1. Difference Between List and Tuple

List	Tuple
1. Lists are **mutable** 2. List is a container to contain different types of objects and is used to iterate objects. 3. Syntax Of List: **list = ['a', 'b', 'c', 1,2,3]** 4. List iteration is slower 5. Lists consume more memory 6. Operations like insertion and deletion are better performed.	1. Tuples are **immutable** 2. Tuple is also like list but contains immutable objects. 3. Syntax Of Tuple: **tuples = ('a', 'b', 'c', 1, 2)** 4. Tuple processing is faster than List. 5. Tuple consume less memory 6. Elements can be accessed better.

2. What is Decorator? Explain With Example.

In the simplest terms, a **decorator** in Python is a way to add extra features or functionality to an existing function without changing its actual code.
Imagine a decorator like a gift wrapper. The gift (function) stays the same, but you are adding something around it (like wrapping paper) to make it look or behave a little differently.

How it works:

1. **You have a function**: This is the main "gift" you want to use.
2. **You have a decorator**: This is a "wrapper" that adds something extra (like logging, timing, or modifying the output) when the function is called.
3. **You combine them**: You place the @decorator_name on top of your function, and now whenever you call that function, it will have that extra feature automatically.

Input	**Output**

def my_decorator(func):
    def wrapper():
        print("Before the function runs")
        func()
        print("After the function runs")
    return wrapper
 	
@my_decorator
def say_hello():
    print("Hello!")
 	
say_hello()
	Before the function runs
	Hello!
	After the function runs

3. Difference Between List and Dict Comprehension

List Comprehension	**Dict Comprehension**

Syntax:
    [expression for item in iterable if conditional]
Example:
Common Way:
l = []
for i in range(10):
    if i%2:
        l.append(i)
print(l)
Using List Comprehension:
ls = [i for i in range(10) if i%2]
print(ls)
Output:
[1, 3, 5, 7, 9]
	Syntax:
	    {key:value for (key,value) in iterable if conditional}
	Example:
	Common Way:
	d = {}
	for i in range(1,10):
	sqr = i*i
	d[i] = i*i
	print(d)
	Using Dict Comprehension:
	d1={n:n*n for n in range(1,10)}
	print (d1)
	Output:
	{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

4. How Memory Managed in Python?

- Memory management in Python involves a **private heap** containing all Python objects and data structures. Interpreter takes care of Python heap and that the programmer has no access to it.
- The allocation of heap space for Python objects is done by **Python memory manager**. The core API of Python provides some tools for the programmer to code reliable and more robust program.
- Python also has a build-in garbage collector which recycles all the unused memory. When an object is no longer referenced by the program, the heap space it occupies can be freed. The garbage collector determines objects which are no longer referenced by the program frees the occupied memory and make it available to the heap space.
- The gc module defines functions to enable /disable garbage collector:

- **gc.enable()** -Enables automatic garbage collection.
- **gc.disable()** - Disables automatic garbage collection.

5. Difference Between Generators and Iterators

Generator	Iterator

● Generators are iterators which can execute only once.
● Generator uses “yield” keyword.
● Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop.
● Every generator is an iterator.
Example:
`def sqr(n):`
`    for i in range(1, n+1):`
`        yield i*i`
`a = sqr(3)`
`print(next(a))`
`print(next(a))`
`print(next(a))`
Output:
1
4
9
	● An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc.
	● Iterators are used mostly to iterate or convert other objects to an iterator using iter() function.
	● Iterator uses iter() and next() functions.
	● Every iterator is not a generator.
	Example:
	`iter_list = iter(['A', 'B', 'C'])`
	`print(next(iter_list))`
	`print(next(iter_list))`
	`print(next(iter_list))`
	Output:
	A
	B
	C

6. What is ‘init’ Keyword In Python?

__init__.py file

The __init__.py file lets the Python interpreter know that a directory contains code for a Python module. It can be blank. Without one, you cannot import modules from another folder into your project.

The role of the __init__.py file is like the __init__ function in a Python class. The file essentially the constructor of your package or directory without it being called such. It sets up how packages or functions will be imported into your other files.

__init__() function

```
# A Sample class with init method
class Person:
  # init method or constructor
  def __init__(self, name):
    self.name = name

  # Sample Method
  def say_hi(self):
    print('Hello, my name is', self.name)
```

```
p = Person('Nitin')
p.say_hi()
```

7. Difference Between Modules and Packages in Python

Module	Package
The module is a simple Python file that contains collections of functions and global variables and with having a .py extension file. It is an executable file and to organize all the modules we have the concept called **Package** in Python. A module is a single file (or files) that are imported under one import and used. E.g. import my_module	The package is a simple directory having collections of modules. This directory contains Python modules and having __init__.py file by which the interpreter interprets it as a Package. The package is simply a namespace. The package also contains sub-packages inside it. A package is a collection of modules in directories that give a package hierarchy. E.g. from my_package.abc import a

8. Difference Between Range and Xrange?

Parameters	Range()	Xrange()
Return type	It returns a list of integers.	It returns a generator object.
Memory Consumption	Since range() returns a list of elements, it takes more memory.	In comparison to range(), it takes less memory.
Speed	Its execution speed is slower.	Its execution speed is faster.
Python Version	Python 2, Python 3	xrange no longer exists.
Operations	Since it returns a list, all kinds of arithmetic operations can be performed.	Such operations cannot be performed on xrange().

9. What are Generators. Explain it with Example.

- Generators are iterators which can execute only once.
- Every generator is an iterator.
- Generator uses “yield” keyword.
- Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop

Example:

```
def sqr(n):
  for i in range(1, n+1):
    yield i*i
a = sqr(3)
print("The square are : ")
print(next(a))
print(next(a))
print(next(a))
```

Output:

```
The square are :
1
4
9
```

10. What are in-built Data Types in Python OR Explain Mutable and Immutable Data Types

Data Type	Mutable Or Immutable?
Boolean (bool)	Immutable
Integer (int)	Immutable
Float	Immutable
String (str)	Immutable
tuple	Immutable
frozenset	Immutable
list	Mutable
set	Mutable
dict	Mutable

11. Explain Ternary Operator in Python?

In Python, the **ternary operator** provides a compact way to write simple if-else conditions in a single line. It is often used for conditional assignments or quick checks where you want to assign one value if a condition is true, and another if it is false.

```
value_if_true if condition else value_if_false

age = 18
status = "Adult" if age >= 18 else "Minor"
print(status)
```

12. What is Inheritance in Python

Inheritance in Python:

Inheritance is an OOP concept in Python that allows a class (called the child or derived class) to inherit attributes and methods from another class (called the parent or base class). This promotes code reuse and helps to create a hierarchy of classes.

Example Explanation:

In the example below, Dog is a child class that inherits from the Animal parent class. The Dog class gains access to Animal's speak method but can also have additional methods or override inherited ones.

```
# Parent Class
class Animal:
    def speak(self):
        return "Some sound"

# Child Class
class Dog(Animal):
    def bark(self):
        return "Woof!"
```

Explanation:

In this code:

- Dog inherits the speak method from Animal, so calling my_dog.speak() works without needing to redefine speak in Dog.
- This shows how inheritance helps reuse code from the parent class in the child class.

```
# Creating an object of Dog
my_dog = Dog()
print(my_dog.speak())  # Inherited method
print(my_dog.bark())   # Child-specific method
```

13. Difference Between Local and Global Variable in Python

Local Variable	Global Variable
1. It is declared inside a function.	1. It is declared outside the function.
2. If it is not initialized, a garbage value is stored	2. If it is not initialized zero is stored as default.
3. It is created when the function starts execution and lost when the functions terminate.	3. It is created before the program's global execution starts and lost when the program terminates.
4. Data sharing is not possible as data of the local variable can be accessed by only one function.	4. Data sharing is possible as multiple functions can access the same global variable.
5. Parameters passing is required for local variables to access the value in other function	5. Parameters passing is not necessary for a global variable as it is visible throughout the program
6. When the value of the local variable is modified in one function, the changes are not visible in another function.	6. When the value of the global variable is modified in one function changes are visible in the rest of the program.
7. Local variables can be accessed with the help of statements, inside a function in which they are declared.	7. You can access global variables by any statement in the program.
8. It is stored on the stack unless specified.	8. It is stored on a fixed location decided by the compiler.

14. Explain Break, Continue and Pass Statement

Break	Continue	Pass
The break statement is used to exit a loop prematurely when a certain condition is met. Once break is executed, the loop stops, and the program moves to the next line of code after the loop.	The continue statement skips the current iteration of the loop and moves directly to the next iteration, without completing the remaining code inside the loop for that iteration.	The pass statement is a placeholder and does nothing when executed. It is used when a statement is syntactically required but no action is needed. Typically, pass is used in empty functions, loops, or conditionals that you plan to implement later.

for i in range(5):
  if i == 3:
    break # exits the loop when i equals 3
  print(i) # Output: 0 1 2
	for i in range(5):
	  if i == 2:
	    continue # skips when i equals 2
	  print(i) # Output: 0 1 3 4
		for i in range(5):
		  if i == 2:
		    pass # does nothing, acts as a placeholder
		  print(i) # Output: 0 1 2 3 4

15. What is 'self' Keyword in python?

The **'self'** parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def info(self):
    print(f"My name is {self.name}. I am {self.age} years old.")
c = Person("Nitin",23)
```

```
c.info()
```

Output:

```
My name is Nitin. I am 23 years old.
```

16. Difference Between Pickling and Unpickling?

Pickling	**Un Pickling**
Pickling is the process of converting a Python object (like a dictionary, list, or custom object) into a byte stream. This serialized byte stream can be saved to a file or transferred over a network. The pickle module in Python provides the pickle.dump() method to perform pickling.	Unpickling is the reverse process of pickling, where the byte stream is converted back into a Python object. This allows you to retrieve and use the original data structure. The pickle module provides the pickle.load() method for unpickling.

import pickle
 	
data = {"name": "Alice", "age": 25}
with open("data.pkl", "wb") as file:
   pickle.dump(data, file) # Pickling the data dictionary
	import pickle
	
	with open("data.pkl", "rb") as file:
	   data = pickle.load(file) # Unpickling to retrieve the data dictionary
	print(data) # Output: {'name': 'Alice', 'age': 25}

17. Explain Function of List, Set, Tuple and Dictionary?

Functions Of List

- ❏ sort(): Sorts the list in ascending order.
- ❏ append(): Adds a single element to a list.
- ❏ extend(): Adds multiple elements to a list.
- ❏ index(): Returns the first appearance of the specified value.
- ❏ max(list): It returns an item from the list with max value.
- ❏ min(list): It returns an item from the list with min value.
- ❏ len(list): It gives the total length of the list.
- ❏ list(seq): Converts a tuple into a list.
- ❏ cmp(list1, list2): It compares elements of both lists list1 and list2.
- ❏ type(list): It returns the class type of an object.

Functions Of Tuple

- ❏ cmp(tuple1, tuple2) - Compares elements of both tuples.
- ❏ len(): total length of the tuple.
- ❏ max(): Returns item from the tuple with max value.
- ❏ min(): Returns item from the tuple with min value.
- ❏ tuple(seq): Converts a list into tuple.
- ❏ sum(): returns the arithmetic sum of all the items in the tuple.
- ❏ any(): If even one item in the tuple has a Boolean value of True, it returns True. Otherwise, it returns False.
- ❏ all(): returns True only if all items have a Boolean value of True. Otherwise, it returns False.
- ❏ sorted(): a sorted version of the tuple.
- ❏ index(): It takes one argument and returns the index of the first appearance of an item in a tuple
- ❏ count(): It takes one argument and returns the number of times an item appears in the tuple.

Functions Of Dictionary

- ❏ clear(): Removes all the elements from the dictionary
- ❏ copy(): Returns a copy of the dictionary
- ❏ fromkeys(): Returns a dictionary with the specified keys and value

- ❏ get(): Returns the value of the specified key
- ❏ items(): Returns a list containing a tuple for each key value pair
- ❏ keys(): Returns a list containing the dictionary's keys
- ❏ pop(): Removes the element with the specified key
- ❏ popitem(): Removes the last inserted key-value pair
- ❏ setdefault(): Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
- ❏ update(): Updates the dictionary with the specified key-value pairs
- ❏ values(): Returns a list of all the values in the dictionary
- ❏ cmp(): compare two dictionaries

Functions Of Set

- ❏ add(): Adds an element to the set
- ❏ clear(): Removes all the elements from the set
- ❏ copy(): Returns a copy of the set
- ❏ difference(): Returns a set containing the difference between two or more sets
- ❏ difference_update(): Removes the items in this set that are also included in another, specified set
- ❏ discard(): Remove the specified item
- ❏ intersection(): Returns a set, that is the intersection of two or more sets
- ❏ intersection_update(): Removes the items in this set that are not present in other, specified set(s)
- ❏ isdisjoint(): Returns whether two sets have a intersection or not
- ❏ issubset(): Returns whether another set contains this set or not
- ❏ issuperset(): Returns whether this set contains another set or not
- ❏ pop(): Removes an element from the set
- ❏ remove(): Removes the specified element
- ❏ symmetric_difference(): Returns a set with the symmetric differences of two sets
- ❏ symmetric_difference_update(): inserts the symmetric differences from this set and another
- ❏ union(): Return a set containing the union of sets
- ❏ update(): Update the set with another set, or any other iterable

18. What are Python Iterators?

- An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc.
- Iterators are used mostly to iterate or convert other objects to an iterator using iter() function.
- Iterator uses iter() and next() functions.
- Every iterator is not a generator.

Example:

```
iter_list = iter(['A', 'B', 'C'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Output:

```
A
B
C
```

19. Explain Type Conversion in Python. [(int(), float(), ord(), oct(), str() etc.)]

- ❏ **int**() - Converts any data type into an integer.
- ❏ **float**() - Returns A floating point number from a number or string
- ❏ **oct**() - Returns its octal representation in a string format.
- ❏ **hex**() - Convert the integer into a suitable hexadecimal form for the number of the integer.
- ❏ **ord**() - Returns the integer of the Unicode point of the character in the Unicode case or the byte value in the case of an 8-bit argument.
- ❏ **chr**(number) - Returns the character (string) from the integer (represents unicode code point of the character).
- ❏ **eval**() - Parses the expression argument and evaluates it as a python expression.

- **str**() - Convert a value (integer or float) into a string.
- **repr**() - Returns the string representation of the value passed to eval function by default. For the custom class object, it returns a string enclosed in angle brackets that contains the name and address of the object by default.

20. What does *args and **kwargs mean? Expain

When you are not clear how many arguments you need to pass to a particular function, then we use ***args** and ****kwargs**.
The ***args** keyword represents a varied number of arguments. It is used to add together the values of multiple arguments
The ****kwargs** keyword represents an arbitrary number of arguments that are passed to a function. **kwargs keywords are stored in a dictionary. You can access each item by referring to the keyword you associated with an argument when you passed the argument.

***args Python Example:**	****Kwargs Python Example**

```
def sum(*args):
    total = 0
    for a in args:
        total = total + a
    print(total)
sum(1,2,3,4,5)
```
Output:
15
	```
	def show(**kwargs):
	    print(kwargs)
	show(A=1,B=2,C=3)
	```
	Output:
	{'A': 1, 'B': 2, 'C': 3}

21. What is "Open" and "With" statement in Python?

open Statement	**with Statement**
The open() function is used to open a file in Python. It requires the file name (and path if necessary) as well as a mode (e.g., 'r' for reading, 'w' for writing). The open function returns a file object, which you can use to read or write data to the file. However, when using open() alone, you must remember to close the file afterward with file.close() to avoid memory leaks or data corruption.	The with statement simplifies file handling by automatically closing the file once the code block is exited, even if an error occurs. This is especially useful for managing resources safely. When you use with open(...) as shown below, Python ensures the file is properly closed after the block finishes, so there's no need to call close() manually.

```
file = open("example.txt", "r")
content = file.read()
file.close()  # Closing the file manually
```
	```
	with open("example.txt", "r") as file:
	    content = file.read()
	# file is automatically closed after this block
	```

22. Different Ways To Read And Write In A File In Python?

Syntax of Python open file function:
```
file_object  = open("filename", "mode")
```

- **Read Only (‘r’)** : Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
- **Read and Write (‘r+’)** : Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
- **Write Only (‘w’)** : Open the file for writing. For existing file, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exists
- **Write and Read (‘w+’)** : Open the file for reading and writing. For existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
- **Append Only (‘a’)** : Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

- **Append and Read ('a+')** : Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
- **Text mode ('t'):** meaning \n characters will be translated to the host OS line endings when writing to a file, and back again when reading.
- **Exclusive creation ('x'):** File is created and opened for writing – but only if it doesn't already exist. Otherwise you get a FileExistsError.
- **Binary mode ('b'):** appended to the mode opens the file in binary mode, so there are also modes like 'rb', 'wb', and 'r+b'.

23. What is Pythonpath?

PYTHONPATH is an environment variable which you can set to add additional directories where python will look for modules and packages
The 'PYTHONPATH' variable holds a string with the name of various directories that need to be added to the sys.path directory list by Python.
The primary use of this variable is to allow users to import modules that are not made installable yet.

24. How Exception Handled in Python?

Try: This block will test the exceptional error to occur.
Except: Here you can handle the error.
Else: If there is no exception then this block will be executed.
Finally: Finally block always gets executed either exception is generated or not.

```
try:
     # Some Code....!
except:
     # Optional Block
     # Handling of exception (if required)
else:
     # Some code .....
     # execute if no exception
finally:
     # Some code .....(always executed)
```

25. Difference Between Python 2.0 & Python 3.0

Basis	**Python 3**	**Python 2**

Syntax	`def main():`
	`    print("Hello World!")`
	
	`if __name__== "__main__":`
	`    main()`
		`def main():`
		`   print "Hello World!"`
		
		`if __name__== "__main__":`
		`    main()`

Release Date	2008	2000
Function print	print ("hello")	print "hello"
Division of Integers	Whenever two integers are divided, you get a float value	When two integers are divided, you always provide integer value.
Unicode	In Python 3, default storing of strings is Unicode.	To store Unicode string value, you require to define them with "u".
Syntax	The syntax is simpler and easily understandable.	The syntax of Python 2 was comparatively difficult to understand.

26. What is 'PIP' In Python

Python pip is the package manager for Python packages. We can use pip to install packages that do not come with Python.
The basic syntax of pip commands in command prompt is:

```
>> pip install pandas
```

27. Where Python Is Used?

1. Web Applications
2. Desktop Applications
3. Database Applications
4. Networking Application
5. Machine Learning
6. Artificial Intelligence
7. Data Analysis
8. IOT Applications
9. Games and many more…!

28. How to use F String and Format or Replacement Operator?

F-String:
Introduced in Python 3.6, f-strings (formatted string literals) allow embedding expressions inside string literals with curly braces {}. F-strings are prefixed with f or F and provide a concise, readable way to format strings.

```
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
# Output: My name is Alice and I am 25 years old.
```

.format() Method:
The .format() method is an older way to insert values into a string. Placeholders {} are replaced by the values passed to .format(). You can also use positional or keyword arguments for more flexibility.

```
name = "Alice"
age = 25
print("My name is {} and I am {} years old.".format(name, age))
# Output: My name is Alice and I am 25 years old.
```

Replacement Operator %:
This is a legacy method from earlier Python versions, using % as a placeholder for variables, similar to C-style string formatting. It is less flexible but still works in modern Python.

```
name = "Alice"
age = 25
print("My name is %s and I am %d years old." % (name, age))
# Output: My name is Alice and I am 25 years old.
```

29. How to Get List of all keys in a Dictionary?

Get all keys from a dict	Get all values from a dict

`dct = {'A': 1, 'B': 2, 'C': 3}`
 	
`for key in dct.keys():`
`    print (key)`
	`dct = {'A': 1, 'B': 2, 'C': 3}`
	
	`for key in dct.values():`
	`    print (key)`

30. Difference Between Abstraction and Encapsulation.

Abstraction	**Encapsulation**
1. Abstraction works on the design level.	1. Encapsulation works on the application level.
2. Abstraction is implemented to hide unnecessary data and withdrawing relevant data.	2. Encapsulation is the mechanism of hiding the code and the data together from the outside world or misuse.
3. It highlights what the work of an object instead of how the object works is	3. It focuses on the inner details of how the object works. Modifications can be done later to the settings.

4. Abstraction focuses on outside viewing, for example, shifting the car.	4. Encapsulation focuses on internal working or inner viewing, for example, the production of the car.
5. Abstraction is supported in Java with the interface and the abstract class.	5. Encapsulation is supported using, e.g. public, private and secure access modification systems.
6. In a nutshell, abstraction is hiding implementation with the help of an interface and an abstract class.	6. In a nutshell, encapsulation is hiding the data with the help of getters and setters.

Abstraction:
Abstraction focuses on hiding unnecessary details and exposing only the essential features of an object. It helps simplify complex systems by allowing users to interact only with high-level functionalities. In Python, abstraction is often implemented using abstract classes and methods (from the abc module) or through defining simple methods that perform complex logic internally.
Example:
A Car class could have a drive() method. When a user calls drive, they don’t need to understand the internal mechanics of how the car engine works; they only need to know that drive() will make the car move.

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def drive(self):
        pass

class Car(Vehicle):
    def drive(self):
        return "Car is moving"
```

Encapsulation:
Encapsulation involves bundling data (attributes) and methods that operate on that data within a single class, restricting direct access to some components. This is achieved using access modifiers (_ and __) to make variables and methods private or protected, which helps protect the object’s internal state and ensures controlled access.
Example:
In a BankAccount class, the balance attribute could be private, allowing modification only through specific methods like deposit and withdraw, which enforce validation.

```
class BankAccount:
    def __init__(self, initial_balance):
        self.__balance = initial_balance  # Private attribute

    def deposit(self, amount):
        self.__balance += amount

    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
        else:
            return "Insufficient balance"

    def get_balance(self):
        return self.__balance
```

Summary:
- **Abstraction:** Hides unnecessary details, focusing on what an object does.
- **Encapsulation:** Bundles data and methods, restricting access to ensure data integrity.

31. Does Python Support Multiple Inheritance. (Diamond Problem)

Yes, Python supports multiple inheritance, allowing a class to inherit from more than one base class. However, this can lead to the "diamond problem," which occurs when a class inherits from two classes that have a common ancestor. This can create ambiguity about which parent class's method should be called.

```
class A:
    def greet(self):
        return "Hello from A"

class B(A):
    def greet(self):
        return "Hello from B"

class C(A):
    def greet(self):
        return "Hello from C"

class D(B, C):  # D inherits from both B and C
    pass

# Creating an instance of D
d = D()
print(d.greet())  # Output: Hello from B
```

Explanation of the Diamond Problem:

In the example above:

- D inherits from both B and C, which both inherit from A.
- When d.greet() is called, Python uses the method resolution order (MRO) to determine which greet method to execute. In this case, it follows the order defined in the class declaration (left-to-right), so it calls B's greet method first.

32. How to initialize Empty List, Tuple, Dict and Set?

- **List:** empty_list1 = [] or empty_list2 = list()
- **Tuple:** empty_tuple1 = () or empty_tuple2 = tuple()
- **Dictionary:** empty_dict1 = {} or empty_dict2 = dict()
- **Set:** empty_set1 = set() (use set() for an empty set, not {})

What is a Constructor in Python?

A constructor in Python is a special method that is automatically called when an object of a class is created. It is used to initialize the object's attributes and set up any necessary state for the object. The constructor method in Python is defined using the __init__ method.

Key Points:

1. **Initialization:**
   The primary purpose of a constructor is to initialize the attributes of the class when an object is instantiated. This can include setting default values or accepting parameters to assign to the object's attributes.
2. **Naming Convention:**
   The constructor is always defined with the name __init__ (two underscores before and after).
3. **Parameters:**
   The constructor can take parameters (besides self, which refers to the instance being created) to initialize attributes based on user input or default values.

Example:

Here’s a simple example of a constructor in a class:

```
class Person:
    def __init__(self, name, age):  # Constructor with parameters
        self.name = name  # Initializing instance variable
        self.age = age    # Initializing instance variable
```

```
# Creating an instance of the Person class
person1 = Person("Alice", 30)

# Accessing the attributes
print(person1.name)  # Output: Alice
print(person1.age)   # Output: 30
```

Explanation:

- In this example, the Person class has a constructor __init__ that takes name and age as parameters.
- When you create an instance of Person (e.g., person1 = Person("Alice", 30)), the constructor initializes person1.name to "Alice" and person1.age to 30.
- You can then access these attributes through the instance.

Summary:

- A constructor is a special method in a class used to initialize object attributes.
- It is defined with __init__ and is called automatically when an object is created.

33. Difference Between .py and .pyc

- **.py files** contain the source code of a program. Whereas, **.pyc file** contains the bytecode of your program.
- Python compiles the **.py files** and saves it as **.pyc files** , so it can reference them in subsequent invocations.
- The .pyc contain the compiled bytecode of **Python** source files. This code is then executed by Python's **virtual machine.**

34. How Slicing Works In String Manipulation. Explain.

Syntax: Str_Object[Start_Position:End_Position:Step]

s = 'HelloWorld'

Indexing:

H	E	L	L	O	W	O	R	L	D
0	1	2	3	4	5	6	7	8	9
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

1.	print(s[:])	**HelloWorld**
2.	print(s[::])	**HelloWorld**
3.	print(s[:5])	**Hello**
4.	print(s[2:5])	**llo**
5.	print(s[2:8:2])	**loo**
6.	print(s[8:1:-1])	**lroWoll**
7.	print(s[-4:-2])	**or**
8.	print(s[::-1])	**dlroWolleh**

35. Can You Concatenate Two Tuples. If Yes, How Is It Possible? Since it is Immutable?

Yes, you can concatenate two tuples in Python. Although tuples are immutable, meaning that you cannot change their contents after they are created, you can create a new tuple that combines the elements of the original tuples. Concatenation does not modify the existing tuples but instead produces a new tuple as the result.

```
# Defining two tuples
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)

# Concatenating the tuples
concatenated_tuple = tuple1 + tuple2

# Outputting the result
print(concatenated_tuple)  # Output: (1, 2, 3, 4, 5, 6)
```

36. Difference Between Python Arrays and Lists

LIST	ARRAY
The list can store the value of different types.	It can only consist of value of same type.
The list cannot handle the direct arithmetic operations.	It can directly handle arithmetic operations.
The lists are the build-in data structure so we do not need to import it.	We need to import the array before work with the array
The lists are less compatible than the array to store the data.	An array are much compatible than the list.
It consumes a large memory.	It is a more compact in memory size comparatively list.
It is suitable for storing the longer sequence of the data item.	It is suitable for storing shorter sequence of data items.
We can print the entire list using explicit looping.	We can print the entire list without using explicit looping.

37. What Is _a, __a, __a__ in Python?

_a

- Python does not have real private methods, so one underline in the beginning of a variable/function/method name means it's a private variable/function/method and it is for internal use only
- We also call it weak Private

__a

- Leading double underscore tell python interpreter to rewrite name to avoid conflict in subclass.
- Interpreter changes variable name with class extension and that feature known as the **Mangling**.
- In Mangling python interpreter modify variable name with __.
- So Multiple time It use as the Private member because another class cannot access that variable directly.
- Main purpose for __ is to use variable/method in class only If you want to use it outside of the class you can make public api.

__a__

- Name with start with __ and ends with same considers special methods in Python.
- Python provide these methods to use it as the operator overloading depending on the user.
- Python provides this convention to differentiate between the user defined function with the module's function

38. How To Read Multiple Values from Single Input?

In Python, you can read multiple values from a single input line by using the input() function and then splitting the input string into separate values. The most common way to do this is to use the split() method, which divides the input string into a list of substrings based on whitespace or a specified delimiter.

```
# Prompting the user for input
user_input = input("Enter multiple values separated by space: ")

# Splitting the input into a list
values = user_input.split()

# Outputting the results
print("You entered:", values)
```

39. How To Copy and Delete a Dictionary

Copy A Dictionary Using copy():
d2 = {'A':1,'B':2,'C':3}
print(d2) # {'A': 1, 'B': 2, 'C': 3}
d3 = d2.copy()
print(d3) # {'A': 1, 'B': 2, 'C': 3}
Copy A Dictionary Using '=':
d2 = {'A':1,'B':2,'C':3}
print(d2) # {'A': 1, 'B': 2, 'C': 3}
d3 = d2
print(d3) # {'A': 1, 'B': 2, 'C': 3}
	Delete By Using clear():
	d1 = {'A':1,'B':2,'C':3}
	d1.clear()
	print(d1) #{}
	Delete By Using pop():
	d1 = {'A':1,'B':2,'C':3}
	print(d1) #{'A': 1, 'B': 2, 'C': 3}
	d1.pop('A')
	print(d1) # {'B': 2, 'C': 3}
	Delete By Using del():
	del d1['B']
	print(d1) # {'C': 3}

40. Difference Between Anonymous and Lambda Function

Lambda function:

- It can have any number of arguments but only one expression.
- The expression is evaluated and returned.
- Lambda functions can be used wherever function objects are required.

Anonymous function:

- In Python, **Anonymous function** is a function that is defined without a name.
- While normal functions are defined using the **def** keyword, Anonymous functions are defined using the **lambda** keyword.
- Hence, anonymous functions are also called lambda functions.

Advance Questions / Rarely asked

41. How to achieve Multiprocessing and Multithreading in Python?

Multithreading:

- It is a technique where multiple threads are spawned by a process to do different tasks, at about the same time, just one after the other.
- This gives you the illusion that the threads are running in parallel, but they are actually run in a concurrent manner.
- In Python, the Global Interpreter Lock (GIL) prevents the threads from running simultaneously.

Multiprocessing:

- It is a technique where parallelism in its truest form is achieved.
- Multiple processes are run across multiple CPU cores, which do not share the resources among them.
- Each process can have many threads running in its own memory space.
- In Python, each process has its own instance of Python interpreter doing the job of executing the instructions.

42. What is GIL. Explain

- The Global Interpreter Lock (GIL) of Python allows only one thread to be executed at a time. It is often a hurdle, as it does not allow multi-threading in python to save time
- The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.
- This means that only one thread can be in a state of execution at any point in time. The impact of the GIL is not visible to developers who execute single-threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.
- Since the GIL allows only one thread to execute at a time even in a multi-threaded architecture with more than one CPU core, the GIL has gained a reputation as an “infamous” feature of Python.
- Basically, GIL in Python does not allow multi-threading which can sometimes be considered as a disadvantage.

43. How Class and Object Created in Python?

- Python is an object-oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class:
To create a class, use the keyword ‘class’:

```
class MyClass:
  x = 5
```

Create Object:
Now we can use the class named MyClass to create objects:
(Create an object named obj, and print the value of x:)

```
obj= MyClass()
print(obj.x)
```

44. Explain Namespace and Its Types in Python.

Namespace:

- In python we deal with variables, functions, libraries, and modules etc.
- There is a chance the name of the variable you are going to use is already existing as name of another variable or as the name of another function or another method.
- In such scenario, we need to learn about how all these names are managed by a python program. This is the concept of **namespace**.

Categories Of Namespace: Following are the three categories of namespace

- **Local Namespace**: All the names of the functions and variables declared by a program are held in this namespace. This namespace exists if the program runs.
- **Global Namespace**: This namespace holds all the names of functions and other variables that are included in the modules being used in the python program. It includes all the names that are part of the Local namespace.
- **Built-in Namespace**: This is the highest level of namespace which is available with default names available as part of the python interpreter that is loaded as the programing environment. It include Global Namespace which in turn include the local namespace.

We can access all the names defined in the built-in namespace as follows.

```
builtin_names = dir(__builtins__)
for name in builtin_names:
   print(name)
```

45. Explain Recursion by Reversing a List.

```
def reverseList(lst):
   if not lst:
      return []
   return [lst[-1]] + reverseList(lst[:-1])
print(reverseList([1, 2, 3, 4, 5]))
Output:
[5,4,3,2,1]
```

46. What are Unit tests in Python

Unit Testing is the first level of software testing where the smallest testable parts of a software are tested. This is used to validate that each unit of the software performs as designed. The unittest test framework is python's xUnit style framework. This is how you can import it.

import unit test

- Unit testing is a software testing method by which individual units of source code are put under various tests to determine whether they are fit for use (Source). It determines and ascertains the quality of your code.
- Generally, when the development process is complete, the developer codes criteria, or the results that are known to be potentially practical and useful, into the test script to verify a particular unit's correctness. During test case execution, various frameworks log tests that fail any criterion and report them in a summary.
- The developers are expected to write automated test scripts, which ensures that each section or a unit meets its design and behaves as expected.
- Though writing manual tests for your code is a tedious and time-consuming task, Python's built-in unit testing framework has made life a lot easier.
- The unit test framework in Python is called unit test, which comes packaged with Python.
- Unit testing makes your code future proof since you anticipate the cases where your code could potentially fail or produce a bug. Though you cannot predict all the cases, you still address most of them.

47. How to use Map, Filter and Reduce Function in Python?

Map() Function

The map() function iterates through all items in the given iterable and executes the function we passed as an argument on each of them.

The syntax is:

map(function, iterable(s))

```
fruit = ["Apple", "Banana", "Pear"]
map_object = map(lambda s: s[0] == "A", fruit)
print(list(map_object))
```

Output:

```
[True, False, False]
```

Filter() Function

The filter() function takes a function object and an iterable and creates a new list.

As the name suggests, filter() forms a new list that contains only elements that satisfy a certain condition, i.e. the function we passed returns True.

The syntax is:

filter(function, iterable(s))

```
fruit = ["Apple", "Banana", "Pear"]
filter_object = filter(lambda s: s[0] == "A", fruit)
print(list(filter_object))
```

Output:

```
['Apple', 'Apricot']
```

Reduce() Function

The reduce() Function works differently than map() and filter(). It does not return a new list based on the function and iterable we've passed. Instead, it returns a single value.

Also, in Python 3 reduce() isn't a built-in function anymore, and it can be found in the functools module.

The syntax is:

reduce(function, sequence[, initial])

```
from functools import reduce
list = [2, 4, 7, 3]
print(reduce(lambda x, y: x + y, list))
print("With an initial value: " + str(reduce(lambda x, y: x + y, list, 10)))
```

Output:
```
16
With an initial value: 26
```

48. Difference Between Shallow Copy and Deep Copy

- **Shallow Copy** (copy.copy()): Creates a new object but copies references to nested objects. Modifications to nested objects will reflect in both the original and the copied object.

```
import copy

# Original list with nested objects
original_list = [1, 2, [3, 4]]

# Creating a shallow copy
shallow_copied_list = copy.copy(original_list)

# Modifying a nested object
shallow_copied_list[2][0] = 'changed'

print("Original List:", original_list)       # Output: Original List: [1, 2, ['changed', 4]]
print("Shallow Copied List:", shallow_copied_list)  # Output: Shallow Copied List: [1, 2, ['changed', 4]]
```

- **Deep Copy** (copy.deepcopy()): Creates a new object and recursively copies all nested objects. Modifications to nested objects in the copied object do not affect the original object.

```
import copy

# Original list with nested objects
original_list = [1, 2, [3, 4]]

# Creating a deep copy
deep_copied_list = copy.deepcopy(original_list)

# Modifying a nested object
deep_copied_list[2][0] = 'changed'

print("Original List:", original_list)       # Output: Original List: [1, 2, [3, 4]]
print("Deep Copied List:", deep_copied_list)  # Output: Deep Copied List: [1, 2, ['changed', 4]]
```

49. How An Object Be Copied in Python

Refer above answer.

50. What does term MONKEY PATCHING refer to in python?

Definition: Monkey patching is a technique in Python (and other programming languages) where you modify or extend the behaviour of libraries or classes at runtime without altering the original source code. This can be useful for making quick fixes or enhancements to existing code, particularly in third-party libraries.

```
# Original class
class MyClass:
    def greet(self):
        return "Hello!"

# Function to monkey patch
def new_greet(self):
    return "Hi there!"

# Applying monkey patch
```

```
MyClass.greet = new_greet

# Using the patched class
obj = MyClass()
print(obj.greet())  # Output: Hi there!
```

In this example:

- We have a MyClass with a greet() method that returns "Hello!".
- We define a new function new_greet() that returns "Hi there!".
- We then replace the original greet() method in MyClass with our new function.
- When we create an instance of MyClass and call greet(), it now uses the patched method.

51. What is Operator Overloading & Dunder Method.

- Dunder methods in Python are special methods.
- In Python, we sometimes see method names with a double underscore (__), such as the __init__ method that every class has. These methods are called "dunder" methods.
- In Python, Dunder methods are used for operator overloading and customizing some other function's behavior.

Some Examples:

+	__add__(self, other)
–	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)
>>	__rshift__(self, other)
<<	__lshift__(self, other)
&	__and__(self, other)
|	__or__(self, other)
^	__xor__(self, other)

52. Draw Pattern.

```
# This is the example of print simple pyramid pattern
n = int(input("Enter the number of rows"))
# outer loop to handle number of rows
for i in range(0, n):
    # inner loop to handle number of columns
    # values is changing according to outer loop
        for j in range(0, i + 1):
            # printing stars
            print("* ", end="")

        # ending line after each row
        print()
```

Output:

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