Interview Questions

**Python**

1. **What is init keyword ?**

Is a special method used for initializing objects of a class. It's also known as the constructor method.

1. **What is self keyword ?**

self is not a keyword but a convention in Python. It's used as the first parameter in instance methods of a class to refer to the instance itself. It's similar to this in some other programming languages. For example:

class MyClass:

def \_\_init\_\_(self, value):

self.value = value

def print\_value(self):

print(self.value)

obj = MyClass(42)

obj.print\_value()

1. **What is lambda functon?**

A lambda function, also known as an anonymous function or lambda expression, is a concise way to create small, unnamed functions in Python. It's defined using the lambda keyword and can take multiple arguments but can only have one expression. Lambda functions are often used for short, simple operations.

add = lambda x, y: x + y

print(add(2, 3)) # Outputs 5

1. **Difference between lambda and normal function?**

**Lambda** Function: Anonymous, defined using lambda keyword, limited to a single expression.

**Normal** Function: Defined using def, has a name, can contain multiple expressions and statements.

1. **What are generators?**

Generators are a type of iterable in Python that allow you to iterate over a potentially large sequence of items without loading them all into memory at once. They are created using functions with the **yield keyword instead of return**. Generators are memory-efficient and used for lazy evaluation.

def count\_up\_to(limit):

count = 1

while count <= limit:

yield count

count += 1

# Create a generator object

counter = count\_up\_to(3)

# Iterate through the generator

for number in counter:

print(number)

#Output

1

2

3

**Here's what happens step-by-step:**

* The count\_up\_to generator is defined to count from 1 to limit.
* A generator object counter is created by calling the generator without executing it (no parentheses).
* The for loop iterates through counter. During each iteration, count\_up\_to runs until it encounters yield, then returns the count and pauses.
* On the next iteration, the function resumes where it left off, keeping all previous states (like the value of count) intact. This continues until the while condition is False.

1. **Python is compiled or interpreted language ? what does it mean?**

Python is often referred to as an interpreted language, but it's actually both compiled and interpreted. Code is first compiled into bytecode and then interpreted by the Python interpreter.

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

List: Mutable, defined using square brackets [].

Tuple: Immutable, defined using parentheses ().

1. **What is the difference between list and set in Python?**

List: Ordered, allows duplicates, defined with [].

Set: Unordered, doesn't allow duplicates, defined with {}.

1. **When to use dictionary?**

A dictionary is an unordered collection of data stored as key-value pairs. Dictionaries are mutable, meaning they can be modified, and are known for their ability to quickly retrieve, insert, and delete items based on the key.

**Basic Syntax:**

my\_dict = {

'key1': 'value1',

'key2': 'value2',

'key3': 'value3',

# ...

}

**Example:**

person = {

'name': 'Alice',

'age': 30,

'gender': 'female'

}

# Access value

print(person['name']) # Output: Alice

# Update value

person['age'] = 31

print(person['age'])

# Add key-value pair

person['city'] = 'New York'

print(person['city'])

# Delete key-value pair

del person['gender']

print(person)

**Use dictionaries when** you have a set of unique keys that map to values and you need efficient insertion, deletion, and lookup operations.

**Example:**

text = "apple banana apple orange banana apple"

words = text.split()

# Empty dictionary to store word:frequency pairs

word\_count = {}

# Iterating through words to populate dictionary

for word in words:

if word in word\_count:

word\_count[word] += 1 # Increment count if word is already in dictionary

else:

word\_count[word] = 1 # Add word to dictionary if not present

# Output: {'apple': 3, 'banana': 2, 'orange': 1}

print(word\_count)

1. **What are decorators?**

Decorators is a design pattern that allows you to modify or enhance the behaviour of functions or methods without changing their code. They are often used to add functionality to functions, such as logging, authentication, or memorization.

def welcome\_decorator(func):

def wrapper():

return "Welcome " + func()

return wrapper

@welcome\_decorator

def get\_name():

return "Alice"

print(get\_name()) # Output: Welcome Alice

**Explanation**:

* Define a Decorator: welcome\_decorator is a function that takes another function func as parameter, defines a function wrapper inside it, and returns that wrapper function. The wrapper function calls the input func and modifies its output.
* Apply the Decorator: @welcome\_decorator above get\_name function is Python's decorator syntax. It's equivalent to get\_name = welcome\_decorator(get\_name). The get\_name function is passed to welcome\_decorator as func, and gets replaced with wrapper function.
* Call the Function: print(get\_name()) now calls wrapper, which calls get\_name and adds "Welcome " to its output.

1. **What are Iterators?**

Iterators are objects in Python that represent a sequence of items. They implement the \_\_iter\_\_() and \_\_next\_\_() methods, allowing you to iterate over the items one at a time. Common examples of iterators include lists, tuples, and dictionaries.

1. **What is slicing?**

Slicing is a technique in Python used to extract a portion of a sequence (e.g., a list or string) by specifying a start and end index. It is done using square brackets and the : operator. For example, my\_list[1:4] would return a sublist containing elements from index 1 to 3.

**Example 1: Slicing a List**

my\_list = [0, 1, 2, 3, 4, 5]

# Extract from index 1 to index 4

slice1 = my\_list[1:5]

# Output: [1, 2, 3, 4]

print(slice1)

**Example 2: Slicing with Step**

# Extract every second element between index 1 and index 6

slice2 = my\_list[1:7:2]

# Output: [1, 3, 5]

print(slice2)

**Example 3: Slicing a String**

my\_string = "Hello, World!"

# Extract substring from index 7 to the end

slice3 = my\_string[7:]

# Output: World!

print(slice3)

**Example 4: Negative Index Slicing**

# Extract last 3 elements

slice4 = my\_list[-3:]

# Output: [3, 4, 5]

print(slice4)

**Example 5: Slicing with No Indices**

# Extract all elements with step 2 (every second element)

slice5 = my\_list[::2]

# Output: [0, 2, 4]

print(slice5)

* If start is omitted, it defaults to the first element (index 0).
* If stop is omitted, it defaults to the last element (inclusive).
* If step is omitted, it defaults to 1, meaning extract every element between start and stop.
* Negative indices count from the end of the sequence.

1. **What is mutable and immutable?**

**Mutable**: Objects that can be changed after creation (e.g., list, dict, set).

**Immutable**: Objects that cannot be changed after creation (e.g., tuple, string).

1. **Python is single thread or multithread?**

Python has a Global Interpreter Lock (GIL), which allows only one thread to execute Python code at a time in a multi-threaded program. This means that Python threads are not suitable for CPU-bound tasks but can be used for I/O-bound tasks.

1. **What is GIL**

The Global Interpreter Lock (GIL) is a mutex that protects access to Python objects, preventing multiple native threads from executing Python bytecodes at once.

1. **What you don’t like about python?**

X

1. **What is list Comprehension?**

List comprehension is a concise way to create lists in Python. It allows you to generate a new list by applying an expression to each item in an existing iterable (e.g., a list) and optionally applying a condition

1. **What are Dunder methods? Give examples**

Dunder methods, short for "double underscore" methods, are special methods in Python that have double underscores at both the beginning and end of their names.

class MyClass:

def \_\_init\_\_(self, value):

self.value = value

obj = MyClass(42) # Calls \_\_init\_\_ and initializes obj.value to 42

1. **What does \_init\_ method do?**

The \_\_init\_\_ method in Python is a special method, also known as a dunder method due to its double underscores. It's also referred to as a magic method or an initializer method. It is automatically called when an object of a class is instantiated, and it allows developers to set initial values to attributes or perform any setup required for the object. The \_\_init\_\_ method is defined using the def keyword, followed by \_\_init\_\_.

**Basic syntax:**

class ClassName:

def \_\_init\_\_(self, parameter1, parameter2, ...):

# initialization code goes here

**Example**

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

def display\_info(self):

print(f"{self.year} {self.make} {self.model}")

# Creating an object of the Car class

my\_car = Car(“Toyota", "Corolla", 2022)

# Using object's method to display information

my\_car.display\_info() # Output: 2022 Toyota Corolla

**20. What is def?**

def is a keyword used to declare or define a function. The keyword def is followed by the function name, parentheses (), and a colon :

**Basic Syntax:**

def function\_name(parameters):

# Function Body

Pass

**Example:**

def greet(name):

return f"Hello, {name}!"

# Using the function

print(greet("World"))