**ASSIGNMENT – 15**

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

**Ans:**

Python 3.8 introduced several new features and improvements, including:

1. Assignment Expressions (the Walrus Operator): This new operator (:=) allows assignment within an expression, allowing for more concise and readable code.

2. Positional-only parameters: A new syntax (/) allows defining function parameters that can only be passed by position and not by keyword. This can be useful for defining functions with a fixed number of arguments.

3. f-strings support =: A new `=` specifier can be used within f-strings to print the value of an expression with a specific format, similar to the `format()` method.

4. Improved syntax errors: Syntax errors in Python 3.8 provide more helpful and informative error messages, making it easier to debug code.

5. New modules and functions: Several new modules and functions were added to Python 3.8, including the `math.isqrt()` function for computing integer square roots and the `statistics.fmean()` function for computing the arithmetic mean of floating-point values.

1. **What is monkey patching in Python?**

**Ans:**

Monkey patching is a technique in Python where you dynamically modify a class or module at runtime by adding, removing, or replacing attributes, methods, or functions. This technique is sometimes used to change the behavior of a module or class without modifying the original code.

In Python, modules and classes are first-class objects, which means that they can be assigned to variables, passed as arguments to functions, and modified at runtime. Monkey patching takes advantage of this flexibility to modify the behavior of a module or class after it has been imported or defined.

Here's an example of monkey patching a class in Python:

```python

class MyClass:

def my\_method(self):

print("Original behavior")

# Create an instance of MyClass

obj = MyClass()

# Define a new method for MyClass

def new\_method(self):

print("New behavior")

# Monkey patch the new method into the MyClass class

MyClass.my\_method = new\_method

# Call the my\_method() method on the MyClass instance

obj.my\_method() # Output: "New behavior"

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

**Ans:**

In Python, copying an object means creating a new object with the same contents as the original. There are two main ways to copy an object in Python: shallow copy and deep copy.

A shallow copy creates a new object that shares some of the contents of the original object. Specifically, the new object points to the same memory locations as the original for some or all of its contents. In other words, a shallow copy creates a new object, but its contents are still connected to the original object.

In contrast, a deep copy creates a new object that is completely independent of the original object. In other words, a deep copy creates a new object with the same contents as the original, but the new object has its own memory locations for all of its contents.

Here's an example that demonstrates the difference between shallow and deep copies:

import copy

# create a list with nested lists

original = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# make a shallow copy of the list

shallow\_copy = original.copy()

# make a deep copy of the list

deep\_copy = copy.deepcopy(original)

# modify the first item in the original list

original[0][0] = 99

# print the original, shallow copy, and deep copy

print("Original:", original)

print("Shallow copy:", shallow\_copy)

print("Deep copy:", deep\_copy)

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

**Ans:**

In Python, an identifier is a name used to identify a variable, function, class, module, or other object. Identifiers in Python can be composed of letters, digits, and underscores (\_), but cannot start with a digit. There is no maximum length for an identifier in Python, but it is recommended to keep the length reasonable for readability and maintainability purposes.

The practical limit for the length of an identifier in Python is determined by the maximum length of a string in the Python implementation being used. In CPython (the reference implementation of Python), the maximum length of a string is limited to the maximum value of a Py\_ssize\_t integer, which varies depending on the platform and the build options used. Typically, the maximum length of a string in CPython is several billion characters, which is more than enough for any practical purpose. Therefore, the maximum length of an identifier in Python is effectively limited only by the available memory on the system.

1. **What is generator comprehension?**

**Ans:**

Generator comprehension is a concise way to create generator objects in Python. It is similar to list comprehension, but instead of creating a list, it creates a generator that yields values on demand.

The syntax for generator comprehension is similar to that of list comprehension, but uses parentheses instead of square brackets:

```

(generator expression)

```

Here's an example that demonstrates how to use generator comprehension to create a generator object that yields the square of each number in a list:

```python

numbers = [1, 2, 3, 4, 5]

squares = (x\*\*2 for x in numbers)

for n in squares:

print(n)

```

In this example, we create a list of numbers and then use generator comprehension to create a generator object that yields the square of each number. We then iterate over the generator object using a `for` loop and print each value.

The output of this code will be:

1

4

9

16

25