development libraries, such as Pygame and Panda3D, which make it easy to create games.

6. Scientific Computing: Python has a rich collection of scientific computing libraries, such as SciPy, that make it useful for scientific applications.

7. Education: Python's simple and easy-to-read syntax makes it a great language for teaching programming to beginners.

Overall, Python's versatility and large collection of libraries and frameworks make it useful for a wide range of applications, from small scripts to large-scale software projects.

6. What is the difference between list and tuple?

Ans:

Mutability:
Lists are mutable, which means you can add, remove, or modify elements in a list after it has been created. Tuples, on the other hand, are immutable, which means that once a tuple is created, its contents cannot be changed.
Syntax:
Lists are created using square brackets [], while tuples are created using parentheses ().
Performance:
Tuples are generally faster than lists because they are immutable, so they can be optimized by the interpreter more efficiently. Lists require more memory allocation and deallocation, which makes them slower.
Usage:
Lists are typically used when you need to store a collection of items that may change over time, such as a list of to-do items. Tuples, on the other hand, are typically used when you need to store a collection of items that won't change, such as the coordinates of a point in space.

7. What are the global and local variables?

Ans: In Python, a variable's scope determines where it can be accessed from within a program. There are two main types of variable scopes in Python:

Local Variables:
A local variable is defined within a function, and its scope is limited to that function. The variable can be accessed only within that function and not from outside of it. Once the function execution is completed, the memory allocated to the local variable is released.

Example:

```
def my_func():
    x = 10   # local variable
```

```
    print(x)

my_func()   # prints 10
```

Global Variables:
A global variable is defined outside of any function, and its scope is not limited to a particular function but extends throughout the entire program. Any function can access and modify the global variable. To modify a global variable within a function, you must use the global keyword to indicate that the variable is global and not local. If a variable is referenced in a function, and it is not defined within the function, Python will look for it in the global scope.

Example:

```
x = 10  # global variable

def my_func():
    global x
    x = 20  # modify the global variable
    print(x)

my_func()   # prints 20
print(x)    # prints 20
```

8. Define Python Path.

Ans:

9. What are the two major loop in python?

Ans: The two major loops in Python are the `for` loop and the `while` loop.

`for` loop:

A `for` loop is used to iterate over a sequence (such as a list, tuple, or string) or other iterable objects (such as a dictionary, set, or generator) and perform some operations on each item in the sequence.

The syntax of the `for` loop is as follows:

```python
for variable in sequence:
    # do something with variable
```

`while` loop:

A `while` loop is used to repeatedly execute a block of code as long as a particular condition is true.

The syntax of the `while` loop is as follows:

```python
while condition:
    # do something
```

In a `while` loop, the condition is evaluated before each iteration. If the condition is true, the loop body is executed. If the condition is false, the loop terminates.

Both `for` and `while` loops are powerful constructs that are commonly used in Python programming to control the flow of execution and automate repetitive tasks.

10. How is memory management done in python?

Ans: Memory management in Python is handled automatically by the Python runtime environment. The process is done through a mechanism called "garbage collection," which is the process of automatically freeing up memory that is no longer being used by the program.

The Python runtime environment uses a technique called "reference counting" to manage memory. Every object in Python has a reference count, which is the number of references or pointers to that object. When an object is created, its reference count is set to 1. When a new reference to the object is created, such as when an object is assigned to a variable or passed as an argument to a function, the reference count is incremented. When a reference is deleted, the reference count is decremented. When the reference count reaches 0, the object is no longer being used by the program and is automatically deleted by the garbage collector.

Python also has a memory manager that handles larger blocks of memory, called "memory arenas." Memory arenas are used to allocate and deallocate memory for Python objects. The memory manager is responsible for requesting memory from the operating system and dividing it into smaller blocks as needed. When an object is deleted, its memory is returned to the memory manager and can be reused for other objects.

In summary, Python's memory management is automated through the use of garbage collection and reference counting, which ensures that memory is allocated and deallocated efficiently and automatically.

11. What are the built in types available in python?

Ans: Python has several built-in data types, including:

1. Numeric types:
   - `int` (integer)
   - `float` (floating-point number)
   - `complex` (complex number)

2. Boolean type:
   - `bool` (boolean)

3. Sequence types:
   - `str` (string)
   - `list` (list)
   - `tuple` (tuple)
   - `range` (range)

4. Mapping type:
   - `dict` (dictionary)

5. Set types:
   - `set` (set)
   - `frozenset` (frozen set)

6. Binary types:
   - `bytes` (bytes)
   - `bytearray` (byte array)
   - `memoryview` (memory view)

Each of these built-in types has its own set of operations and methods that can be used to manipulate and work with data of that type.

12. Define string.

Ans: In Python, a string is a sequence of characters enclosed in quotation marks (either single or double). Strings are one of the built-in data types in Python and are used to represent text and other types of data that can be represented as a sequence of characters.

In Python, strings are immutable, which means that once a string is created, it cannot be changed. However, new strings can be created by concatenating existing strings or using string formatting. String concatenation is done using the `+` operator, and string formatting is done using various string formatting methods, such as the `format()` method or f-strings.

For example, the following code creates a string variable and assigns it the value "Hello, World!":

```python
my_string = "Hello, World!"
```

```

Strings can be indexed and sliced like other sequences in Python. For example, the first character of a string can be accessed using an index of 0:

```python
first_char = my_string[0]  # 'H'
```

Slicing can be used to extract a substring from a string:

```python
substring = my_string[0:5]  # 'Hello'
```

There are also many built-in string methods in Python that can be used to perform various operations on strings, such as converting the case of the string, searching for substrings, replacing substrings, and more.

13. How do you create a python function?

Ans: In Python, a function is defined using the `def` keyword followed by the name of the function and a set of parentheses that may or may not contain parameters. The body of the function is indented below the function definition.

Here's a basic example of a Python function that takes two arguments and returns their sum:

```python
def add_numbers(x, y):
    sum = x + y
    return sum
```

In this example, the function is named `add_numbers` and takes two parameters, `x` and `y`. The body of the function calculates the sum of `x` and `y` and assigns it to a variable named `sum`. Finally, the function returns the value of `sum`.

To call a function in Python, you simply write the name of the function followed by a set of parentheses containing the arguments to be passed to the function. For example:

```python
result = add_numbers(3, 4)
print(result)  # Output: 7
```
```

This code calls the `add_numbers` function with the arguments 3 and 4, and assigns the return value (7) to the variable `result`. The `print` function is then used to display the value of `result`.

Functions can also have optional parameters, default parameter values, variable-length argument lists, and other features. However, the basic syntax for defining and calling a function in Python is as described above.

14. How does a function return values?

Ans: In Python, a function can return a value using the `return` keyword followed by the value to be returned. When a function is called, it executes the code within its body and returns a value to the caller using the `return` statement.

Here is a simple example that demonstrates how to return a value from a function:

```python
def multiply_numbers(x, y):
    product = x * y
    return product

result = multiply_numbers(5, 7)
print(result)   # Output: 35
```

In this example, the function `multiply_numbers` takes two arguments, multiplies them together, and returns the result using the `return` statement. The return value is then assigned to the variable `result` and printed to the console.

If a function does not have a return statement or if it has a return statement without any value, it returns `None` by default. For example:

```python
def print_hello():
    print("Hello, World!")

result = print_hello()
print(result)   # Output: None
```

In this example, the function `print_hello` does not have a return statement. When it is called, it prints "Hello, World!" to the console, but it does not return a value. As a result, the variable `result` is assigned the value `None` by default.

HCL:

1. How is multithreading achieved in python?

Ans: Multithreading is the ability of a program to perform multiple tasks concurrently. In Python, multithreading can be achieved using the threading module. The threading module provides a way to create and manage threads in a Python program.

Example:

```
import threading

# Define a function to be executed in a thread
def print_numbers():
    for i in range(1, 11):
        print(i)

# Create a thread for the function
thread = threading.Thread(target=print_numbers)

# Start the thread
thread.start()

# Do other things in the main thread
for i in range(11, 21):
    print(i)

# Wait for the thread to finish
thread.join()

print("Done")
```

In this example, a function print_numbers is defined that prints the numbers from 1 to 10. A thread is created for this function using the Thread class from the threading module. The start method is called on the thread object to start the thread, and the join method is called to wait for the thread to finish before continuing. While the thread is running, the main thread continues to execute other code. In this example, the main thread prints the numbers from 11 to 20. When the thread finishes, the program prints "Done".

Note that in Python, multithreading is limited by the Global Interpreter Lock (GIL), which means that only one thread can execute Python bytecode at a time. This can limit the effectiveness of multithreading in certain situations, such as CPU-bound tasks that require a lot of computation. However, multithreading can still be useful for tasks that are I/O-bound, such as network or file I/O, where threads can be used to perform non-blocking I/O operations. In cases where CPU-bound tasks are a concern, multiprocessing or asynchronous programming may be better options.

2. How can you build a simple logistic regression model in python?

Ans:

3. Which library would you prefer for plotting in python language: Seaborn or Matplotlib.

Ans:

4. What is the main difference between a pandas series and a single column DataFrame in python?

Ans: In Python's Pandas library, a Pandas Series is a one-dimensional labeled array that can hold data of any type, whereas a DataFrame is a two-dimensional labeled data structure that can hold data of multiple types in columns. A single column DataFrame is essentially a DataFrame with only one column.

The main difference between a Pandas Series and a single column DataFrame is that a Series does not have column names, whereas a single column DataFrame has a column name. In other words, a Series is a data structure that contains a sequence of values with an associated index, but no column name. A single column DataFrame is a data structure that contains a sequence of values with an associated index and a column name.

Here's an example to illustrate the difference:

```python
import pandas as pd

# Creating a Pandas Series
s = pd.Series([1, 2, 3, 4, 5])

# Creating a single column DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})

# Displaying the Series and DataFrame
print(s)
print(df)
```

Output:

```
0    1
1    2
2    3
3    4
4    5
dtype: int64

   A
0  1
1  2
```

```
2  3
3  4
4  5
```

As you can see, the Series does not have a column name, whereas the single column DataFrame has a column name 'A'.

In summary, a Pandas Series is a one-dimensional labeled array that does not have a column name, whereas a single column DataFrame is a two-dimensional labeled data structure that has a column name and only one column.

5. How can you handle duplicate values in a dataset for a variable in python?

Ans: In Python, Pandas is a commonly used library for handling data. To handle duplicate values in a dataset for a variable using Pandas, you can use the `duplicated()` and `drop_duplicates()` methods.

The `duplicated()` method returns a boolean Series indicating which rows are duplicated. By default, it considers all columns to find duplicates, but you can specify a specific column or subset of columns using the `subset` parameter.

```python
import pandas as pd

# Creating a sample dataframe
df = pd.DataFrame({'A': [1, 2, 3, 1, 2], 'B': ['x', 'y', 'z', 'x', 'y']})

# Find duplicated rows
duplicates = df.duplicated()
print(duplicates)
```

Output:

```
0    False
1    False
2    False
3     True
4     True
dtype: bool
```

The `drop_duplicates()` method removes duplicated rows from the dataframe. By default, it keeps the first occurrence of each duplicated row, but you can specify to keep the last occurrence or drop all duplicates using the `keep` parameter.

```python
import pandas as pd

# Creating a sample dataframe
df = pd.DataFrame({'A': [1, 2, 3, 1, 2], 'B': ['x', 'y', 'z', 'x', 'y']})

# Remove duplicated rows
df.drop_duplicates(inplace=True)
print(df)
```

Output:

```
   A  B
0  1  x
1  2  y
2  3  z
```

As you can see, the duplicated rows have been removed, and only the unique rows remain.

You can also use the `drop_duplicates()` method on a specific column or subset of columns by specifying the `subset` parameter.

```python
import pandas as pd

# Creating a sample dataframe
df = pd.DataFrame({'A': [1, 2, 3, 1, 2], 'B': ['x', 'y', 'z', 'x', 'y']})

# Remove duplicated rows based on column 'A'
df.drop_duplicates(subset='A', inplace=True)
print(df)
```

Output:

```
   A  B
0  1  x
1  2  y
2  3  z
```

In this example, only the rows that have unique values in column 'A' are kept, and the duplicated rows with the same value in column 'A' are dropped.

6. Can we create a dataframe with multiple data types in python? If yes, how can you do?

   Ans: Yes, you can create a Pandas DataFrame with multiple data types in Python. Pandas supports a wide range of data types for the columns of a DataFrame, including numeric types (integers, floats), boolean, datetime, and object (strings, mixed types). To create a DataFrame with multiple data types, you can pass a dictionary to the Pandas DataFrame() constructor, where the keys of the dictionary represent the column names and the values represent the data for each column. Each column can have a different data type.

7. Why should you use numpy arrays instead of nested python lists?

   Ans: NumPy arrays are preferred over nested Python lists for several reasons:

   1. **Efficient memory usage**:

   NumPy arrays are stored in contiguous blocks of memory, which makes them more efficient to work with than nested lists. In contrast, Python lists are scattered in memory, making it slower to access elements and taking up more memory.

   2. **Faster computations**:

   NumPy is optimized for numerical computations and provides functions that operate on entire arrays, which is faster than operating on individual elements of a nested list. Additionally, NumPy uses compiled code for many operations, which can significantly speed up computations.

   3. **Broadcasting**:

   NumPy arrays allow for broadcasting, which means that arithmetic operations can be performed on arrays of different shapes and sizes. In contrast, Python lists require that elements be of the same shape and size to perform arithmetic operations.

   4. **Support for advanced operations**:

   NumPy provides many advanced operations, such as slicing, indexing, and masking, which are not available with Python lists. These operations can make complex computations much simpler and more efficient.

   5. **Interoperability with other libraries**:

   NumPy is widely used in scientific computing and data analysis, and it can easily integrate with other libraries like Pandas, Scikit-Learn, and Matplotlib. Using NumPy arrays can make it easier to work with these libraries and share code with others.

In summary, NumPy arrays offer several advantages over nested Python lists in terms of memory usage, speed, flexibility, and advanced operations. If you are working with numerical data, it is recommended to use NumPy arrays instead of nested Python lists.

8. Which scientific libraries in Scipy have you worked on in your project?

   Ans:
9. Which python library is used for machine learning?

   Ans: The most popular Python library used for machine learning is scikit-learn, also known as sklearn. Scikit-learn is a free open-source machine learning library that is built on top of NumPy, SciPy, and Matplotlib. It provides simple and efficient tools for data mining and data analysis, implementing a wide variety of machine learning algorithms, including classification, regression, clustering, and dimensionality reduction.

   Scikit-learn offers a uniform interface for working with various machine learning algorithms, making it easy to switch between different models or to compare their performance. It also includes functions for data preprocessing, feature selection, and model evaluation, making it a complete toolkit for machine learning tasks.

   Other popular machine learning libraries in Python include TensorFlow, PyTorch, Keras, and MXNet. These libraries are often used for deep learning, a subset of machine learning that involves neural networks with many layers. They provide more flexibility and customization options than scikit-learn, but require more expertise and computing power to use effectively.

10. How can you train and interpret a linear regression model in Scikit learn?

11. Define slicing in python.

    Ans: Slicing in Python refers to a technique used to extract a subset of elements from a sequence, such as a list, tuple, or string. Slicing allows you to access a contiguous section of the sequence by specifying the starting and ending indices of the desired elements.

    The general syntax for slicing a sequence is as follows:

    ```
    sequence[start:end:step]
    ```

    Here, `start` is the index of the first element to include in the slice (inclusive), `end` is the index of the last element to include in the slice (exclusive), and `step` is the distance between each element in the slice (default is 1).

For example, consider the following list:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

To slice the list and extract the elements from index 2 (inclusive) to index 5 (exclusive), you would use the following code:

```
my_slice = my_list[2:5]
print(my_slice)
```

This would output the following:

```
[3, 4, 5]
```

You can also use negative indices to slice from the end of the sequence. For example, to slice the last three elements of the list, you could use the following code:

```
my_slice = my_list[-3:]
print(my_slice)
```

This would output the following:

```
[7, 8, 9]
```

Slicing is a powerful technique in Python that allows you to easily manipulate sequences and extract the data you need.

12. How can python be an interpreted language?

Ans: Python is an interpreted language. This means that Python code is not compiled into machine code before it is run. Instead, the Python interpreter reads and executes the code directly from the source code files. When you run a Python program, the interpreter parses the code and executes it one line at a time. This makes it easier to write and debug Python code, as you can see the results of your code immediately. However, interpreted languages are generally slower than compiled languages, as the interpreter has to translate the code into machine code at runtime.

13. Define packages in python?

Ans: In Python, a package is a way of organizing related modules into a single namespace. A package is simply a directory of Python modules that contains a special file called `__init__.py`. This file can be empty or can contain initialization code that is executed when the package is imported.

Packages provide a way to structure larger Python projects by organizing modules into logical units. By grouping related modules into a package, you can avoid naming conflicts and make it easier to reuse code across different parts of your project.

To use a package in Python, you can import it using the `import` statement. For example, to import the `numpy` package, you would use the following code:

```
import numpy
```

Once a package is imported, you can access its modules using the dot notation. For example, to use the `array` function from the `numpy` package, you would use the following code:

```
my_array = numpy.array([1, 2, 3, 4])
```

You can also use the `from` keyword to import specific modules or functions from a package. For example, to import just the `array` function from the `numpy` package, you could use the following code:

```
from numpy import array
my_array = array([1, 2, 3, 4])
```

Packages are an essential part of the Python ecosystem and are used extensively in many popular libraries and frameworks. By organizing your code into packages, you can make it more modular, reusable, and maintainable.

14. Which command is used to delete files in python?

Ans: In Python, you can use the `os.remove()` function to delete a file. The `os.remove()` function deletes the file at the specified path, which can be either a relative or an absolute path.

Here's an example of how to use the `os.remove()` function to delete a file:

```python
import os

# specify the path of the file to be deleted
file_path = "path/to/file.txt"

# delete the file
os.remove(file_path)
```

In the example above, the `os.remove()` function is used to delete the file located at the path `path/to/file.txt`. Once the function is executed, the file will be permanently deleted from the file system.

It's important to note that deleting a file using `os.remove()` is a permanent action and cannot be undone. Be sure to double-check that you're deleting the correct file before running this command.

15. Define pickling and unpickling in python.

Ans: In Python, pickling and unpickling are techniques used to serialize and deserialize Python objects, respectively. Serialization refers to the process of converting an object into a format that can be stored or transmitted, while deserialization refers to the process of converting a serialized object back into its original form.

Pickling is the process of converting a Python object into a byte stream that can be written to a file or transmitted over a network. This is done using the `pickle` module in Python. The `pickle.dump()` function is used to serialize an object and write it to a file, while the `pickle.dumps()` function can be used to return the serialized object as a byte string.

Here's an example of how to use the `pickle.dump()` function to pickle an object:

```python
import pickle

# define an object to be pickled
my_object = {"name": "Alice", "age": 30, "city": "New York"}

# open a file for writing
with open("my_object.pickle", "wb") as file:
    # pickle the object and write it to the file
    pickle.dump(my_object, file)
```

Unpickling is the reverse process of pickling, and it involves converting a byte stream back into a Python object. This is done using the `pickle.load()` function to read the serialized object from a file, or the `pickle.loads()` function to deserialize a byte string.

Here's an example of how to use the `pickle.load()` function to unpickle an object:

```python
import pickle

# open the file containing the pickled object
with open("my_object.pickle", "rb") as file:
    # unpickle the object
    my_object = pickle.load(file)

# print the unpickled object
print(my_object)
```

In the example above, the `pickle.load()` function is used to read the pickled object from the file "my_object.pickle" and convert it back into a Python object. The unpickled object is then printed to the console.

Pickling and unpickling are useful techniques in Python for saving and loading complex data structures, such as dictionaries, lists, and custom objects. However, it's important to note that pickled objects can be potentially unsafe, since they can execute arbitrary code during the unpickling process. As a result, you should only unpickle data from trusted sources.

DXC Technology:

1. What type of a language is python? Interpreted or compiled?

   Ans: Interpreted.

2. What do you mean by python being an interpreted language?

   Ans: Python is an interpreted language. This means that Python code is not compiled into machine code before it is run. Instead, the Python interpreter reads and executes the code directly from the source code files. When you run a Python program, the interpreter parses the code and executes it one line at a time. This makes it easier to write and debug Python code, as you can see the results of your code immediately. However, interpreted languages are generally slower than compiled languages, as the interpreter has to translate the code into machine code at runtime.

3. Is python statistically typed or dynamically?

Ans: Dynamically Typed

4. Is python strongly typed or weakly typed?

Ans: Python is a strongly typed language, which means that the data type of a variable is determined at the time of assignment and cannot be changed later. Once a variable is assigned a value of a certain type, it can only be used with other values of the same type.
For example, if you assign an integer value to a variable, you can only use that variable with other integer values. If you try to use it with a string or a float, Python will raise a TypeError at runtime.

5. What is python syntax for switch case statement?

Ans: Python doesn't have switch case statement.

6. What is a Lambda statement? Give an example.

Ans: In Python, a lambda function is a small anonymous function that can take any number of arguments, but can only have one expression. It is a way to create a function without using the def keyword, and instead uses the lambda keyword.

Example:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# filter the list to only include even numbers
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))

# print the filtered list
print(filtered_list)
```

O/P:
[2, 4, 6, 8, 10]

7. What are the rules for local and global variables in python?

Ans: In Python, a variable's scope determines where it can be accessed from within a program. There are two main types of variable scopes in Python:

Local Variables:
A local variable is defined within a function, and its scope is limited to that function. The variable can be accessed only within that function and not from outside of it. Once the function execution is completed, the memory allocated to the local variable is released.

Example:

```
def my_func():
    x = 10   # local variable
    print(x)

my_func()   # prints 10
```

Global Variables:
A global variable is defined outside of any function, and its scope is not limited to a particular function but extends throughout the entire program. Any function can access and modify the global variable. To modify a global variable within a function, you must use the global keyword to indicate that the variable is global and not local. If a variable is referenced in a function, and it is not defined within the function, Python will look for it in the global scope.

Example:

```
x = 10  # global variable

def my_func():
    global x
    x = 20  # modify the global variable
    print(x)

my_func()   # prints 20
print(x)    # prints 20
```

8. What are generators in python?

Ans: In Python, generators are functions that use the yield keyword to return values one at a time, instead of returning all the values at once. Generators are a way to create iterators, which are objects that can be iterated over (e.g., using a for loop) to produce a sequence of values.

9. What can you use python generator functions for?

Ans: Python generator functions are a powerful feature in the language that allow you to create iterators that generate values on-the-fly rather than storing them in memory. Here are some common use cases for generator functions:

1. Lazily Generating Data: Generator functions are useful when you need to generate a large amount of data that can't fit into memory all at once. Instead of creating a list or other data structure to store the data, you can use a generator to lazily compute and yield the data one item at a time as needed.

2. Infinite Sequences: You can use generator functions to create infinite sequences such as the Fibonacci sequence, prime numbers, or any other sequence that can be computed iteratively.

3. Parsing Data: When working with large data sets, you may need to parse data from files or other sources. You can use a generator function to read and parse the data one record at a time, which can be more memory-efficient than loading the entire data set into memory at once.

4. Filtering Data: You can use generator functions to filter data based on some criteria. For example, you can use a generator to yield only the even numbers from a sequence or to filter out certain types of data from a larger data set.

5. Asynchronous Programming: Generator functions can be used in combination with the `asyncio` library to write asynchronous code that can run concurrently.

Overall, generator functions are a powerful tool in Python that can help you write more efficient and scalable code.

10. When is not a good time to use a python generator?

Ans: While Python generators are a powerful tool that can help you write more efficient and scalable code, there are some scenarios where using a generator may not be the best choice. Here are a few cases where generators may not be a good fit:

1. When you need random access to elements: Generators generate elements one at a time and do not keep all elements in memory, so you cannot randomly access elements like you can with a list or other sequence. If you need to access elements randomly or out of order, a generator may not be the best choice.

2. When you need to modify the sequence: Generators are immutable, meaning you cannot modify the sequence of elements once it has been generated. If you need to modify or update the sequence of elements, you may need to use a different data structure.

3. When the data set is small enough to fit in memory: Generators are most useful when working with large data sets that cannot fit into memory all at once. If your data set is small enough to fit into memory, you may be better off using a list or other sequence.

4. When performance is not a concern: While generators can be more memory-efficient than lists, they may not always be faster. In some cases, using a list or other sequence may be faster and easier to write and read.

5. When the code is simpler without a generator: Sometimes, using a generator can add complexity to the code and make it harder to read and understand. If the code is

simpler and easier to read without a generator, you may be better off using a list or other sequence.

Overall, generators are a powerful tool that can be very useful in many scenarios, but they may not always be the best choice depending on the specific requirements of your code.

11. What is the difference between range and xrange function?

Ans:

range() returns a list of numbers from the starting value to the ending value, incrementing by the step size. The list is generated in memory and stored in its entirety, which means that it can consume a lot of memory if the range is large.

Example:

```
range(0, 10, 2)  # returns [0, 2, 4, 6, 8]
```

xrange() generates the numbers in the sequence on-the-fly as they are needed, rather than generating them all at once and storing them in memory. This makes it much more memory efficient than range(), especially for large ranges.

Example:

```
xrange(0, 10, 2)  # generates 0, 2, 4, 6, 8 on-the-fly
```

In Python 2, there are two built-in functions for generating a sequence of numbers: range() and xrange(). In Python 3, xrange() is no longer available, and the range() function behaves like the xrange() function used to in Python 2.

12. What is PEP8?

Ans: PEP 8 (Python Enhancement Proposal 8) is a style guide for Python code. It was created to promote consistent, readable, and maintainable code across different projects and developers. The style guide covers topics such as naming conventions, whitespace, comments, and formatting.

The goal of PEP 8 is to make Python code more readable and understandable, which can help make it easier to maintain, debug, and modify in the future. By following PEP 8 guidelines, developers can create code that is easier for themselves and others to understand, which can lead to fewer errors and faster development times.

Some of the key guidelines outlined in PEP 8 include:

- Use four spaces to indent code blocks.
- Use underscores for variable and function names, and use camelCase for class names.
- Limit lines to 79 characters.
- Use whitespace to separate logical sections of code.
- Use comments to explain complex sections of code.

- Use docstrings to document functions, modules, and classes.

Overall, PEP 8 is an important resource for Python developers who want to write clean, consistent, and maintainable code. By following these guidelines, developers can create code that is easier to read, understand, and modify.

13. Explain how to copy an object in python?

Ans:

14. How to create a multi dimensional list?

Ans:

15. Disadvantages of python?

Ans:

1. Slower execution speed: Python is an interpreted language, which means that the code is executed line by line, and not compiled into machine code like C or C++. This makes Python slower than compiled languages in terms of execution speed. However, there are tools like PyPy and Cython that can improve Python's execution speed.

2. Weak in mobile computing: Python is not as strong in mobile computing as it is in other areas. While there are Python frameworks like Kivy and BeeWare that allow for mobile development, they are not as popular or well-established as native mobile development tools like Swift and Java.

3. Design restrictions: Python's dynamic nature can sometimes lead to design restrictions. For example, it can be difficult to enforce strong typing and encapsulation in Python code.

4. Global Interpreter Lock (GIL): The GIL is a mechanism in Python that prevents multiple threads from executing Python code in parallel on multiple CPU cores. This can limit Python's ability to take advantage of multiple CPUs and can cause performance issues in certain situations.

5. Limited memory efficiency: Python's dynamic nature can also lead to memory inefficiencies, as the interpreter must allocate memory dynamically for objects and perform garbage collection to free memory that is no longer needed.