1. What are the new features added in Python 3.8 version?

Answer :

Python 3.8 introduced several new features and improvements. Here are some notable additions in Python 3.8:

1. Assignment Expressions (the "Walrus Operator"): Python 3.8 introduced the := operator, known as the Walrus Operator.

2. Positional-only Parameters: Python 3.8 introduced the ability to define function parameters that can only be passed positionally, not as keyword arguments. This helps in enforcing the intended usage and provides more flexibility for function signatures.

3. f-strings Support "=" for Self-Documenting Expressions: Python 3.8 expanded the functionality of f-strings to support an = specifier. It allows expressions inside f-strings to include a debugging representation of the evaluated expression along with the result.

4.Simplified Debugging with f-strings: Python 3.8 introduced the ability to use f-strings directly inside pdb, the Python debugger. This makes it easier to display and examine variable values during debugging sessions.

5. Syntax Warning for Unparenthesized Context Managers: Python 3.8 introduced a new syntax warning to encourage the use of parentheses around multiple context managers in with statements. This helps avoid potential ambiguity and improves code clarity.

6. Improved Performance: Python 3.8 included various performance enhancements and optimizations, such as faster function calls, improved dictionary performance, and optimized built-in functions.

1. What is monkey patching in Python?

Answer : Monkey patching in Python refers to the practice of modifying or extending the behavior of an existing module, class, or object at runtime by adding, replacing, or modifying its attributes or methods. It allows you to dynamically modify the code and behavior of existing objects without altering their original implementation.

The term "monkey patching" comes from the idea that you are making changes to the code on the fly, similar to how a mischievous monkey might tinker with things.

class MyClass:

def say\_hello(self):

print("Hello!")

def say\_goodbye(self):

print("Goodbye!")

obj = MyClass()

obj.say\_hello() # Output: Hello!

MyClass.say\_hello = say\_goodbye

obj.say\_hello() # Output: Goodbye!

1. What is the difference between a shallow copy and deep copy?

Answer : The difference between a shallow copy and a deep copy lies in how they handle nested objects or collections when making a copy of an object.

Shallow Copy:

A shallow copy creates a new object and copies the references of the nested objects from the original object to the new object. In other words, it creates a new container object but still references the same nested objects as the original. Changes made to the nested objects in either the original or the shallow copy will be reflected in both. It copies the top-level structure, but not the complete nested structure.

Deep Copy:

A deep copy creates a new object and recursively copies the nested objects from the original object to the new object. It creates entirely independent copies of the original object and all its nested objects. Changes made to the nested objects in the original or the deep copy do not affect each other. It copies both the top-level structure and the complete nested structure.

1. What is the maximum possible length of an identifier?

Answer : In Python, the maximum possible length of an identifier (variable, function, class name, etc.) is not explicitly defined. However, there are practical limits imposed by the programming language and the underlying system.

1. What is generator comprehension?

Answer : Generator comprehension, also known as generator expression, is a concise way to create and define generator objects in Python. It is similar to list comprehensions but with a crucial difference: instead of creating a list, it creates a generator.