Python Assignment

**Q1. What are the types of Applications?**

Ans.1. Database apps

2. Custom apps

3. Web-based apps

4. Low code apps

5. No code apps

6. Enterprise apps

**Q2. What is programing?**

Ans. Programming to plan for something to happen at a particular time, to make somebody/something work or act automatically in a particular way.

**Q3. What is Python?**

Ans. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

**Q7. How memory is managed in Python?**

Ans. You can classify memory management in Python in one of two ways: dynamic allocation or static allocation. Dynamic allocation occurs as the program is running. This means that as the program operates, it can dynamically determine where to allocate memory while reusing and releasing it.

**Q8. What is the purpose continuing statement in python?**

Ans. The continue statement is used to skip the remaining code inside a loop for the current iteration only. For instance, let's use continue instead of a break statement in the previous example. When the condition num == 5 becomes True, the continue statement gets executed.

**Q17. What are negative indexes and why are they used?**

Ans. Negative indexes are a feature in some programming languages, like Python, that allow you to access elements of a sequence (like a list, tuple, or string) starting from the end of the sequence rather than the beginning.

Why Negative Indexes Are Used:

1. Convenience: They provide an easy way to access elements from the end of a sequence without needing to know the length of the sequence.

2. Readability: They can make code easier to read and understand, particularly when the intention is to access elements near the end of a sequence.

3. Efficiency: Negative indexing avoids the need to manually calculate the positive index when accessing elements from the end, reducing the chance of errors.

Negative indexes are particularly useful when you need to reference items relative to the end of a sequence, such as when you're interested in the last few elements.

**Q25. What is List? How will you reverse a list?**

Ans. A list is a data structure in Python that is used to store multiple items in a single variable. Lists are ordered, mutable (i.e., you can change their content), and can contain elements of different data types (e.g., integers, strings, or even other lists).

How to Reverse a List:

There are several ways to reverse a list in Python:

1. Using the reverse() Method:

The reverse() method reverses the elements of the list in place (i.e., it modifies the original list).

2. Using Slicing:

You can reverse a list using slicing, which creates a new list that is the reverse of the original.

3. Using the reversed() Function:

The reversed() function returns an iterator that yields the elements of the list in reverse order. You can convert this iterator back into a list.

**Q26. How will you remove last object from a list?**

Ans. 1. Pop() is the most common method and also allows you to capture the removed element.

2. Slicing creates a new list without the last element.

3. del removes the last element by directly deleting it from the list.

**Q28. Differentiate between append () and extend () methods?**

Ans. 1. append() when you want to add a single element to the list.

2. extend() when you want to add multiple elements from an iterable to the list.

**Q30. How will you compare two lists?**

Ans. 1. == Operator: Checks if two lists are exactly identical.

2. sorted() + ==: Checks if two lists have the same elements in any order.

3. set() + ==: Checks if two lists have the same unique elements (ignoring duplicates).

4. Element-wise comparison: Allows detailed, customized comparison between elements.

5. Sublist check: Useful to determine if all elements of one list are contained within another.

**Q43. What is tuple? Difference between list and tuple.**

Ans. Definition: A tuple is an ordered collection of items. It is defined by placing the items within parentheses (), separated by commas.

Mutability:

1. Mutability:

List: Lists are mutable, meaning their elements can be changed or updated after the list is created.

Tuple: Tuples are immutable, meaning once created, their elements cannot be changed or updated.

2. Syntax:

List: Lists are created using square brackets [].

Tuple: Tuples are created using parentheses ().

3. Performance:

List: Due to their mutability, lists generally have a bit more overhead compared to tuples. They can be slower for certain operations.

Tuple: Tuples, being immutable, are generally faster and have less memory overhead, making them more efficient in situations where a fixed-size, unchangeable sequence is needed.

4. Use Cases:

List: Lists are more suitable when you need a collection of items that may change during the program's execution.

Tuple: Tuples are often used to represent fixed collections of items (e.g., coordinates, records) and can be used as dictionary keys due to their immutability.

5. Methods:

List: Lists have various methods available, such as .append(), .extend(), .remove(), and .pop().

Tuple: Tuples have fewer methods available since they are immutable. The primary methods are count() and .index().

6. Syntax for Single Element:

List: To create a single-element list, you use a trailing comma inside the brackets.

Tuple: To create a single-element tuple, you must include a trailing comma inside the parentheses.

**Q47. How will you create a dictionary using tuples in python?**

Ans. 1. Using dict() with a list of tuples is the most straightforward and commonly used method.

2. Adding individual tuples directly to the dictionary is also possible but less common.

3. Dictionary comprehension offers a more flexible way to create a dictionary from a list of tuples, especially if you need to manipulate the data during creation.

**Q 51. How Do You Traverse Through a Dictionary Object in Python?**

Ans. 1. Keys: Use for key in my\_dict: to iterate over keys.

2. Values: Use for value in my\_dict.values(): to iterate over values.

3. Key-Value Pairs: Use for key, value in my\_dict.items(): to iterate over key-value pairs.

4. Comprehensions: Use dictionary comprehensions to create new dictionaries or lists based on existing ones.

5. Enumerate: Use enumerate() if you need index positions during iteration

**Q52. How Do You Check the Presence of a Key in A Dictionary?**

Ans. 1. in Operator: Most common and efficient way to check for the presence of a key.

2. get() Method: Useful if you want to retrieve the value and check for key presence simultaneously.

3. keys() Method: Checks key presence by iterating over the dictionaryâ€™s keys.

Each method has its use cases, but the in operator is generally preferred for its simplicity and readability.

**Q65. How Many Basic Types of Functions Are Available in Python?**

Ans. 1. Built-in Functions: Predefined functions available in Python.

2. User-Defined Functions: Functions defined by the user using def.

3. Lambda Functions: Anonymous functions defined using lambda.

4. Recursive Functions: Functions that call themselves.

5. Higher-Order Functions: Functions that take or return other functions.

6. Generator Functions: Functions that use yield to produce an iterator.

These types of functions provide flexibility and power in Python programming, enabling a wide range of functional programming techniques and design patterns.

**Q67. How can you pick a random item from a range?**

Ans. 1. Using random.randrange()

The random.randrange() function can be used to select a random number from a specified range. It works with integer ranges and allows you to specify the start, stop, and step.

2. Using random.randint()

The random.randint() function can also be used to pick a random integer from a specified range, inclusive of both endpoints.

**Q68. How can you get a random number in python?**

Ans. 1. random.randint(a, b): Returns a random integer N such that a <= N <= b.

2. random.randrange(start, stop): Returns a randomly selected element from the range [start, stop), where stop is exclusive.

3. random.random(): Returns a random floating-point number between 0.0 and 1.0.

4. random.uniform(a, b): Returns a random floating-point number N such that a <= N <= b.

5. random.gauss(mu, sigma): Returns a random number from a Gaussian (normal) distribution with mean mu and standard deviation sigma.

These functions allow you to generate random numbers based on your needs, whether you require integers, floating-point numbers, or numbers from a specific distribution.

**Q69. How will you set the starting value in generating random numbers?**

Ans. 1. random.seed(a=None): Initializes the random number generator with a specific seed value. The seed value a can be any hashable object, typically an integer.

2. Reproducibility: By setting the seed to the same value (42 in the example), the sequence of random numbers generated will be the same each time the code is executed. This is useful for debugging or for situations where you need repeatable results.

**Q70. How will you randomize the items of a list in place?**

Ans. 1. import random: Imports the random module which provides various functions for generating random numbers and performing random operations.

2. random.shuffle(list): Randomly shuffles the items of the list list in place. This means that the original list is modified and no new list is created.

**Q71. What is File function in python? What are keywords to create and write file.**

Ans. 1. open(filename, mode): Opens a file with the specified mode ('r', 'w', 'a', etc.).

2. write(): Writes a string to the file.

3. writelines(): Writes a list of strings to the file.

4. read(): Reads the entire file content.

5. readline(): Reads a single line from the file.

6. readlines(): Reads all lines into a list.

7. close(): Closes the file.

with statement: Simplifies file handling by automatically managing file open/close operations.

These functions and methods help manage file operations effectively in Python.

**Q83. Explain Exception handling? What is an Error in Python?**

Ans. Exception Handling in Python-

Exception handling is a mechanism in Python that allows you to manage and respond to runtime errors or exceptional conditions that arise during the execution of a program. It provides a way to gracefully handle errors without terminating the program abruptly.

Key Concepts

1. Exception :- An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. Examples include division by zero, file not found, or invalid user input.

2. Error :- An error is a specific type of exception that usually indicates a serious issue that the program cannot recover from, such as syntax errors or runtime errors.

3. Try and Except Blocks :- Contains code that might raise an exception.

except block: Contains code that executes if an exception occurs in the try block.

4. Finally Block :- Contains code that always executes, regardless of whether an exception occurred or not. It is typically used for cleanup actions, such as closing files or releasing resources.

5. Else Block :- Contains code that executes if no exceptions occur in the try block. It is optional and used for code that should run when no exceptions are raised.

6. Raising Exceptions :- We can manually raise exceptions using the raise statement to indicate that an error condition has occurred.

**Q84. How many except statements can a try-except block have? Name Some built-in exception classes:**

Ans. Built-in Exception Classes

Python has several built-in exception classes that you can use to handle different types of errors. Some of the most common ones include:

1. Exception: The base class for all built-in exceptions.

2. ArithmeticError: Base class for arithmetic-related errors.

3. ZeroDivisionError: Raised when dividing by zero.

4. OverflowError: Raised when an operation exceeds the limits of a numeric type.

5. FloatingPointError: Raised when a floating-point operation fails.

6. AttributeError: Raised when an invalid attribute reference is made.

7. IndexError: Raised when an index is out of range for a sequence.

8. KeyError: Raised when a dictionary key is not found.

9. FileNotFoundError: Raised when a file or directory is requested but cannot be found.

10. IOError: Raised when an I/O operation (such as reading or writing a file) fails.

11. ValueError: Raised when a function receives an argument of the right type but inappropriate value.

12. TypeError: Raised when an operation or function is applied to an object of inappropriate type.

13. NameError: Raised when a local or global name is not found.

14. RuntimeError: Raised for errors that do not fall into other categories.

15. NotImplementedError: Raised when an abstract method or function is not implemented.

16. SyntaxError: Raised when there is a syntax error in the code.

17. IndentationError: A subclass of SyntaxError raised when there is incorrect indentation.

18. EOFError: Raised when the end of a file is reached unexpectedly.

These built-in exceptions cover a wide range of error conditions and can be used to handle errors more effectively in your programs.

**Q85. When will the else part of try-except-else be executed?**

Ans. 1. The else block runs only if no exceptions are raised in the try block.

2. It is useful for placing code that should execute only if the try block is successful, ensuring that it does not run if an exception occurs.

**Q86. Can one block of except statements handle multiple exception?**

Ans. 1. Single except Block: You can catch multiple exceptions with a single except block by specifying them in a tuple.

2. Handling Multiple Exceptions: This approach allows you to handle various exceptions in a unified manner, which can be useful if the handling logic for the exceptions is the same or similar.

This feature is particularly useful for simplifying code and avoiding redundant except blocks for exceptions that should be handled similarly.

**Q87. When is the finally block executed?**

Ans. 1. The finally block ensures that certain code is executed regardless of whether an exception occurs or not.

2. It is useful for performing clean-up actions such as closing files, releasing locks, or freeing up resources that were acquired in the try block.

3. Even if an exception is raised and not caught, the finally block is still executed before the exception is propagated further.

**Q88. What happens when „1‟== 1 is executed?**

Ans. 1. '1' == 1: Evaluates to False because Python does not consider a string and an integer to be equal.

2. Type Comparison: In Python, equality comparisons between different types (such as string and integer) typically result in False.

**Q89. How Do You Handle Exceptions with Try/Except/Finally in Python?**

Ans. 1. try Block: Contains code that might raise an exception.

2. except Block: Catches and handles exceptions.

3. finally Block: Executes regardless of whether an exception occurred, ideal for cleanup tasks.

This structure allows you to manage exceptions gracefully, ensuring necessary cleanup and resource management even if errors occur during execution.