GROWAI
AN EDTECH COMPANY

HOW TO CRACK PYTHON INTERVIEW

PYTHON INTERVIW QUESTION

1. What is Python, and list some of its key features.

Python is a versatile, high-level programming language known for its easy-to-read syntax and broad applications. Here are some of Python's key features:

- Simple and Readable Syntax: Python's syntax is clear and straightforward, making it accessible for beginners and efficient for experienced developers.
- Interpreted Language: Python executes code line by line, which helps in debugging and testing.
- Dynamic Typing: Python does not require explicit data type declarations, allowing more flexibility.
- Extensive Libraries and Frameworks: Libraries like NumPy, Pandas, and Django expand Python's functionality for specialized tasks in data science, web development, and more.

PYTHON INTERVIW QUESTION

- Cross-Platform Compatibility: Python can run on different operating systems, including Windows, macOS, and Linux.

2. What are Python lists and tuples?

Lists and tuples are fundamental Python data structures with distinct characteristics and use cases.

List:

- Mutable: Elements can be changed after creation.
- Memory Usage: Consumes more memory.
- Performance: Slower iteration compared to tuples but better for insertion and deletion operations.
- Methods: Offers various built-in methods for manipulation.

Example:

```
a_list = ["Data", "Camp", "Tutorial"]
a_list.append("Session")
print(a_list)  # Output: ['Data', 'Camp', 'Tutorial', 'Session']
```

PYTHON INTERVIW QUESTION

Tuple:

- Immutable: Elements cannot be changed after creation.
- Memory Usage: Consumes less memory.
- Performance: Faster iteration compared to lists but lacks the flexibility of lists.
- Methods: Limited built-in methods.

Example:

```
a_tuple = ("Data", "Camp", "Tutorial")
print(a_tuple)  # Output: ('Data', 'Camp', 'Tutorial')
```

3. What is __init__() in Python?

The __init__() method is known as a constructor in object-oriented programming (OOP) terminology. It is used to initialize an object's state when it is created. This method is automatically called when a new instance of a class is instantiated.

Purpose:

- Assign values to object properties.
- Perform any initialization operations.

PYTHON INTERVIW QUESTION

Example:

We have created a book_shop class and added the constructor and book() function. The constructor will store the book title name and the book() function will print the book name. To test our code we have initialized the b object with "Sandman" and executed the book() function.

```
class book_shop:

    # constructor
    def __init__(self, title):
        self.title = title

    # Sample method
    def book(self):
        print('The tile of the book is', self.title)


b = book_shop('Sandman')
b.book()
# The tile of the book is Sandman
```

PYTHON INTERVIW QUESTION

4. What is the difference between a mutable data type and an immutable data type?

Mutable data types:

- Definition: Mutable data types are those that can be modified after their creation.
- Examples: List, Dictionary, Set.
- Characteristics: Elements can be added, removed, or changed.
- Use Case: Suitable for collections of items where frequent updates are needed.

Example:


```
# List Example
a_list = [1, 2, 3]
a_list.append(4)
print(a_list)  # Output: [1, 2, 3, 4]

# Dictionary Example
a_dict = {'a': 1, 'b': 2}
a_dict['c'] = 3
print(a_dict)  # Output: {'a': 1, 'b': 2, 'c': 3}
```

PYTHON INTERVIW QUESTION

Immutable data types:

- Definition: Immutable data types are those that cannot be modified after their creation.
- Examples: Numeric (int, float), String, Tuple.
- Characteristics: Elements cannot be changed once set; any operation that appears to modify an immutable object will create a new object.

Example:

```
# Numeric Example
a_num = 10
a_num = 20  # Creates a new integer object
print(a_num)  # Output: 20

# String Example
a_str = "hello"
a_str = "world"  # Creates a new string object
print(a_str)  # Output: world

# Tuple Example
a_tuple = (1, 2, 3)
# a_tuple[0] = 4  # This will raise a TypeError
print(a_tuple)  # Output: (1, 2, 3)
```

PYTHON INTERVIW QUESTION

5. Explain list, dictionary, and tuple comprehension with an example.

List

List comprehension offers one-liner syntax to create a new list based on the values of the existing list. You can use a for loop to replicate the same thing, but it will require you to write multiple lines, and sometimes it can get complex.

List comprehension eases the creation of the list based on existing iterable.

```
my_list = [i for i in range(1, 10)]
my_list
# [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Dictionary

Similar to a List comprehension, you can create a dictionary based on an existing table with a single line of code. You need to enclose the operation with curly brackets {}.

PYTHON INTERVIW QUESTION

```
# Creating a dictionary using dictionary comprehension
my_dict = {i: i**2 for i in range(1, 10)}

# Output the dictionary
my_dict

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Tuple

It is a bit different for Tuples. You can create Tuple comprehension using round brackets (), but it will return a generator object, not a tuple comprehension.

You can run the loop to extract the elements or convert them to a list.

```
my_tuple = (i for i in range(1, 10))
my_tuple
# <generator object <genexpr> at 0x7fb91b151430>
```

PYTHON INTERVIW QUESTION

6. Can you explain common searching and graph traversal algorithms in Python?

Python has a number of different powerful algorithms for searching and graph traversal, and each one deals with different data structures and solves different problems. I can them here:

- Binary Search: If you need to quickly find an item in a sorted list, **binary search** is your go-to. It works by repeatedly dividing the search range in half until the target is found.
- AVL Tree: An **AVL tree** keeps things balanced, which is a big advantage if you're frequently inserting or deleting items in a tree. This self-balancing binary search tree structure keeps searches fast by making sure the tree never gets too skewed.

PYTHON INTERVIW QUESTION

- Breadth-First Search (BFS): **BFS** is all about exploring a graph level by level. It's especially useful if you're trying to find the shortest path in an unweighted graph since it checks all possible moves from each node before going deeper.
- Depth-First Search (DFS): **DFS** takes a different approach by exploring as far as it can down each branch before backtracking. It's great for tasks like maze-solving or tree traversal.
- A Algorithm*: The **A* algorithm** is a bit more advanced and combines the best of both BFS and DFS by using heuristics to find the shortest path efficiently. It's commonly used in pathfinding for maps and games.

PYTHON INTERVIW QUESTION

7. What is a KeyError in Python, and how can you handle it?

A KeyError in Python occurs when you try to access a key that doesn't exist in a dictionary. This error is raised because Python expects every key you look up to be present in the dictionary, and when it isn't, it throws a KeyError.

For example, if you have a dictionary of student scores and try to access a student who isn't in the dictionary, you'll get a KeyError. To handle this error, you have a few options:

- Use the .get() method: This method returns None (or a specified default value) instead of throwing an error if the key isn't found.
- Use a try-except block: Wrapping your code in try-except allows you to catch the KeyError and handle it gracefully.
- Check for the key with in: You can check if a key exists in the dictionary using if key in dictionary before trying to access it.

PYTHON INTERVIW QUESTION

8. How does Python handle memory management, and what role does garbage collection play?

Python manages memory allocation and deallocation automatically using a private heap, where all objects and data structures are stored. The memory management process is handled by Python's memory manager, which optimizes memory usage, and the garbage collector, which deals with unused or unreferenced objects to free up memory.

Garbage collection in Python uses reference counting as well as a cyclic garbage collector to detect and collect unused data. When an object has no more references, it becomes eligible for garbage collection. The **gc** module in Python allows you to interact with the garbage collector directly, providing functions to enable or disable garbage collection, as well as to perform manual collection.

PYTHON INTERVIW QUESTION

9. What is the difference between shallow copy and deep copy in Python, and when would you use each?

In Python, shallow and deep copies are used to duplicate objects, but they handle nested structures differently.

- Shallow Copy: A shallow copy creates a new object but inserts references to the objects found in the original. So, if the original object contains other mutable objects (like lists within lists), the shallow copy will reference the same inner objects. This can lead to unexpected changes if you modify one of those inner objects in either the original or copied structure. You can create a shallow copy using the copy() method or the copy module's copy() function.
- Deep Copy: A deep copy creates a new object and recursively copies all objects found within the original.

PYTHON INTERVIW QUESTION

This means that even nested structures get duplicated, so changes in one copy don't affect the other. To create a deep copy, you can use the copy module's deepcopy() function.

Example Usage: A shallow copy is suitable when the object contains only immutable items or when you want changes in nested structures to reflect in both copies. A deep copy is ideal when working with complex, nested objects where you want a completely independent duplicate. Read our **Python Copy List: What You Should Know** tutorial to learn more. This tutorial includes a whole section on the difference between shallow copy and deep copy.

10. What is monkey patching in Python?

Monkey patching in Python is a dynamic technique that can change the behavior of the code at run-time. In short, you can modify a class or module at run-time.

PYTHON INTERVIW QUESTION

Example:

Let's learn monkey patching with an example.

1. We have created a class monkey with a patch() function. We have also created a monk_p function outside the class.
2. We will now replace the patch with the monk_p function by assigning monkey.patch to monk_p.
3. In the end, we will test the modification by creating the object using the monkey class and running the patch() function.

Instead of displaying patch() is being called, it has displayed monk_p() is being called.

```
class monkey:
    def patch(self):
          print ("patch() is being called")

def monk_p(self):
    print ("monk_p() is being called")

# replacing address of "patch" with "monk_p"
monkey.patch = monk_p

obj = monkey()

obj.patch()
# monk_p() is being called
```

PYTHON INTERVIW QUESTION

11. What is the Python "with" statement designed for?

The **with** statement is used for exception handling to make code cleaner and simpler. It is generally used for the management of common resources like creating, editing, and saving a file.

Example:

Instead of writing multiple lines of open, try, finally, and close, you can create and write a text file using the **with** statement. It is simple.

```
# using with statement
with open('myfile.txt', 'w') as file:
    file.write('DataCamp Black Friday Sale!!!')
```

12. Why use else in try/except construct in Python?

try: and **except**: are commonly known for exceptional handling in Python, so where does **else**: come in handy? **else**: will be triggered when no exception is raised.

PYTHON INTERVIW QUESTION

Example:

Let's learn more about **else**: with a couple of examples.

1. On the first try, we entered **2** as the numerator and **d** as the denominator. Which is incorrect, and **except**: was triggered with "Invalid input!".
2. On the second try, we entered **2** as the numerator and **1** as the denominator and got the result **2**. No exception was raised, so it triggered the **else**: printing the message **Division is successful.**

```
try:
    num1 = int(input('Enter Numerator: '))
    num2 = int(input('Enter Denominator: '))
    division = num1/num2
    print(f'Result is: {division}')
except:
    print('Invalid input!')
else:
    print('Division is successful.')


## Try 1 ##
# Enter Numerator: 2
# Enter Denominator: d
# Invalid input!

## Try 2 ##
# Enter Numerator: 2
# Enter Denominator: 1
# Result is: 2.0
# Division is successful.
```

PYTHON INTERVIW QUESTION

13. What are decorators in Python?

Decorators in Python are a design pattern that allows you to add new functionality to an existing object without modifying its structure. They are commonly used to extend the behavior of functions or methods.

Example:

```
[ ]  def my_decorator(func):
         def wrapper():
             print("Something is happening before the function is called.")
             func()
             print("Something is happening after the function is called.")
         return wrapper

     @my_decorator
     def say_hello():
         print("Hello!")

     say_hello()
     # Output:
     # Something is happening before the function is called.
     # Hello!
     # Something is happening after the function is called.
```

PYTHON INTERVIW QUESTION

14. What are context managers in Python, and how are they implemented?

Context managers in Python are used to manage resources, ensuring that they are properly acquired and released. The most common use of context managers is the **with** statement.

Example:

```
class FileManager:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        self.file.close()

with FileManager('test.txt', 'w') as f:
    f.write('Hello, world!')
```

PYTHON INTERVIW QUESTION

15. Given a positive integer num, write a function that returns True if num is a perfect square else False.

This has a relatively straightforward solution. You can check if the number has a perfect square root by:

1. Finding the square root of the number and converting it into an integer.
2. Applying the square to the square root number and checking if it's a perfect square root.
3. Returning the result as a boolean.

Test 1

We have provided number 10 to the **valid_square()** function.

1. By taking the square root of the number, we get 3.1622776601683795.
2. By converting it into an integer, we get 3.
3. Then, take the square of 3 and get 9.
4. 9 is not equal to the number, so the function will return False.

PYTHON INTERVIW QUESTION

Test 2

We have provided number 36 to the **valid_square()** function.

1. By taking the square root of the number, we get 6.
2. By converting it into an integer, we get 6.
3. Then, take the square of 6 and get 36.
4. 36 is equal to the number, so the function will return True.

```
def valid_square(num):
    square = int(num**0.5)
    check = square**2==num
    return check

valid_square(10)
# False
valid_square(36)
# True
```

PYTHON INTERVIW QUESTION

16. Given an integer n, return the number of trailing zeroes in n factorial n!

To pass this challenge, you have to first calculate n factorial (n!) and then calculate the number of training zeros.

Finding factorial

In the first step, we will use a while loop to iterate over the n factorial and stop when the n is equal to 1.

Calculating trailing zeros

In the second step, we will calculate the trailing zero, not the total number of zeros. There is a huge difference.

```
7! = 5040
```

The seven factorials have a total of two zeros and only one trailing zero, so our solution should return 1.

1. Convert the factorial number to a string.
2. Read it back and apply for a loop.
3. If the number is 0, add +1 to the result, otherwise break the loop.
4. Returns the result.

PYTHON INTERVIW QUESTION

The solution is elegant but requires attention to detail.

```python
def factorial_trailing_zeros(n):

    fact = n
    while n > 1:
        fact *= n - 1
        n -= 1

    result = 0

    for i in str(fact)[::-1]:
        if i == "0":
            result += 1
        else:
            break

    return result


factorial_trailing_zeros(10)
# 2
factorial_trailing_zeros(18)
# 3
```

PYTHON INTERVIW QUESTION

17. Can you find the maximum single sell profit?

You are provided with the list of stock prices, and you have to return the buy and sell price to make the highest profit.

Note: We have to make maximum profit from a single buy/sell, and if we can't make a profit, we have to reduce our losses.

Example 1: stock_price = [8, 4, 12, 9, 20, 1], buy = 4, and sell = 20. Maximizing the profit.

Example 2: stock_price = [8, 6, 5, 4, 3, 2, 1], buy = 6, and sell = 5. Minimizing the loss.

Solution:

- We will calculate the global profit by subtracting global sell (the first element in the list) from current buy (the second element in the list).

PYTHON INTERVIW QUESTION

- Run the loop for the range of 1 to the length of the list.
- Within the loop, calculate the current profit using list elements and current buy value.
- If the current profit is greater than the global profit, change the global profit with the current profit and global sell to the i element of the list.
- If the current buy is greater than the current element of the list, change the current buy with the current element of the list.
- In the end, we will return global buy and sell value. To get global buy value, we will subtract global sell from global profit.

The question is a bit tricky, and you can come up with your unique algorithm to solve the problems.

PYTHON INTERVIW QUESTION

```python
def buy_sell_stock_prices(stock_prices):
    current_buy = stock_prices[0]
    global_sell = stock_prices[1]
    global_profit = global_sell - current_buy

    for i in range(1, len(stock_prices)):
        current_profit = stock_prices[i] - current_buy

        if current_profit > global_profit:
            global_profit = current_profit
            global_sell = stock_prices[i]

        if current_buy > stock_prices[i]:
            current_buy = stock_prices[i]

    return global_sell - global_profit, global_sell

stock_prices_1 = [10,9,16,17,19,23]
buy_sell_stock_prices(stock_prices_1)
# (9, 23)
```

```python
stock_prices_2 = [8, 6, 5, 4, 3, 2, 1]
buy_sell_stock_prices(stock_prices_2)
# (6, 5)
```

PYTHON INTERVIW QUESTION

18. How will you check if a class is a child of another class?

This is done by using a method called issubclass() provided by python. The method tells us if any class is a child of another class by returning true or false accordingly.

For example:

```
class Parent(object):
    pass

class Child(Parent):
    pass

# Driver Code
print(issubclass(Child, Parent))    #True
print(issubclass(Parent, Child))    #False
```

- We can check if an object is an instance of a class by making use of isinstance() method:

```
obj1 = Child()
obj2 = Parent()
print(isinstance(obj2, Child))    #False
print(isinstance(obj2, Parent))   #True
```

PYTHON INTERVIW QUESTION

19. What is init method in python?

The init method works similarly to the constructors in Java. The method is run as soon as an object is instantiated. It is useful for initializing any attributes or default behaviour of the object at the time of instantiation.

For example:

```
class InterviewbitEmployee:

    # init method / constructor
    def __init__(self, emp_name):
        self.emp_name = emp_name

    # introduce method
    def introduce(self):
        print('Hello, I am ', self.emp_name)

emp = InterviewbitEmployee('Mr Employee')
# __init__ method is called here and initializes the object name with "Mr Employee"
emp.introduce()
```

20. Why is finalize used?

Finalize method is used for freeing up the unmanaged resources and clean up before the garbage collection method is invoked. This helps in performing memory management tasks.

PYTHON INTERVIW QUESTION

21. Explain 'Everything in Python is an object'.

What's an object? An object in a object-oriented programming language is an entity that contains data along with some methods for operations related to that data.

Everything from numbers, lists, strings, functions and classes are python objects.

```
>>> a = 10.5
>>> a.is_integer()
# Float type has is_integer() method cause a is an object of float class
False
>>> type(a)
<class 'float'>
>>> def func()
....    pass
>>> type(func)
<class 'function'>
>>> # like functions, classes are also objects of 'type' class
```

Look at the below example

```
>>> var = 'Tom' # Object 'Tom' is created in memory and name 'var' is binded to
it.
>>> var = 'Harry' # Another object is created however note that name 'var' is
now binded to 'Harry' but 'Tom' is still somewhere in memory and is unaffected.
```

PYTHON INTERVIW QUESTION

22. What is mutable and immutable objects/data types in Python?

Mutation generally refers to 'change'. So when we say that an object is mutable or immutable we meant to say that the value of object can/cannot change.

When an object is created in Python, it is assigned a type and an id. An object/data type is mutable if with the same id, the value of the object changes after the object is created.

Mutable objects in Python -- Objects that can change after creation. Lists, byte arrays, sets, and dictionaries.

```
>>> list_var = [17, 10]
>>> list_var
[17, 10]
>>> id(list_var)
2289772854208
>>> list_var += [17]
>>> list_var
[17, 10, 17]
>>> id(list_var) # ID of the object didn't change
2289772854208
```

PYTHON INTERVIW QUESTION

Immutable objects in Python -- Numeric data types, strings, bytes, frozen sets, and tuples.

```
>>> # Example of tuples
>>> tuple_var = (17,)
>>> tuple_var
(17,)
>>> id(tuple_var)
1753146091504
>>> tuple_var += (10,)
>>> tuple_var
(17,10)
>>> id(tuple_var) # ID changes when made changes in object.
1753153466880
```

Mutable objects and functon arguments

```
def sample_func(sample_arg):
    sample_agr.append(10)
    # No need to return the obj since it is utilizing the same memory block

sample_list = [7, 8, 9]
sample_func(sample_list)
print(sample_list) # [7, 8, 9, 10]
```

PYTHON INTERVIW QUESTION

23. What is the difference between list and tuples in Python?

Parameter	List	Tuples
Syntax	Square brackets or list keyword	Round brackets/parenthesis or tuple keyword
Nature	Mutable	Immutable
Item Assignment	Possible	Not Possible
Reusablity	Copied	Not Copied
Performance	Relatively slower	Relatively faster
Memory	Large-Extra than the element size	Fixed to element size

Note: It is not required for tuples to have parenthesis, one can also define tuple python a = 2, 3, 4

Memory Allocation of Tuple and List

Tuple does not allot any extra memory during construction because it will be immutable so does not have to worry about addition of elements.

```
>>> tuple_var = tuple()
>>> tuple_var.__sizeof__() # take 24 bytes for empty tuple
24
>>> tuple_var = (1,2) # additional 8 bytes for each integer element
>>> tuple_var.__sizeof__()
40
```

List over-allocates memory otherwise list.append would be an O(n) operation.

PYTHON INTERVIW QUESTION

```
>>> list_var = list()
>>> list_var.__sizeof__() # take 40 bytes for empty list
40
>>> list_var.append(1)
>>> list_var.__sizeof__() # append operation allots extra memory size considering future appends
72
>>> list_var
[1]
>>> list_var.append(2)
>>> list_var.__sizeof__() # size remains same since list has space available
72
>>> list_var
[1,2]
```

Reusablity

Tuple literally assigns the same object to the new variable while list basically copies all the elements of the existing list.

```
>>> # List vs Tuples | Reused vs. Copied
>>> old_list = [1,2]
>>> old_list.append(3)
>>> old_list
[1, 2, 3]
>>> id(old_list)
2594206915456
>>> old_list.__sizeof__()
88

>>> # Copying list
>>> new_list = list(old_list)
>>> new_list
[1, 2, 3]
>>> id(new_list) # new id so new list is created
2594207110976
>>> new_list.__sizeof__() # size is also not same as old_list
64
```

PYTHON INTERVIW QUESTION

```
>>> Tuple Copy
>>> old_tuple = (1,2)
>>> id(old_tuple)
2594206778048
>>> old_tuple.__sizeof__()
40
>>> new_tuple = tuple(old_tuple)
>>> id(new_tuple) # same id as old_tuple
2594206778048
>>> new_tuple.__sizeof__() # also same size as old_tuple since it is refering to old_tuple
40
```

24. How is memory managed in Python?

Unlike other programming languages, python stores references to an object after it is created. For example, an integer object 17 might have two names(variables are called names in python) a and b. The memory manager in python keeps track of the reference count of each object, this would be 2 for object 17. Once the object reference count reaches 0, object is removed from memory.

The reference count

- increases if an object is assigned a new name or is placed in a container, like tuple or dictionary.
- decreases when the object's reference goes out of scope or when name is assigned to another object. Python's garbage collector handles the job of removing objects & a programmer need not to worry about allocating/de-allocating memory like it is done in C.

PYTHON INTERVIW QUESTION

```
>>> import sys
>>> sys.getrefcount(17)
>>> 11
>>> a = 17
>>> b = 17
>>> a is b
>>> True
>>> sys.getrefcount(17)
>>> 13 # addition of two
```

25. Explain exception handling in Python.

Exception handling is the way by which a programmer can control an error within the program without breaking out the flow of execution.

```
try:
    # Part which might cause an error
except TypeError:
    # What happens when error occurs | In this case what happens what a TypeError occurs
else:
    # what happens if there is no exception | Optional
finally:
    # Executed after try and except| always executed | Optional
```

Examples :- TypeError, ValueError, ImportError, KeyError, IndexError, NameError, PermissionError, EOFError, ZeroDivisionError, StopIteration

26. Explain some changes in Python 3.8

Positional arguments representation

```
def sum(a,b,/,c=10):
    return a+b+c
sum(10,12,c=12)
```

PYTHON INTERVIW QUESTION

F String can also do operations

```
a,b = 10, 12
f"Sum of a and b is {a+b}"
f"Value of c is {(c := a+b)}"
```

27. How would you load large data file in Python?

The best way is to load data in chunks, Pandas provides option to define chunk size while loading any file.

```
for chunk in pd.read_csv(file.csv, chunksize=1000):
    process(chunk)

pd.read_csv('file.csv', sep='\t', iterator=True, chunksize=1000)
```

Another way is to use context manager.

```
with open("log.txt") as infile:
    for line in infile:
        do_something_with(line)
```


28. Explain Generators and use case of it.

A function or method which uses the yield statement is called a generator function. Such a function, when called, always returns an iterator object which can be used to execute the body of the function: calling the iterator's iterator._next_()method will cause the function to execute until it provides a value using the yield statement.

When the function executes a return statement or falls off the end, a StopIteration exception is raised and the iterator will have reached the end of the set of values to be returned. Note that a generator can have n numbers of yield statements

PYTHON INTERVIW QUESTION

Use Case

Generators are good for calculating large sets of results where you don't know if you are going to need all results, or where you don't want to allocate the memory for all results at the same time.

```
# Search function as generator,
# effective for returning some set as result with functionality like
# 'Load 10 more items'
def search_result(keyword):
    while keyword in dataset:
        yield matched_data

search_object = search_result('keyword')
# type(search_function)  --> <class 'generator'>

search_object.__next__()
```

Note: *You can only iterate over a generator once, if you try to loop over it second time it will return nothing. Generators also do not store all the values in memory, they generate the values on the fly*

```
mygenerator = (x*x for x in range(3))
```

29. Is there a sequence in defining exceptions in except block for exception handling?

Yes can be defined in a tuple. From left to right will be executed based on the exception raised.

```
try:
    pass
except (TypeError, IndexError, RuntimeError) as error:
    pass
```

PYTHON INTERVIW QUESTION

30. Explain Closures in Python

A closure is a functionality in Python where some data is memorized in the memory for lazy execution. Decorators heavily rely on the concept of closure.
To create a closure in python:-

1. There must be a nested function(function within a enclosing/outer function)
2. The nested function must refer to a value defined in the enclosing function
3. The enclosing function must return(not call it) the nested function.

```
def enclosing_function(defined_value):
    def nested_function():
        return defined_value+some_operation
    return nested_function

closure_function = enclosing_function(20)
closure_function() # returns 20+some_operation
```

Objects are data with methods attached, closures are functions with data attached.

PYTHON INTERVIW QUESTION

31. How to make a chain of function decorators?

```
def make_bold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def make_italic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@make_bold
@make_italic
def index():
    return "hello world"

print(index())
## returns "<b><i>hello world</i></b>"
```


32. Three different ways to fetch every 3rd item of a list

Using index jump

```
>>> example_list = [0,1,2,3,4,5,6]
>>> example_list = [::3] # returns [0,3,6]
```

Using list comphrehension

```
>>> [x for x in example_list if example_list.index(x)%3==0]
>>> [0,3,6]
```

PYTHON INTERVIW QUESTION

Using while loop

```
i = 0
while i < len(example_list):
    print(example_list[i])
    i += 3
```

32. What is MRO in Python? How does it work?

Method Resolution Order (MRO) is the order in which Python looks for a method in a hierarchy of classes. Especially it plays vital role in the context of multiple inheritance as single method may be found in multiple super classes.

```
class A:
    def process(self):
        print('A')

class B(A):
    pass

class C(A):
    def process(self):
        print('C')

class D(B,C):
    pass

obj = D()
obj.process()
# D -> B -> C -> A -> object
```

Note: a class can't be called before its superclass in resolving MRO. Super Class has to be called after derived class

PYTHON INTERVIW QUESTION

33. What is monkey patching? How to use it in Python?

Monkey patching refers to the practice of dynamically modifying or extending code at runtime, typically to change the behavior of existing classes, modules, or functions.

Monkey patching can be useful in several scenarios:

1. Testing: It allows to replace parts of code during testing with mock objects or functions.
2. Hotfixes: In situations where you can't immediately deploy a fix to production code, monkey patching can be used as a temporary solution to address critical issues without waiting for a formal release cycle.
3. Extending functionality: It enables to add new features or alter the behavior of existing code without modifying the original source. This can be useful when working with third-party libraries or legacy code that you can't modify directly.


```python
# Original module
class OriginalClass:
    def method(self):
        return "Original method"

# Monkey patching
def new_method(self):
    return "Patched method"

# Patching the class method
OriginalClass.method = new_method
# Using the patched code
obj = OriginalClass()
print(obj.method())  # Output: "Patched method"
```

PYTHON INTERVIW QUESTION

34. What's the difference between a Python module and a Python package?

Module
The module is a Python file that contains collections of functions and global variables and with having a .py extension file.

Package
The package is a directory having collections of modules. This directory contains Python modules and also having init.py file by which the interpreter interprets it as a Package.

35. Which is faster, list comprehension or for loop?

List comprehensions are generally faster than 'a for loop' because of the implementation of both. One key difference is that 'for loop' generally rounds up more than one statement/operation as compared to 'list comprehension' which has to perform single operation on all the elements. For example, creating a list or update in an existing list is faster when done using list comprehension.

PYTHON INTERVIW QUESTION

34. Explain Meta Classes in Python.

In Python everything is an object, even a class is an object. As a result, a class also must have a type. All classes in Python are of 'type' type. Even the class of 'type' is 'type'. So 'type' is the meta class in Python and to create custom meta class, you would need to inherit from 'type'.

Use Case of Meta Class

A meta class is the class of a class. A class is an instance of a metaclass. A metaclass is most commonly used as a class-factory. When you create an object by calling the class, Python creates a new class (when it executes the 'class' statement) by calling the metaclass.

```
>>> type(17) # <class 'int'>
>>> type(int) # <class 'type'>
>>> str.__class__ # <class 'type'>
>>> type.__class__ # <class 'type'>
```

Meta Class call

The metaclass is called with the

- name: name of the class,
- bases: tuple of the parent class (for inheritance, can be empty) and
- attributes: dictionary containing attributes names and values.

PYTHON INTERVIW QUESTION

```
def init(self, make):
    self.make = make

# type(name, bases, attrs)
>>> Car = type('Car', (object,), {'__init__': init, '__repr__': lambda self: self.make,  'wheels': 4})
>>> seltos = Car('Kia')
>>> seltos # Kia
```

35. Best way to concatenate n number of strings together into one.

The best way of appending a string to a string variable is to use + or +=. This is because it's readable and fast. However in most of the codebases we see use of append and join when joining strings together, this is done for readablity and cleaner code. Sometimes it is more important to have code readablity to actually understand the operation.

```
first_name = 'Max '
last_name = 'Verstappen'
full_name = first_name + last_name
# using join string method
full_name = ''.join( (first_name, last_name) )
# takes in tuple of string in case of multiple values
```


36. Explain object creation process in detail. Which method is called first?

When an object of a class is created or a class is instantiated, the _new_() method of class is called. This particular method is resposible for returning a new class object. It can be overriden to implement object creational restrictions on class.

- The constructor of the class is _new_() and
- the initializer of the class is _init_().
- Initializer is called right after the constructor, if the constructor has not returned a class object, the initializer call is useless.

PYTHON INTERVIW QUESTION

Note that the reason _init_() could use class object(self) to initialize is because when the code flow reaches _init_() the object of the class is already created.

37. Explain the concept behind dictionary in Python

- A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.
- A key can appear in a dictionary only once. Duplicate keys are not allowed
- Using a key twice in initial dict definition will override the first entry
- Key must be of a type that is immutable. Values can be anything

```
>>> dict_sample_01 = {1: 12, 2: 14, 1: 16}
>>> dict_sample_02 # {1: 16, 2: 14}
>>> dict_sample_02 = dict.fromkeys('123')
>>> dict_sample_02 # {'1': None, '2': None, '3': None}
```


38. Difference between an expression and a statement in Python

A statement is a complete line of code that performs some action, while an expression is any section of the code that evaluates to a value. An expression is also a statement. Note that lambda function in Python only accepts expressions.

PYTHON INTERVIW QUESTION

39. Can set have lists as elements?

You can't add a list to a set because lists are mutable, meaning that you can change the contents of the list after adding it to the set. You can however add tuples to the set, because you cannot change the contents of a tuple.

The objects have to be hashable so that finding, adding and removing elements can be done faster than looking at each individual element every time you perform these operations.

Some unhashable datatypes:

- list: use tuple instead
- set: use frozenset instead

40. Is method overloading possible in Python?

Yes method overloading is possible in Python. It can be achived using different number of arguments.


```
def increment(value, by=1):
    return value+by

# calling function
increment(5) # returns 6
increment(5, 2) # return 7
```

PYTHON INTERVIW QUESTION

41. What can be used as keys in dictionary?

Any immutable object type can be used as dictionary key even functions and classes can also be used as dictionary keys.

Why can't list or another dict(mutable object) be used as key in dictionary?

Dict implementation reduces the average complexity of dictionary lookups to O(1) by requiring that key objects provide a "hash" function. Such a hash function takes the information in a key object and uses it to produce an integer, called a hash value. This hash value is then used to determine which "bucket" this (key, value) pair should be placed into. Mutuable objects like list or dict cannot provide a valid /hash method.

42. Explain shallow and deep copy in Python

For collections that are mutable or contain mutable items, a copy is sometimes needed so one can change one copy without changing the other.
Python follows a pattern of compiling the original source to byte codes, then interpreting the byte codes on a virtual machine. The .pyc file generated contains byte code.

PYTHON INTERVIW QUESTION

```
>>> import copy
>>> sample_1 = [1,2,3]
>>> id(sample_1)
139865052152768
>>> sample_2 = sample_1
>>> id(sample_2)
139865052152768
>>> sample_3 = copy.copy(sample_1)
>>> id(sample_3)
139865052236736
```

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

43. Why Python generates a .pyc file even when it is an interpreted language?

.pyc files are created by the Python interpreter when a .py file is imported, and they contain the "compiled bytecode" of the imported module/program, the idea being that the "translation" from source code to bytecode (which only needs to be done once) can be skipped on subsequent imports if the .pyc is newer than the corresponding .py file, thus speeding startup a little. But it's still interpreted.

PYTHON INTERVIW QUESTION

43. How private varibles are declared in Python?

Python does not have anything called private member however by convention two underscore before a variable or function makes it private.

```
class XSpecial:
    normal_var = 10
    __private_var = 17

>>> special_obj = XSpecial()
>>> special_obj.normal_var
>>> special_obj.__private_var # AttributeError
```

44. Difference between an array and list

List	Array
Can contain elements of different data types	Contains homogeneous elements only i.e. same data type
No need to import	Need to import via numpy or array
Preferred for short sequence of data items	Preferred for large sequence of data items i.e., data analysis
Can't perform airthmetic operations on whole list	Great for airthmetic operations

45. How do you define a dict where several keys has same value?

```
products = {}
products.update(
    dict.keys(['Apple','Mango','Oranges'], 20)
)
products.update(
    dict.keys(['Pizza','Kind Pizza','Bad Pizza'], 30)
)
```

PYTHON INTERVIW QUESTION

46. What are different types of namespaces in Python?

Namespace is a way to implement scope. In Python, each package, module, class, function and method function owns a "namespace" in which variable names are resolved. Plus there's a global namespace that's used if the name isn't in the local namespace.
Each variable name is checked in the local namespace (the body of the function, the module, etc.), and then checked in the global namespace.

Types of Namespaces

- Local Namespace: The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function.
- Global Namespace: The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.
- Built-in Namespace: The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted.

PYTHON INTERVIW QUESTION

47. How can you access attribute of parent class bypassing the attribute with the same name in derived class?

```
class Parent:
    variable = 12

class Derived(Parent):
    variable = 10

Parent.variable # returns 12
```

48. Evaulation of boolean expressions

- The expression x and y first evaluates x; if x is false, its value is returned; otherwise, y is evaluated and the resulting value is returned.
- The expression x or y first evaluates x; if x is true, its value is returned; otherwise, y is evaluated and the resulting value is returned.

```
x = 'Some Value'
y = 24
z = False
x or y # returns x
z or y # returns y
x and y # returns y
z and x # returns z
```

PYTHON INTERVIW QUESTION

49. Difference between multiprocessing and multithreading

The threading module uses threads, the multiprocessing module uses processes. The difference is that threads run in the same memory space, while processes have separate memory. This makes it a bit harder to share objects between processes with multiprocessing. Since threads use the same memory, precautions have to be taken or two threads will write to the same memory at the same time.

- Multithreading is concurrent and is used for IO-bound tasks
- Multiprocessing achieves true parallelism and is used for CPU-bound tasks Use Multithreading if most of your task involves waiting on API-calls, because why not start up another request in another thread while you wait, rather than have your CPU sit idly by.

50. How to merge two dictionaries together?

```
first_dict = {'name': 'Tom', 'age': 44}
second_dict = {'occupation': 'actor', 'nationality': 'British'}
# merging
final_dict = {**first_dict, **second_dict}
```

In case any key is repeated in both dictionaries, the second key will hold supremacy.

PYTHON INTERVIW QUESTION

51. Explain the below code

```
def func(sample_list=[]):
    sample_list.append(12)
    # print(id(sample_list))
    return sample_list

print(func()) # [12]
print(func()) # [12,12]
```

Since list is mutualable type of data structure, the first time func is called, the list is empty, but when the same function is called twice, the list already has an item. We cans sure of the that by printing the id of the sample_list used in the first, on each subsquent call to the function, the id will return the same value.

52. Example filter with lambda expression.
filter

filter(function, iterable) # function must return True or False

```
input_list = ['Delhi', 'Mumbai', 'Noida', 'Gurugram']
to_match = 'Gurugram'

matched_list = list(filter(lambda item: item == to_match, input_list))
matched_list # ['Gurugram']
```

For every single item in the input_list, the condition is checked in the lambda function which returns either True or False.

PYTHON INTERVIW QUESTION

53. What is the maximum length of a Python identifier?

1. 32
2. 16
3. 128
4. No fixed length is specified.[ANSWER]

54. What will be the output of the following code snippet?

```
print(2**3 + (5 + 6)**(1 + 1))
```

1. 129 [ANSWER]
2. 8
3. 121
4. None of the above.

55. What will be the datatype of the var in the below code snippet?

```
var = 10
print(type(var))
var = "Hello"
print(type(var))
```

1. str and int
2. int and int
3. str and str
4. int and str [ANSWER]

56. How is a code block indicated in Python?

1. Brackets.
2. Indentation.[ANSWER]
3. Key.
4. None of the above.

PYTHON INTERVIW QUESTION

57. What will be the output of the following code snippet?

```
print(type(5 / 2))
print(type(5 // 2))
```

1. float and int [ANSWER]
2. int and float
3. float and float
4. int and int

58. What will be the output of the following code snippet?

```
a = [1, 2, 3, 4, 5]
sum = 0
for ele in a:
  sum += ele
print(sum)
```

1. 15 [ANSWER]
2. 0
3. 20
4. None of these

59. Which of the following types of loops are not supported in Python?

1. for
2. while
3. do-while
4. None of the above

PYTHON INTERVIW QUESTION

57. What will be the output of the following code snippet?

```
print(type(5 / 2))
print(type(5 // 2))
```

1. float and int [ANSWER]
2. int and float
3. float and float
4. int and int

58. What will be the output of the following code snippet?

```
a = [1, 2, 3, 4, 5]
sum = 0
for ele in a:
  sum += ele
print(sum)
```

1. 15 [ANSWER]
2. 0
3. 20
4. None of these

59. Which of the following types of loops are not supported in Python?

1. for
2. while
3. do-while [ANSWER]
4. None of the above

PYTHON INTERVIW QUESTION

60. What will be the output of the following code snippet?

```
a = [1, 2]
print(a * 3)
```

1. Error
2. [1, 2]
3. [1, 2, 1, 2]
4. [1, 2, 1, 2, 1, 2] [ANSWER]

61. What will be the output of the following code snippet?

```
example = ["Sunday", "Monday", "Tuesday", "Wednesday"];
del example[2]
print(example)
```

1. ['Sunday', 'Monday', 'Tuesday', 'Wednesday']
2. ['Sunday', 'Monday', 'Wednesday'] [ANSWER]
3. ['Monday', 'Tuesday', 'Wednesday']
4. ['Sunday', 'Monday', 'Tuesday']

62. Which of the following is the proper syntax to check if a particular element is present in a list?

1. if ele in list
2. if not ele not in list
3. Both A and B [ANSWER]
4. None of the above

PYTHON INTERVIW QUESTION

63. What will be the type of the variable sorted_numbers in the below code snippet?

```
numbers = (4, 7, 19, 2, 89, 45, 72, 22)
sorted_numbers = sorted(numbers)
print(sorted_numbers)
```

1. List [ANSWER]
2. Tuple
3. String
4. Int

64. What will be the output of the following code snippet?
def thrive(n):

```
def thrive(n):
 if n % 15 == 0:
   print("thrive", end = " ")
 elif n % 3 != 0 and n % 5 != 0:
   print("neither", end = " ")
 elif n % 3 == 0:
   print("three", end = " ")
 elif n % 5 == 0:
   print("five", end = " ")
thrive(35)
thrive(56)
thrive(15)
thrive(39)
```

1. five neither thrive three [ANSWER]
2. five neither three thrive
3. three three three three
4. five neither five neither

PYTHON INTERVIW QUESTION

65. What will be the output of the following code snippet?

```
numbers = (4, 7, 19, 2, 89, 45, 72, 22)
sorted_numbers = sorted(numbers)
even = lambda a: a % 2 == 0
even_numbers = filter(even, sorted_numbers)
print(type(even_numbers))
```

1. filter [ANSWER]
2. int
3. list
4. tuple

66. What will be the output of the following code snippet?

```
numbers = (4, 7, 19, 2, 89, 45, 72, 22)
sorted_numbers = sorted(numbers)
odd_numbers = [x for x in sorted_numbers if x % 2 != 0]
print(odd_numbers)
```

1. [7, 19, 45, 89] [ANSWER]
2. [2, 4, 22, 72]
3. [4, 7, 19, 2, 89, 45,72, 22]
4. [2, 4, 7, 19, 22, 45, 72, 89]

67. What will be the output of the following code snippet?

```
example = ["Sunday", "Monday", "Tuesday", "Wednesday"];
print(example[-3:-1])
```

1. ['Monday', 'Tuesday'] [ANSWER]
2. ['Sunday', 'Monday']
3. ['Tuesday', 'Wednesday']
4. ['Wednesday', 'Monday']

PYTHON INTERVIW QUESTION

68. What will be the output of the following code snippet?
def check(a):

```
def check(a):
   print("Even" if a % 2 == 0 else "Odd")

check(12)
```

1. Even [ANSWER]
2. Odd
3. Error
4. None

69. What will be the output of the following code snippet?

```
def is_even(number):
  message =  f"{number} is an even number" if number % 2 == 0 else  f"{number} is an odd number"
 return message
print(is_even(54))
```

1. 54 is an even number [ANSWER]
2. 54 is an odd number
3. number is an even number
4. number is an odd number

70. What will be the output of the following code snippet?

```
dict1 = {'first' : 'sunday', 'second' : 'monday'}
dict2 = {1: 3, 2: 4}
dict1.update(dict2)
print(dict1)
```

1. {'first': 'sunday', 'second': 'monday', 1: 3, 2: 4} [ANSWER]
2. {'first': 'sunday', 'second': 'monday'}
3. {1: 3, 2: 4}
4. None of the above.

PYTHON INTERVIW QUESTION

71. What will be the output of the following code snippet?

```
s = {1, 2, 3, 3, 2, 4, 5, 5}
print(s)
```

1. {1, 2, 3, 3, 2, 4, 5, 5}
2. {1, 2, 3, 4, 5} [ANSWER]
3. None
4. {1, 5}

72. What will be the output of the following code snippet?

```
a = {'Hello':'World', 'First': 1}
b = {val: k for k , val in a.items()}
print(b)
```

1. {'Hello':'World', 'First': 1}
2. {'World': 'Hello', 1: 'First'} [ANSWER]
3. Can be both A or B
4. None of the above

73. Which of the following functions converts date to corresponding time in Python?

1. strptime()
2. strftime()
3. Both A and B
4. None of the above

PYTHON INTERVIW QUESTION

74. What will be the output of the following code snippet?

```
word = "Python Programming"
n = len(word)
word1 = word.upper()
word2 = word.lower()
converted_word = ""
for i in range(n):
 if i % 2 == 0:
   converted_word += word2[i]
 else:
   converted_word += word1[i]
print(converted_word)
```

1. pYtHoN PrOgRaMmInG [ANSWER]
2. Python Programming
3. python programming
4. PYTHON PROGRAMMING

75. What will be the output of the following code snippet?

```
a = "4, 5"
nums = a.split(',')
x, y = nums
int_prod = int(x) * int(y)
print(int_prod)
```

1. 20 [ANSWER]
2. 45
3. 54
4. 4,5

PYTHON INTERVIW QUESTION

76. What will be the output of the following code snippet?

```
square = lambda x: x ** 2
a = []
for i in range(5):
   a.append(square(i))

print(a)
```

1. [0, 1, 4, 9, 16] [ANSWER]
2. [1, 4, 9, 16, 25]
3. [0, 1, 2, 3, 4]
4. [1, 2, 3, 4, 5]

77. What will be the output of the following code snippet?

```
def tester(*argv):
   for arg in argv:
       print(arg, end = ' ')
tester('Sunday', 'Monday', 'Tuesday', 'Wednesday')
```

1. Sunday
2. Wednesday
3. Sunday Monday Tuesday Wednesday [ANSWER]
4. None of the above.

78. As what datatype are the *args stored, when passed into a function?

1. List.
2. Tuple.
3. Dictionary.
4. None of the above.

PYTHON INTERVIW QUESTION

79. What will be the output of the following code snippet?

```
def tester(**kwargs):
   for key, value in kwargs.items():
       print(key, value, end = " ")
tester(Sunday = 1, Monday = 2, Tuesday = 3, Wednesday = 4)
```

1. Sunday 1 Monday 2 Tuesday 3 Wednesday 4 [ANSWER]
2. Sunday 1
3. Wednesday 4
4. None of the above

80. As what datatype are the *kwargs stored, when passed into a function?

1. Lists.
2. Tuples.
3. Dictionary. [ANSWER]
4. None of the above.

81. Which of the following blocks will always be executed whether an exception is encountered or not in a program?

1. try
2. except
3. finally [ANSWER]
4. None of These

PYTHON INTERVIW QUESTION

82. What will be the output of the following code snippet?

```
from math import *
a = 2.19
b = 3.999999
c = -3.30
print(int(a), floor(b), ceil(c), fabs(c))
```

1. 2 3 -3 3.3 [ANSWER]
2. 3 4 -3 3
3. 2 3 -3 3
4. 2 3 -3 -3.3

83. What will be the output of the following code snippet?

```
set1 = {1, 3, 5}
set2 = {2, 4, 6}
print(len(set1 + set2))
```

1. 3
2. 6
3. 0
4. Error [ANSWER]

84. What keyword is used in Python to raise exceptions?

1. raise [ANSWER]
2. try
3. goto
4. except

85. Which of the following is not a valid set operation in python?

1. Union
2. Intersection
3. Difference
4. None of the above [ANSWER]

PYTHON INTERVIW QUESTION

86. What will be the output of the following code snippet?

```
s1 = {1, 2, 3, 4, 5}
s2 = {2, 4, 6}
print(s1 ^ s2)
```

1. {1, 2, 3, 4, 5}
2. {1, 3, 5, 6} [ANSWER]
3. {2, 4}
4. None of the above

87. What will be the output of the following code snippet?

```
a = [1, 2, 3, 4]
b = [3, 4, 5, 6]
c = [x for x in a if x not in b]
print(c)
```

1. [ ]
2. [5, 6]
3. [1, 2, 5, 6]
4. [3, 4]

88. Which of the following are valid escape sequences in Python?

1. \n
2. \t
3. \\
4. All of the above [ANSWER]

89. Which of the following are valid string manipulation functions in Python?

1. count()
2. upper()
3. strip()
4. All of the above [ANSWER]

PYTHON INTERVIW QUESTION

90. Which of the following modules need to be imported to handle date time computations in Python?

1. datetime [ANSWER]
2. date
3. time
4. timedate

91. How can assertions be disabled in Python?

1. Passing -O when running Python. [ANSWER]
2. Assertions are disabled by default.
3. Both A and B are wrong.
4. Assertions cannot be disabled in Python.

92. What will be the output of the following code snippet?

```
a = [[], "abc", [0], 1, 0]
print(list(filter(bool, a)))
```

1. ['abc', [0], 1] [ANSWER]
2. [1]
3. [“abc”]
4. None of the above

93. In which language is Python written?

1. C++
2. C [ANSWER]
3. Java
4. None of these

PYTHON INTERVIW QUESTION

94. What will be the result of the following expression in Python "2 ** 3 + 5 ** 2"?

1. 65536
2. 33 [ANSWER]
3. 169
4. None of these

95. What will be the output of the following code snippet?

```
count = 0
while(True):
   if count % 3 == 0:
       print(count, end = " ")
   if(count > 15):
       break;
   count += 1
```

1. 0 1 2 15
2. Infinite Loop
3. 0 3 6 9 12 15 [ANSWER]
4. 0 3 6 9 12

96. Which of the following concepts is not a part of Python?

1. Pointers. [ANSWER]
2. Loops.
3. Dynamic Typing.
4. All of the above.

97. Which of the following statements are used in Exception Handling in Python?

1. try
2. except
3. finally
4. All of the above [ANSWER]

PYTHON INTERVIW QUESTION

98. What will be the output of the following code snippet?
def solve(a, b):

```
def solve(a, b):
   return b if a == 0 else solve(b % a, a)
print(solve(20, 50))
```

1. 10 [ANSWER]
2. 20
3. 50
4. 1

99. What will be the output of the following code snippet?

```
def func():
   global value
   value = "Local"

value = "Global"
func()
print(value)
```

1. Local [ANSWER]
2. Global
3. None
4. Cannot be predicted

PYTHON INTERVIW QUESTION

100. What will be the output of the following code snippet?

```
def solve(a):
    a = [1, 3, 5]
a = [2, 4, 6]
print(a)
solve(a)
print(a)
```

1. [2, 4, 6]. [2, 4, 6] [ANSWER]
2. [2, 4, 6], [1, 3, 5]
3. [1. 3. 5], [1, 3, 5]
4. None of these.