PYTHON INTERVIEW QUESTIONS (THEORY)

1. What is Python? List some popular applications of Python in the world of technology.
Python is a widely-used general-purpose, high-level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.
It is used for:

- IOT
- Web Development
- Game Development
- Software Development
- Data analysis

2. What are the benefits of using Python language as a tool in the present scenario?
The following are the benefits of using Python language:

- Object-Oriented Language
- High-Level Language
- Dynamically Typed language
- Extensive support Libraries
- Presence of third-party modules
- Open source and community development
- Portable and Interactive
- Portable across Operating systems

3. Is Python a compiled language or an interpreted language?
Actually, Python is a partially compiled language and partially interpreted language. The compilation part is done first when we execute our code and this will generate byte code internally this byte code gets converted by the Python virtual machine(p.v.m) according to the underlying platform(machine+operating system).

4. What does the '#' symbol do in Python?
'#' is used to comment on everything that comes after on the line.

5. What is the difference between a Mutable datatype and an Immutable data type?
Mutable data types can be edited i.e., they can change at runtime. Eg – List, Dictionary, etc. Immutable data types can not be edited i.e., they can not change at runtime. Eg – String, Tuple, etc.

6. How are arguments passed by value or by reference in Python?
Everything in Python is an object and all variables hold references to the objects. The reference values are according to the functions; as a result, you cannot change the value of the references. However, you can change the objects if it is mutable.

7. What is the difference between a Set and Dictionary?
The set is an unordered collection of data types that is iterable, mutable and has no duplicate elements. A dictionary in Python is an ordered collection of data values, used to store data values like a map.

8. What is List Comprehension? Give an Example.
List comprehension is a syntax construction to ease the creation of a list based on existing iterable.

9. What is a lambda function?

A lambda function is an anonymous function. This function can have any number of parameters but, can have just one statement.

10. What is a pass in Python?

Pass means performing no operation or in other words, it is a placeholder in the compound statement, where there should be a blank left and nothing has to be written there.

11. What is the difference between / and // in Python?

/ represents precise division (result is a floating point number) whereas // represents floor division (result is an integer).

12. How is Exceptional handling done in Python?

There are 3 main keywords i.e. try, except, and finally which are used to catch exceptions and handle the recovering mechanism accordingly. Try is the block of a code that is monitored for errors. Except block gets executed when an error occurs.

The beauty of the final block is to execute the code after trying for an error. This block gets executed irrespective of whether an error occurred or not. Finally, block is used to do the required cleanup activities of objects/variables.

13. What is swapcase function in Python?

It is a string's function that converts all uppercase characters into lowercase and vice versa. It is used to alter the existing case of the string. This method creates a copy of the string which contains all the characters in the swap case.

14. Difference between for loop and while loop in Python

The "for" Loop is generally used to iterate through the elements of various collection types such as List, Tuple, Set, and Dictionary. Developers use a "for" loop where they have both the conditions start and the end. Whereas, the "while" loop is the actual looping feature that is used in any other programming language. Programmers use a Python while loop where they just have the end conditions.

15. Can we Pass a function as an argument in Python?

Yes, Several arguments can be passed to a function, including objects, variables (of the same or distinct data types), and functions. Functions can be passed as parameters to other functions because they are objects. Higher-order functions are functions that can take other functions as arguments.

16. What are *args and *kwargs?

***args**

- `*args` is used to pass a variable number of non-keyword arguments to a function.
- It allows you to pass a variable number of arguments to a function, which are then accessible as a tuple within the function.

****kwargs**

- `**kwargs` allows you to pass a variable number of keyword arguments to a function.
- It allows you to handle named arguments that you have not defined in advance. These arguments are accessible as a dictionary within the function.

17. Is Indentation Required in Python?

Yes, indentation is required in Python. A Python interpreter can be informed that a group of statements belongs to a specific block of code by using Python indentation. Indentations make the code easy to read for developers in all programming languages but in Python, it is very important to indent the code in a specific order.

18. What is a list in Python?

A list is an ordered collection of elements which can be of different types (integers, floats, strings, etc.). Lists are mutable, meaning their elements can be changed after the list is created.

19. How are lists implemented internally in Python?

Lists are implemented as dynamic arrays in Python. This means they use contiguous memory locations to store their elements, which allows for efficient indexing and iteration.

20. What are some common use cases for lists?

Lists are commonly used to store collections of related items, such as a list of names, numbers, or objects. They are also used in algorithms that require a collection of items that can be dynamically modified.

21. What is the time complexity for adding or removing elements from a list?

- Appending an element to the end of a list has an average time complexity of O(1).
- Inserting or removing an element at the beginning or middle of a list has a time complexity of O(n), where n is the number of elements in the list.

22. What is the difference between shallow copy and deep copy of a list?

- A shallow copy of a list creates a new list with references to the same elements as the original list. Changes to the elements in the copied list will affect the original list if the elements are mutable.
- A deep copy creates a new list and recursively copies all elements, including nested lists. Changes to the elements in the copied list will not affect the original list.

23. What are list comprehensions and how are they used?

List comprehensions provide a concise way to create lists. They consist of brackets containing an expression followed by a for clause, and optionally, one or more if clauses.

24. What are some methods available for lists in Python?

Some common methods are:

- append(): Adds an element to the end of the list.
- extend(): Adds all elements of an iterable to the end of the list.
- insert(): Inserts an element at a specified position.
- remove(): Removes the first occurrence of a specified value.
- pop(): Removes and returns the element at a specified position.

- sort(): Sorts the list in place.
- reverse(): Reverses the elements of the list in place.
- index(): Returns the index of the first occurrence of a specified value.
- count(): Returns the number of occurrences of a specified value.

25.What is a string in Python?

A string is an immutable sequence of characters enclosed within single quotes (' ') or double quotes (" "). Strings are used to store and manipulate text data.

26. How are strings implemented internally in Python?

Strings in Python are implemented as arrays of bytes representing Unicode characters. Each character in a string is stored as a sequence of one or more bytes.

27. What are some common string operations?

Common string operations include concatenation, slicing, searching for substrings, replacing substrings, and splitting strings into lists based on delimiters.

28. Why are strings immutable in Python?

Strings are immutable to ensure that they can be safely shared between different parts of a program without unintended side effects. This immutability also allows for various optimizations, such as interning, which can improve performance and memory usage.

29. What is string interpolation and how can it be done in Python?

String interpolation is the process of embedding values into a string.

30. What are some common methods available for strings in Python?

Some common methods are:

- `split()`: Splits a string into a list of substrings based on a delimiter.
- `join()`: Joins a list of strings into a single string with a specified delimiter.
- `strip()`: Removes leading and trailing whitespace.
- `replace()`: Replaces occurrences of a substring with another substring.
- `find()`: Returns the lowest index of a substring.
- `upper()`: Converts all characters to uppercase.
- `lower()`: Converts all characters to lowercase.
- `startswith()`: Checks if the string starts with a specified prefix.
- `endswith()`: Checks if the string ends with a specified suffix.

31. What is a tuple in Python?

A tuple is an ordered, immutable collection of elements which can be of different types. Tuples are similar to lists, but once created, their elements cannot be modified.

32. How are tuples implemented internally in Python?

Tuples are implemented as immutable arrays. Like lists, they store their elements in contiguous memory locations, but unlike lists, they cannot be resized.

33. What are some common use cases for tuples?
Tuples are often used to represent fixed collections of items, such as coordinates (x, y), RGB color values (R, G, B), or records where the number of elements is fixed and known in advance.

34. What are the advantages of using tuples over lists?

- Tuples are more memory-efficient than lists.
- Tuples can be used as keys in dictionaries because they are immutable and hashable, whereas lists cannot.
- Tuples provide a clear indication that the data they hold is constant and should not be modified.

35. Can you nest tuples?

Yes, tuples can be nested within each other, just like lists.
Example: nested_tuple = ((1, 2), (3, 4), (5, 6))

36. What are some common operations and methods available for tuples?
Common operations and methods include:

- Indexing and slicing to access elements.
- Using len() to get the length of a tuple.
- Concatenation using the + operator.
- Repeating a tuple using the * operator.
- Methods like count() and index() to count occurrences and find the index of an element.

37. What is a dictionary in Python?

A dictionary is an unordered collection of key-value pairs, where each key is unique. Dictionaries are mutable, meaning that they can be changed after their creation.

38. How are dictionaries implemented internally in Python?

Dictionaries are implemented using hash tables. Keys are hashed to calculate an index, and this index is used to store the corresponding value. This allows for fast access, insertion, and deletion of key-value pairs.

39. What are some common use cases for dictionaries?

Dictionaries are commonly used for fast lookups of data, such as storing configuration settings, counting occurrences of items, and representing mappings between related data.

40. What are the time complexities for common dictionary operations?
Accessing, inserting, and deleting elements in a dictionary have an average time complexity of O(1), thanks to the underlying hash table implementation.

41. Can dictionary keys be of any type?

No, dictionary keys must be of a type that is immutable and hashable. Common hashable types include strings, numbers, and tuples containing only hashable types.

42. What is the difference between `dict.items()`, `dict.keys()`, and `dict.values()`?

- `dict.items()`: Returns a view object that displays a list of dictionary's key-value tuple pairs.
- `dict.keys()`: Returns a view object that displays a list of all the keys.
- `dict.values()`: Returns a view object that displays a list of all the values.

43. What are some common methods available for dictionaries?

Some common methods are:

- `get()`: Returns the value for a specified key if the key is in the dictionary.
- `update()`: Updates the dictionary with elements from another dictionary or from an iterable of key-value pairs.
- `pop()`: Removes and returns the value for a specified key.
- `popitem()`: Removes and returns the last inserted key-value pair.
- `setdefault()`: Returns the value for a key if the key is in the dictionary; otherwise, inserts the key with a specified value.

44. What is a set in Python?

A set is an unordered collection of unique elements. Sets are mutable and can be used to store items without any duplicates.

45. How are sets implemented internally in Python?

Sets are implemented using hash tables, similar to dictionaries, but without the key-value pairing. Each element in a set is hashed to calculate an index for storage.

46. What are some common use cases for sets?

Sets are commonly used for membership testing (checking if an element is in a set), removing duplicates from a list, and performing mathematical set operations like union, intersection, and difference.

47. What are the time complexities for common set operations?

Adding, removing, and checking for the presence of elements in a set have an average time complexity of O(1).

48. Can set elements be of any type?

No, set elements must be of a type that is immutable and hashable. Common hashable types include strings, numbers, and tuples containing only hashable types.

49.What is the difference between a set and a frozenset?

A set is mutable, meaning its elements can be changed. A frozenset is immutable, meaning its elements cannot be changed once assigned. Frozensets can be used as dictionary keys or as elements of other sets because they are hashable.

50. What are some common methods available for sets?

Some common methods are:

- `add()`: Adds an element to the set.
- `remove()`: Removes a specified element from the set. Raises a KeyError if the element is not found.
- `discard()`: Removes a specified element from the set if it is present.
- `pop()`: Removes and returns an arbitrary element from the set.
- `union()`: Returns a set containing all elements from both sets.
- `intersection()`: Returns a set containing only elements that are present in both sets.
- `difference()`: Returns a set containing elements that are in the first set but not in the second.
- `symmetric_difference()`: Returns a set containing elements that are in either set, but not in both.
- `issubset()`: Checks if a set is a subset of another set.
- `issuperset()`: Checks if a set is a superset of another set.

51. What is a variable in Python?

A variable in Python is a symbolic name that references or points to an object. It acts as a storage location in memory for data that can be used and manipulated in a program.

52. How do you declare a variable in Python?

In Python, you declare a variable by simply assigning a value to it using the `=` operator. For example:

```
Example:
x                                   =                                   10
name                                =                                 "Alice"
```

53. What are the rules for naming variables in Python?

- Variable names must start with a letter (a-z, A-Z) or an underscore (_).
- The rest of the variable name can contain letters, digits (0-9), or underscores.
- Variable names are case-sensitive (`name` and `Name` are different variables).
- Variable names cannot be Python reserved keywords (e.g., `False`, `class`, `return`, etc.).

54. What is variable scope in Python?

Variable scope refers to the region of a program where a variable is defined and can be accessed. There are four types of variable scopes in Python:

- **Local Scope**: Variables defined inside a function, accessible only within that function.
- **Enclosing Scope**: Variables in the local scope of enclosing functions (non-local).
- **Global Scope**: Variables defined at the top-level of a script or module, accessible throughout the module.
- **Built-in Scope**: Variables preassigned in Python, such as built-in functions like `len()` and `range()`.

55. What is a global variable and how can you modify it inside a function?

A global variable is a variable that is defined outside of any function and is accessible throughout the entire module. To modify a global variable inside a function, you must use the `global` keyword:

56. What is a local variable and how is it different from a global variable?

A local variable is a variable that is declared inside a function and is accessible only within that function. It differs from a global variable, which is accessible throughout the entire module. Local variables take precedence over global variables with the same name within the function.

57. What is the `nonlocal` keyword in Python and when would you use it?

The `nonlocal` keyword is used to refer to variables defined in the nearest enclosing scope (not the global scope). It is typically used in nested functions to modify variables in the enclosing function's scope.

58. What is type inference in Python?

Type inference refers to the ability of Python to automatically determine the data type of a variable based on the value assigned to it. Python is a dynamically typed language, meaning you don't need to explicitly declare the type of a variable.

59. What are mutable and immutable variables in Python?

- **Mutable variables** are those whose values can be changed after they are created. Examples include lists, dictionaries, and sets.
- **Immutable variables** are those whose values cannot be changed once they are created. Examples include integers, floats, strings, and tuples.

60. What is variable shadowing and how can it affect your code?

Variable shadowing occurs when a variable declared within a certain scope (e.g., within a function) has the same name as a variable in an outer scope. This can lead to confusion and bugs, as the inner variable will "shadow" the outer one, making the outer variable inaccessible within the inner scope.

61. What is the difference between the `=` operator and the `==` operator in Python?

The `=` operator is the assignment operator, used to assign a value to a variable. The `==` operator is the equality operator, used to compare two values to see if they are equal.

62. How can you delete a variable in Python?

You can delete a variable in Python using the `del` statement:

63. What are environment variables and how are they used in Python?

Environment variables are external variables to a program, set in the operating system, and can influence the behavior of running processes. In Python, you can access environment variables using the `os` module.

64. What is variable interpolation and how can it be achieved in Python?

Variable interpolation is the process of embedding variables within a string. In Python, this can be achieved using f-strings, the `format()` method, or the `%` operator.

65. What is variable aliasing in Python?
Variable aliasing occurs when two different variable names refer to the same object in memory. Changes made through one alias will affect the other.

66. What are conditional statements in Python?
Conditional statements are used to perform different actions based on different conditions. The primary conditional statements in Python are `if`, `elif`, and `else`.

67. How does an `if` statement work in Python?

An `if` statement evaluates a condition. If the condition is `True`, the block of code within the `if` statement is executed.

68. What is the purpose of the `elif` statement in Python?

The `elif` (short for "else if") statement allows you to check multiple conditions. If the condition for `if` is `False`, it checks the condition for `elif` and executes the block of code if the `elif` condition is `True`.

69. How does an `else` statement work in Python?

The `else` statement is used to execute a block of code if none of the preceding conditions are `True`.

70. What is the difference between `if-elif-else` and multiple `if` statements?

`if-elif-else` statements are used when you have mutually exclusive conditions, meaning only one of the conditions can be true. Multiple `if` statements are used when you need to check several conditions independently.

71. What are control statements in Python?

Control statements in Python are used to control the flow of execution of a program. They include loops (`for` and `while`), and loop control statements (`break`, `continue`, and `pass`).

72. How does a `for` loop work in Python?

A `for` loop is used to iterate over a sequence (such as a list, tuple, string, or range). It executes a block of code for each item in the sequence.

73. How does a `while` loop work in Python?
A `while` loop repeatedly executes a block of code as long as a specified condition is `True`.

74. What is the `break` statement and how is it used?
The `break` statement is used to exit a loop prematurely when a certain condition is met.

75. What is the `continue` statement and how is it used?
The `continue` statement is used to skip the rest of the code inside the loop for the current iteration and move to the next iteration.

76. What is the `pass` statement and how is it used?
The `pass` statement is a null operation; it is used when a statement is required syntactically, but you do not want any command or code to execute.

77. What is the difference between `break` and `continue` statements?
The `break` statement terminates the loop entirely, while the `continue` statement only skips the current iteration and proceeds with the next iteration.

78. Can you use `else` with loops in Python?
Yes, you can use `else` with both `for` and `while` loops. The `else` block executes after the loop completes normally, but not if the loop is terminated by a `break` statement.

79. What is the purpose of nested loops in Python?
Nested loops are used to perform complex iterations. They allow you to loop over a sequence within another loop, often used for multidimensional data structures like matrices.

80. How do you control the number of iterations in a `for` loop using the `range()` function?
The `range()` function generates a sequence of numbers, which can be used to control the number of iterations in a `for` loop. It can take up to three arguments: start, stop, and step.

81. What is a function in Python?

A function is a block of reusable code that performs a specific task. It can take input, process it, and return an output.

82. How do you define a function in Python?

A function is defined using the `def` keyword, followed by the function name and parentheses containing any parameters.

83. How do you call a function in Python?
You call a function by using its name followed by parentheses.

84. What is the purpose of the `return` statement in a function?
The `return` statement is used to exit a function and return a value to the caller.

85. What is the difference between parameters and arguments in Python functions?
Parameters are the variables listed inside the parentheses in the function definition. Arguments are the values passed to the function when it is called.

86. What are default parameters in Python?

Default parameters allow you to specify default values for function parameters. If an argument is not provided during the function call, the default value is used.

87. What are keyword arguments in Python functions?
Keyword arguments are arguments passed to a function using the parameter names explicitly, allowing you to skip or reorder arguments.

88. What are variable-length arguments in Python functions?

Variable-length arguments allow a function to accept an arbitrary number of arguments. There are two types: `*args` for non-keyword arguments and `**kwargs` for keyword arguments.

89. What is the scope of a variable in Python?

The scope of a variable determines the portion of the program where the variable can be accessed. Variables defined inside a function are local to that function, while variables defined outside any function are global.

90. What is the difference between local and global variables in Python?

Local variables are defined within a function and can only be accessed inside that function. Global variables are defined outside all functions and can be accessed from any function.

91. How do you modify a global variable inside a function?
You can modify a global variable inside a function by using the `global` keyword.

92. What is a lambda function in Python?
A lambda function is an anonymous function defined using the `lambda` keyword. It can have any number of arguments but only one expression.

93. When should you use a lambda function in Python?
Lambda functions are used for short, simple operations where defining a full function would be overkill, such as in higher-order functions like `map()`, `filter()`, and `sorted()`.

94.What are higher-order functions in Python?
Higher-order functions are functions that take other functions as arguments, return a function as a result, or both. Examples include `map()`, `filter()`, and `reduce()`.

95. What is a decorator in Python?

A decorator is a function that takes another function and extends its behavior without explicitly modifying it. Decorators are commonly used for logging, enforcing access control, instrumentation, etc.

96. What are function annotations in Python?

Function annotations are a way of attaching metadata to function arguments and return values. They have no effect on the function's behavior but can be used for documentation or type checking.

97. What is recursion in Python?

Recursion is the process in which a function calls itself directly or indirectly to solve a problem. Recursive functions must have a base case to stop the recursion.

98. What is a docstring in Python?
A docstring is a string literal that appears as the first statement in a function, module, class, or method definition. It is used to document what the function or module does.

99. What is Object-Oriented Programming (OOP)?

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects, which can contain data and code to manipulate that data. It emphasizes the use of objects and classes.

100. What are classes and objects in Python?

A class is a blueprint for creating objects. An object is an instance of a class. Classes encapsulate data for the object and methods to manipulate that data.

101. What is the purpose of the `self` parameter in Python classes?
The `self` parameter refers to the instance of the class. It is used to access variables and methods associated with the current object.

102. What is inheritance in Python?

Inheritance is a mechanism by which one class (child or derived class) can inherit attributes and methods from another class (parent or base class).

103. What are the different types of inheritance in Python?
Python supports single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance, and hybrid inheritance.

104. What is multiple inheritance and how is it handled in Python?

Multiple inheritance occurs when a class inherits from more than one base class. Python handles multiple inheritance using the Method Resolution Order (MRO) to determine the order in which base classes are searched when executing a method.

105. What is encapsulation in Python?

Encapsulation is the practice of bundling data (attributes) and methods that operate on the data into a single unit, or class, and restricting access to some of the object's components.

106. How do you achieve data hiding in Python?
Data hiding is achieved by prefixing attributes with double underscores (`__`). This makes them private and not accessible outside the class.

107. What is polymorphism in Python?

Polymorphism allows methods to be defined in different classes, even if they share the same name. This can be achieved through method overriding or by defining methods with the same name in different classes.

108. What is method overriding in Python?
Method overriding occurs when a child class provides a specific implementation of a method that is already defined in its parent class.

109. What is abstraction in Python?
Abstraction is the concept of hiding the complex implementation details of a process while exposing only the necessary parts. In Python, abstraction can be achieved using abstract base classes (ABCs) from the `abc` module.

110. What are special methods in Python?

Special methods (also called magic methods or dunder methods) are methods with double underscores before and after their names. They are used to define how objects behave with built-in operations.

```
print(p3.x, p3.y) # Outputs: 4 6
```

111. What is the purpose of the `__init__` method?
The `__init__` method is a special method that initializes an object's attributes. It is called automatically when a new instance of the class is created.

112. What is a class method and how do you define it?
A class method is a method that is bound to the class and not the instance. It can modify the class state that applies across all instances of the class. Class methods are defined using the `@classmethod` decorator and the `cls` parameter.

113. What is a static method and how do you define it?
A static method is a method that does not operate on an instance or class. It is defined using the `@staticmethod` decorator. Static methods are used for utility functions that are related to the class but do not require access to class or instance attributes.

114. What is the difference between a class method and a static method?
A class method takes `cls` as the first parameter and can modify the class state, while a static method does not take `cls` or `self` and cannot modify the class or instance state.

115. How can you create a property in Python?

You can create a property using the `property()` function or the `@property` decorator. Properties allow you to define methods that behave like attributes.

116. What is the `__str__` method used for in Python?

The `__str__` method is a special method that is called by the `str()` built-in function and by the `print` function to return a string representation of an object.

117. What is file handling in Python?

File handling in Python refers to the ability to read from and write to files. Python provides built-in functions and methods to create, open, read, write, and close files.

118. How do you open a file in Python?

You can open a file using the `open()` function, which returns a file object. The `open()` function requires at least one argument: the file name. You can also specify the mode in which the file should be opened.

119. What are the different modes for opening a file in Python?

The different modes for opening a file in Python are:

- `'r'`: Read (default mode)
- `'w'`: Write (creates a new file or truncates an existing file)
- `'a'`: Append (writes data to the end of the file)
- `'b'`: Binary mode (used with other modes, e.g., `'rb'`, `'wb'`)
- `'t'`: Text mode (default mode, can be used with other modes, e.g., `'rt'`, `'wt'`)
- `'x'`: Exclusive creation (fails if the file already exists)

120. How do you close a file in Python?
You close a file using the `close()` method of the file object. This ensures that any changes made to the file are saved and resources are released.

121. How do you read the entire contents of a file in Python?

You can read the entire contents of a file using the `read()` method.

122. How do you read a file line by line in Python?

You can read a file line by line using the `readline()` method or by iterating over the file object.

123. What is the `readlines()` method used for?

The `readlines()` method reads all lines from a file and returns them as a list of strings.

124. How do you write to a file in Python?

You can write to a file using the `write()` method.

125. How do you write multiple lines to a file in Python?

You can write multiple lines to a file using the `writelines()` method, which takes a list of strings.

126. How do you read and write binary files in Python?
You can read and write binary files by opening the file in binary mode (`'b'`). For reading, use `'rb'`; for writing, use `'wb'`.

127. What is a context manager in Python and how is it used with files?

A context manager is used to manage resources, such as opening and closing files, ensuring that resources are properly released after use. The `with` statement is used to open a file and automatically close it when the block is exited.

128. How do you move the file pointer to a specific position in a file?

You can move the file pointer using the `seek()` method.

129. How do you get the current position of the file pointer?
You can get the current position of the file pointer using the `tell()` method.

130. What is the difference between text mode and binary mode in file handling?

In text mode (`'t'`), data is read and written as strings. In binary mode (`'b'`), data is read and written as bytes. Text mode handles character encoding automatically, while binary mode does not.

131. What are some common errors encountered in file handling, and how do you handle them?

Common errors include `FileNotFoundError`, `PermissionError`, and `IOError`. These can be handled using try-except blocks.

132. How do you check if a file exists before opening it?

You can check if a file exists using the `os.path.exists()` method from the `os` module.

133. How do you delete a file in Python?
You can delete a file using the `os.remove()` function from the `os` module.

134. What is a regular expression (regex)?

A regular expression (regex) is a sequence of characters that defines a search pattern. It is used for string matching, searching, and manipulation. Regular expressions are commonly used for tasks like validating input, searching within texts, and replacing substrings.

135. How do you create a regular expression in Python?

In Python, regular expressions are created using the `re` module. You can compile a regular expression pattern using the `re.compile()` function.

136. What are some common uses of regular expressions?
Common uses of regular expressions include:

- Validating input (e.g., email addresses, phone numbers)
- Searching for specific patterns within text
- Extracting specific parts of a string
- Replacing parts of a string

137. What does the `r` prefix mean in a regular expression string?

The `r` prefix denotes a raw string in Python. It tells Python to treat backslashes as literal characters and not as escape characters. This is useful in regular expressions where backslashes are common.

138. What does the following regular expression match: `\d+`?

The regular expression `\d+` matches one or more digits. The `\d` is a shorthand for any digit (0-9), and the `+` means one or more occurrences.

139. What are some common regex metacharacters and their meanings?

→ `.`: Matches any character except a newline
→ `^`: Matches the start of the string
→ `$`: Matches the end of the string
→ `*`: Matches zero or more occurrences of the preceding element
→ `+`: Matches one or more occurrences of the preceding element
→ `?`: Matches zero or one occurrence of the preceding element
→ `[]`: Matches any one character within the brackets
→ `|`: Acts as a logical OR
→ `()`: Groups patterns together

140. How do you search for a pattern in a string using regex in Python?

You can search for a pattern using the `re.search()` function, which returns a match object if the pattern is found, or `None` if not.

141. How do you find all occurrences of a pattern in a string using regex?

You can use the `re.findall()` function, which returns a list of all non-overlapping matches in the string.

142.How do you replace occurrences of a pattern in a string using regex?

You can use the `re.sub()` function, which replaces occurrences of the pattern with a specified replacement string.

143. How do you split a string by a pattern using regex?
You can use the `re.split()` function, which splits the string at each occurrence of the pattern.

144. What are capturing groups in regex and how do you use them?
Capturing groups are used to group parts of a pattern and capture the matched subpatterns. They are created by enclosing the pattern in parentheses.

145. What are non-capturing groups in regex and how do you use them?
Non-capturing groups are used to group parts of a pattern without capturing the matched subpatterns. They are created by enclosing the pattern in `(?:...)`.

146. How do you perform a case-insensitive match using regex in Python?
You can perform a case-insensitive match by using the `re.IGNORECASE` flag.

147. What are lookahead and lookbehind assertions in regex?
Lookahead and lookbehind assertions are used to assert that a pattern is followed or preceded by another pattern, without including the asserted pattern in the match.

- Lookahead: `(?=...)` (positive), `(?!...)` (negative)
- Lookbehind: `(?<=...)` (positive), `(?<!...)` (negative)

148. How do you compile a regex pattern for reuse in Python?
You can compile a regex pattern using the `re.compile()` function, which allows you to reuse the pattern for multiple operations.

149. What is the difference between `re.match()` and `re.search()` in Python?

The `re.match()` function checks for a match only at the beginning of the string, while `re.search()` checks for a match anywhere in the string.

150. How do you escape special characters in a regex pattern?

Special characters can be escaped using a backslash (`\`) to treat them as literal characters.

151. What are greedy and non-greedy quantifiers in regex?

Greedy quantifiers match as much of the input as possible, while non-greedy (or lazy) quantifiers match as little as possible. Non-greedy quantifiers are followed by a question mark (`?`).

152. What is a literal character in a regular expression?

A literal character in a regular expression is a character that matches exactly itself. For example, the regex `a` matches the character 'a' in the target string.

153. How do you match a special character (like a period or asterisk) literally in a regex pattern?

To match a special character literally, you need to escape it with a backslash (`\`). For example, to match a period, you use `\.`.

154. What will the regex pattern `abc` match?

The regex pattern `abc` will match the exact sequence of characters "abc" in the target string.

156. What does the dot (.) wildcard character match in a regular expression?

The dot (.) wildcard character matches any single character except a newline character (`\n`).

157. How do you modify the dot (.) wildcard to match newline characters as well?
You can modify the dot (.) wildcard to match newline characters by using the `re.DOTALL` flag.

158. What will the regex pattern `a.c` match in the string `abc a\ndc a1c`?
The regex pattern `a.c` will match the substrings "abc" and "a1c". To match "a\ndc" as well, you need to use the `re.DOTALL` flag.

159. What is a character class in a regular expression?

A character class in a regular expression is a set of characters enclosed in square brackets `[]` that matches any single character from that set. For example, `[abc]` matches 'a', 'b', or 'c'.

160. How do you match a digit using a character class?

You can match a digit using the character class `[0-9]`.

161. What will the regex pattern `[aeiou]` match?
The regex pattern `[aeiou]` will match any single lowercase vowel ('a', 'e', 'i', 'o', 'u').

162. How do you specify a range of characters in a character class?

You can specify a range of characters in a character class using a hyphen (`-`). For example, `[a-z]` matches any lowercase letter from 'a' to 'z'.

163. What does the character class `[^0-9]` match?

The character class `[^0-9]` matches any character that is not a digit. The caret (`^`) inside the brackets negates the character class.

164. How do you include a literal hyphen (`-`) in a character class?

To include a literal hyphen in a character class, place it at the beginning or end of the character class, or escape it with a backslash (\).