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

Sol:-

Assignment Expressions (the Walrus Operator): Introduced the := operator, also known as the walrus operator, which allows assignment within expressions. It enables you to assign a value to a variable as part of a larger expression.

Positional-Only Parameters: The function parameter syntax was extended to support positional-only parameters, denoted by a / marker. This allows defining function parameters that can only be passed positionally and not by keyword.

f-strings support = for self-documenting expressions: Formatted string literals (f-strings) gained support for the = sign within expressions, enabling better debugging and self-documenting code by displaying the evaluated value along with the expression.

Simplified Dictionary Merge with the | Operator: The | operator was introduced for dictionary merge operations, providing a concise way to merge dictionaries.

The math.prod() function: The math module now includes a prod() function that calculates the product of all elements in an iterable. It simplifies the process of calculating the product without the need for a loop.

Improved Syntax Warnings and Error Handling: Python 3.8 introduced improved warnings and error messages for various syntax-related issues, making it easier to identify and fix potential problems in code.

1. What is monkey patching in Python?

Sol:-

Monkey patching in Python refers to the practice of modifying or extending the behavior of an existing module, class, or object at runtime. It involves adding, modifying, or replacing attributes, methods, or functions of an object without altering the original source code.

Monkey patching allows developers to dynamically change the behavior of existing code without having to modify the original implementation. It can be useful in situations where modifying the source code directly is not feasible or desired, such as when working with third-party libraries or modules.

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

Sol:- A shallow copy creates a new object that references the original elements of the copied object.

In other words, it creates a new object but still shares the internal references to the objects within the copied object. Changes made to the copied object will affect the original object, and vice versa. Shallow copy is performed using the copy() method or the slicing syntax [:].

import copy

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

copied\_list = copy.copy(original\_list)

# Modifying the nested list in the copied object

copied\_list[2][0] = 5

print(original\_list) # Output: [1, 2, [5, 4]]

print(copied\_list) # Output: [1, 2, [5, 4]]

On the other hand, a deep copy creates a completely independent object with its own set of copied

elements.

It recursively copies all the objects nested within the original object, creating separate copies for each level. Changes made to the copied object will not affect the original object, and vice versa. Deep copy is performed using the deepcopy() method from the copy module.

import copy

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

copied\_list = copy.deepcopy(original\_list)

# Modifying the nested list in the copied object

copied\_list[2][0] = 5

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

print(copied\_list) # Output: [1, 2, [5, 4]]

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

Sol:-

In Python 3.x, the recommended maximum line length, including identifiers, is 79 characters as per the official Python Style Guide (PEP 8). However, this guideline specifically refers to the recommended line length for code readability and maintainability, rather than being a hard limit imposed by the Python language itself.

1. What is generator comprehension?

Sol:-

Generator comprehension, also known as generator expression, is a concise way to create a generator in Python. It provides a compact syntax for generating elements on-the-fly without creating a full list or other sequence type.

Generator comprehension has a similar syntax to list comprehension, but instead of enclosing the expression within square brackets ([]), it is enclosed within parentheses (()).

even\_numbers = (x for x in range(1, 11) if x % 2 == 0)