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

Ans1

Python 3.8, which was released on October 14, 2019, introduced several new features and improvements. Some of the notable new features are:

Assignment expressions (also known as the walrus operator) - allows the assignment of values to variables as part of an expression. For example, instead of writing if len(my\_list) > 0: x = my\_list[0], you can now write if (x := len(my\_list)) > 0: print(x).

Positional-only parameters - allows a function to specify that some parameters can only be passed positionally and not as keyword arguments. This can make the function's signature clearer and easier to use correctly.

f-strings now support a new syntax for specifying precision and width, making them more flexible and easier to use.

A new parser for CPython, called PEG parser, which is more flexible and easier to maintain than the old LL(1) parser.

The math module has a new function, math.isqrt(), that returns the integer square root of a non-negative integer.

New features and improvements for Python's typing module, including the TypedDict type hint and support for specifying variable annotations in a function's signature.

Improvements to asyncio, including the addition of the asyncio.run() function for easily running an asyncio program, and the ability to cancel a Task from within itself.

Several other smaller improvements and optimizations, such as faster f-string evaluation, improved traceback formatting, and a new statistics.mode() function.

These are just some of the new features in Python 3.8; there are many others as well.

2. What is monkey patching in Python?

Ans2

Monkey patching in Python is the practice of changing or adding code to a module or class at runtime, without changing the source code. This is done by dynamically modifying or replacing attributes, methods, or functions of an object or module at runtime, often for the purpose of fixing a bug, adding a feature, or changing the behavior of the code.

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

Ans3

a copy of an object can be created using either a shallow copy or a deep copy. The main difference between the two is how they handle mutable objects that are nested within the object being copied.

A shallow copy creates a new object that references the same memory locations as the original object. This means that any changes made to the copy will also affect the original object. However, if the original object contains mutable objects, such as lists or dictionaries, the references to those objects will be copied as well, meaning that any changes made to the mutable objects will also affect both the original and copied objects.

original = [1, 2, [3, 4]]

copy = original.copy() # or copy = original[:]

copy[0] = 5

copy[2][0] = 6

print(original) # Output: [1, 2, [6, 4]]

print(copy) # Output: [5, 2, [6, 4]]

import copy

original = [1, 2, [3, 4]]

copy = copy.deepcopy(original)

copy[0] = 5

copy[2][0] = 6

print(original) # Output: [1, 2, [3, 4]]

print(copy) # Output: [5, 2, [6, 4]]

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

Ans4

the maximum length of an identifier is not explicitly defined. However, there are some practical limits that can affect the maximum length of an identifier in practice.

According to the Python documentation, an identifier is a sequence of letters, digits, and underscores that starts with a letter or underscore. The length of an identifier is limited by the maximum size of a string in Python, which can vary depending on the implementation and the amount of memory available. In practice, most Python implementations limit the length of a string to 2^31 - 1 (or 2^63 - 1 on a 64-bit system), which is the maximum value that can be represented by a signed integer.

5. What is generator comprehension?

Ans5

Generator comprehension, also known as generator expression, is a concise way to create a generator object in Python. It is similar to list comprehension, but instead of creating a list, it creates a generator that produces values on the fly as they are needed.

The syntax for generator comprehension is very similar to list comprehension, with the main difference being the use of parentheses instead of brackets.

# List comprehension

my\_list = [x\*\*2 for x in range(10)]

# Generator comprehension

my\_generator = (x\*\*2 for x in range(10))