Deloitte Python Interview Questions

Basic Python Questions

1. What is Python?

Python is a high-level, interpreted programming language known for its simplicity. It supports multiple programming paradigms and is widely used in fields like web development, data science, artificial intelligence, and more.

2. What are the key features of Python?

- **Simple and Readable: Python's syntax is easy to learn and use.**
- **Interpreted: Python is executed line by line, making it easy to debug.**
- **Dynamic Typing: No need to declare variable types.**
- **Extensive Libraries: Python has libraries for AI, data science, and more.**
- **Cross-Platform: Python code runs on multiple operating systems.**

3. What are Python's built-in data types?

Python has several built-in data types like integers, floating-point numbers, strings, lists, tuples, dictionaries, and sets. Each of these is designed for different purposes, like storing numbers, collections of data, or key-value pairs.

4. What is the difference between a list and a tuple in Python?

- **List:**
  - **Mutable (can be modified).**
  - **Defined using square brackets: [1, 2, 3].**
  - **Suitable for dynamic data storage.**
- **Tuple:**
  - **Immutable (cannot be modified).**
  - **Defined using parentheses: (1, 2, 3).**
  - **Suitable for fixed data storage.**

5. What is the difference between Python 2 and Python 3?

Python 2 is an older version that is no longer supported, while Python 3 is the modern version. Python 3 introduced many improvements, including better Unicode handling and changes in syntax like the print() function. New projects should always use Python 3.

6. How is memory managed in Python?

- **Private Heap: All Python objects are stored in a private heap.**
- **Reference Counting: Python uses reference counts to track objects.**
- **Garbage Collection: Cyclic garbage collector to clean up unused objects.**
- **Dynamic Memory Allocation: Handled by Python's memory manager.**

7. What is PEP 8?

PEP 8 is the style guide for writing Python code. It provides guidelines on how to write clean, readable code, covering naming conventions, indentation, and other coding best practices.

8. What are functions in Python?

- **Definition: A block of reusable code that performs a specific task.**
- **Syntax: Defined using the def keyword.**
- **Arguments: Functions can take input parameters (arguments).**
- **Return: They may return a value using the return statement.**

Example: Python code

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

9. What is the difference between local and global variables?

Local variables are declared within a function and can only be accessed inside that function. Global variables are declared outside all functions and can be accessed anywhere in the program. Global variables can be accessed inside functions, but local variables cannot be accessed outside the function they are declared in.

10. What are Python literals?

- **String Literal: 'Hello' or "World".**
- **Numeric Literal: 10, 3.14.**
- **Boolean Literal: True, False.**
- **Special Literal: None, used to represent no value.**

Intermediate Python Questions

11. Explain list comprehensions with an example.

List comprehensions provide a concise way to create lists. Instead of using loops, you can generate a list in a single line by applying an expression to each item of an existing list.

Example:

```
squares = [x**2 for x in range(5)] # Output: [0, 1, 4, 9, 16]
```

12. What are Python decorators?

- **Definition: A function that modifies another function.**
- **Use Case: Used to add functionality to existing functions.**
- **Syntax: Uses the @decorator_function syntax.**

Example:

```
@decorator
def my_function():
pass
```

13. What is the difference between shallow copy and deep copy?

- **Shallow Copy: Creates a new object but copies references to nested objects. Changes in nested objects will affect both the original and the copy.**
- **Deep Copy: Recursively copies all objects, creating an entirely new object. Changes in nested objects in the original do not affect the copy.**

14. What are the differences between append() and extend() in lists?

- **append(): Adds a single element to the end of the list.**

```
my_list = [1, 2]
my_list.append(3) # Output: [1, 2, 3]
```

- **extend(): Adds multiple elements from another iterable to the list.**

```
my_list.extend([4, 5]) # Output: [1, 2, 3, 4, 5]
```

15. Explain how Python handles exceptions.

Python uses the try-except block to handle exceptions. Code that might raise an error is placed in the try block, and the except block catches and handles the error.

Example:

```
try:
result = 10 / 0
except ZeroDivisionError:
print("Cannot divide by zero!")
```

16. What is a lambda function in Python?

- **Anonymous Function: A function without a name.**
- **Syntax: Defined using the lambda keyword.**
- **Use Case: Used for short, simple functions.**

Example:

```
add = lambda x, y: x + y
print(add(2, 3)) # Output: 5
```

17. What are the key differences between lists and dictionaries?

- **Lists: Ordered collections that store elements by index.**
- **Dictionaries: Unordered collections that store key-value pairs. Lists allow duplicate elements, while dictionary keys must be unique.**

18. What are *args and kwargs in Python?

- **args: Allows you to pass a variable number of positional arguments to a function.**

```
def add(*args):
return sum(args)
```

- **kwargs: Allows you to pass a variable number of keyword arguments (key-value pairs) to a function.**

```
def print_values(**kwargs):
for key, value in kwargs.items():
print(f"{key}: {value}")
```

19. How can you sort a dictionary by value?

You can sort a dictionary by its values using the sorted( ) function and the items( ) method.

Example:

```
my_dict = {'a': 2, 'b': 1, 'c': 3}
sorted_dict = dict(sorted(my_dict.items(), key=lambda item: item[1]))
# Output: {'b': 1, 'a': 2, 'c': 3}
```

20. Explain the difference between is and == in Python.

- **is: Checks whether two variables point to the same object in memory.**

```
a = [1, 2]
b = a
print(a is b) # Output: True
```

- **==: Checks whether two variables have the same value.**

```
c = [1, 2]
print(a == c) # Output: True
```

Advanced Python Questions

21. What are generators in Python?

Generators are functions that return an iterator and allow you to iterate over large datasets efficiently. Instead of returning all items at once, they yield items one at a time, making them memory-efficient.

Example:

```
def count_up_to(max):
count = 1
while count <= max:
yield count
count += 1
```

22. Explain how Python handles memory management.

- **Reference Counting: Each object has a reference count, which tracks the number of references to it.**
- **Garbage Collection: Python uses a cyclic garbage collector to clean up unused objects that are involved in circular references.**
- **Private Heap: All objects are stored in Python's private heap.**

23. What are metaclasses in Python?

A metaclass is the class of a class in Python. It defines how classes behave. Metaclasses allow you to control the creation of classes and modify their behavior. By using a metaclass, you can add extra functionality to classes or modify class properties.

24. What is the Global Interpreter Lock (GIL) in Python?

- **Definition: The GIL is a mutex that protects access to Python objects and ensures that only one thread executes Python bytecode at a time.**
- **Impact: It limits the performance of multi-threaded Python programs, especially on multi-core systems.**
- **Workaround: Use multi-processing instead of multi-threading for CPU-bound tasks.**

25. How do you handle file operations in Python?

Python provides built-in functions to handle file operations such as reading, writing, and appending. The most common functions are open( ), read( ), write( ), and close( ).

Example:

```
with open('file.txt', 'r') as file:
content = file.read()
```


26. What is monkey patching in Python?

- **Definition: Dynamically modifying or extending a class or module at runtime.**
- **Use Case: Used in testing or when you need to alter behavior without changing the original code.**

Example:

```
class A:
def display(self):
print("Original")

def monkey_patched_display():
print("Monkey Patched")

A.display = monkey_patched_display
a = A()
a.display() # Output: Monkey Patched
```

27. How do you implement inheritance in Python?

Inheritance allows one class (child) to inherit attributes and methods from another class (parent). Python supports single inheritance, multiple inheritance, and multilevel inheritance.

Example:

```
class Animal:
def speak(self):
return "Animal Sound"

class Dog(Animal):
def speak(self):
return "Bark"
```

28. What is the difference between a class method and a static method?

- **Class Method:**
  - **Uses the @classmethod decorator.**
  - **Takes cls as the first argument, referring to the class.**
  - **Can modify class state that applies to all instances.**
- **Static Method:**
  - **Uses the @staticmethod decorator.**
  - **Does not take self or cls as an argument.**
  - **Cannot modify class or instance state.**

29. How can you implement multithreading in Python?

Multithreading in Python is handled using the threading module. You can create and manage threads to perform multiple tasks concurrently. However, due to the Global Interpreter Lock (GIL), threads in Python do not run in parallel on multi-core systems for CPU-bound tasks.

Example:

```
import threading

def print_numbers():
for i in range(5):
print(i)

thread = threading.Thread(target=print_numbers)
thread.start()
```

30. How do you use the map() function in Python?

- **Definition: The map( ) function applies a function to all items in an input list.**

- **Syntax: map(function, iterable).**
- **Use Case: Useful for applying transformations to lists or other iterables.**

Example:

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x**2, numbers)) # Output: [1, 4, 9, 16]
```

Object-Oriented Programming (OOP) in Python

31. What is Object-Oriented Programming (OOP) in Python?

Object-Oriented Programming (OOP) is a paradigm that organizes code into objects, which contain data (attributes) and functions (methods). Python supports OOP, allowing you to create reusable and modular code.

32. What is the difference between a class and an object?

- **Class:**
  - **A blueprint for creating objects.**
  - **Defines properties and behavior.**
  - **Example: class Dog: pass**
- **Object:**
  - **An instance of a class.**
  - **Example: my_dog = Dog()**

33. What is polymorphism in Python?

Polymorphism allows the same function or method to work in different ways depending on the object. For example, different classes can have methods with the same name, but the behavior will be specific to each class.

Example:

```
class Dog:
def sound(self):
return "Bark"

class Cat:
def sound(self):
return "Meow"

def animal_sound(animal):
print(animal.sound())
```

```
dog = Dog()
cat = Cat()

animal_sound(dog) # Output: Bark
animal_sound(cat) # Output: Meow
```

34. What is inheritance in Python?

- **Definition: A class can inherit properties and methods from another class.**
- **Parent Class: The class being inherited from.**
- **Child Class: The class that inherits from the parent.**

Example:

```
class Animal:
def eat(self):
print("Eating")

class Dog(Animal):
def bark(self):
print("Barking")
```

Error Handling

35. What is exception handling in Python?

Exception handling in Python is done using the try, except, finally, and else blocks. This allows you to manage errors gracefully, preventing the program from crashing when an error occurs.

Example:

```
try:
result = 10 / 0
except ZeroDivisionError:
print("Cannot divide by zero!")
finally:
print("Execution complete.")
```

36. What is the difference between syntax errors and exceptions?

- **Syntax Error:**
  - **Occurs when the code has incorrect syntax.**
  - **Detected by the interpreter before execution.**
  - **Example: if a == 3 print("Hello") (missing :).**

- **Exception:**
  - **Occurs during program execution.**
  - **Detected when the program encounters an unexpected condition.**
  - **Example: Division by zero error.**

File Handling

37. How do you open and read a file in Python?

You can open and read a file using the open( ) function in Python. Files can be opened in different modes like read ('r'), write ('w'), append ('a'), and more.

Example:

```
with open('file.txt', 'r') as file:
content = file.read()
print(content)
```

38. What is the difference between read(), readline(), and readlines()?

- **read():**
  - **Reads the entire content of the file as a single string.**
- **readline():**
  - **Reads a single line from the file.**
- **readlines():**
  - **Reads all lines and returns them as a list of strings.**

Data Structures

39. What is a dictionary in Python?

A dictionary is a built-in data structure that stores data in key-value pairs. Dictionaries are mutable and allow for fast retrieval of values using keys.

Example:

```
my_dict = {'name': 'John', 'age': 30}
print(my_dict['name']) # Output: John
```

40. How do you merge two dictionaries in Python?

- **Using the update( ) method:**

```
dict1 = {'a': 1}
dict2 = {'b': 2}
dict1.update(dict2) # Output: {'a': 1, 'b': 2}
```

- **Using the * * operator:**

```
merged_dict = {**dict1, **dict2}
```

41. What is a set in Python, and how does it differ from a list?

Set:

- **An unordered collection of unique elements.**
- **Defined using curly braces: {1, 2, 3}.**
- **No duplicate values.**

List:

- **An ordered collection of elements.**
- **Defined using square brackets: [1, 2, 3].**
- **Allows duplicate values.**

Data Manipulation

42. How do you remove duplicates from a list in Python?

You can remove duplicates from a list by converting it to a set, which automatically removes duplicates. You can then convert it back to a list if needed.

Example:

```
my_list = [1, 2, 2, 3, 3]
unique_list = list(set(my_list))
print(unique_list) # Output: [1, 2, 3]
```

43. What are the different ways to concatenate strings in Python?

- **Using the '+' operator:**

```
s1 = "Hello"
s2 = "World"
result = s1 + " " + s2 # Output: "Hello World"
```

- **Using the join( ) method:**

```
result = " ".join([s1, s2]) # Output: "Hello World"
```

Regular Expressions

44. What are regular expressions in Python?

Regular expressions (regex) are sequences of characters that define search patterns. In Python, you can use the 're' module to work with regular expressions for pattern matching and text manipulation.

Example:

```
import re
pattern = r'\d+' # Matches one or more digits
result = re.findall(pattern, 'There are 3 cats and 4 dogs.')
print(result) # Output: ['3', '4']
```

45. What is the difference between re.search() and re.findall( )?

- **re.search():**
  - **Returns the first match found in the string.**
- **re.findall():**
  - **Returns all matches found in the string as a list.**

Iterators and Generators

46. What are iterators in Python?

An iterator is an object that allows you to iterate over a collection of elements one at a time. In Python, an object is an iterator if it implements the __iter__( ) and __next__( ) methods.

Example:

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

iterator = iter(my_list)

print(next(iterator)) # Output: 1
```

47. How is a generator different from an iterator?

- **Generator:**
  - **A type of iterator that yields items lazily (one at a time).**
  - **Defined using the yield keyword.**
- **Iterator:**
  - **A general object used for iterating over collections.**

- Requires implementing __iter__( ) and __next__( ) methods.

Concurrency and Parallelism

48. How can you achieve parallelism in Python?

Parallelism in Python can be achieved using the multiprocessing module, which allows you to run multiple processes in parallel, taking advantage of multiple CPU cores.

Example:

```
from multiprocessing import Pool

def square(n):
return n * n

with Pool(5) as p:
result = p.map(square, [1, 2, 3, 4, 5])
print(result) # Output: [1, 4, 9, 16, 25]
```

49. What is the difference between threading and multiprocessing?

- **Threading:**
  - **Allows multiple threads within a single process.**
  - **Limited by the Global Interpreter Lock (GIL).**
  - **Suitable for I/O-bound tasks.**
- **Multiprocessing:**
  - **Runs separate processes, each with its own Python interpreter.**
  - **Can achieve true parallelism.**
  - **Suitable for CPU-bound tasks.**

Data Science Libraries

50. What is Pandas, and how is it used in Python?

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures like DataFrame and Series, which are designed for handling structured data efficiently.

Example:

```
import pandas as pd
data = {'name': ['John', 'Anna'], 'age': [30, 25]}
```

```
df = pd.DataFrame(data)
print(df)
```