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

Some of the key features introduced in **Python 3.8** include:

* **Assignment Expressions (The "walrus operator")**:  
  The new := operator allows assignment inside an expression, which can make code more concise.  
  Example:
* if (n := len(my\_list)) > 10:
* print(f"List is too long with {n} elements")
* **Positional-only parameters**:  
  You can now specify that certain function parameters must be specified positionally and cannot be used as keyword arguments using /. Example:
* def func(a, b, /, c, d):
* print(a, b, c, d)
* **f-strings improvements**:  
  You can now use the = operator inside f-strings to both evaluate and display the expression. Example:
* x = 10
* print(f'{x=}') # Output: x=10
* **TypedDict**:  
  A new way to specify dictionaries with a fixed set of keys and their types. This helps with type checking.
* from typing import TypedDict
* class Person(TypedDict):
* name: str
* age: int
* **New syntax warnings**:  
  A new SyntaxWarning has been added for using is to compare literals.
* **Python 3.8 also includes other improvements** such as changes in the standard library, new built-in functions, and more.

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

**Monkey patching** refers to the practice of dynamically changing or modifying a class or module at runtime, usually by adding or modifying methods or attributes. This can be useful for debugging or testing, but it can also lead to issues if used improperly, as it modifies the behavior of code that was already written.

Example:

# Original function

class MyClass:

def greet(self):

print("Hello!")

# Monkey patching the greet method

def new\_greet(self):

print("Hello, modified greeting!")

MyClass.greet = new\_greet # Replacing the original method with the new one

obj = MyClass()

obj.greet() # Output: Hello, modified greeting!

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

* **Shallow copy**:  
  A shallow copy creates a new object, but does not recursively copy the nested objects. Instead, it copies references to the original objects. As a result, if the original object is modified, the changes may affect the shallow copy as well. Example:
* import copy
* original = [1, [2, 3]]
* shallow = copy.copy(original)
* shallow[1][0] = 99
* print(original) # Output: [1, [99, 3]]
* **Deep copy**:  
  A deep copy creates a new object and recursively copies all objects contained in the original object, including nested objects. The deep copy is independent of the original object, so changes to the original do not affect the deep copy. Example:
* import copy
* original = [1, [2, 3]]
* deep = copy.deepcopy(original)
* deep[1][0] = 99
* print(original) # Output: [1, [2, 3]]

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

In Python, there is **no explicit limit** on the length of an identifier (i.e., variable names, function names, class names, etc.) in the language specification. However, practical limitations are imposed by the system and the underlying implementation of Python, and the length is typically constrained by the available memory or the platform's limitations. Therefore, Python identifiers can technically be **arbitrary in length**.

**5. What is generator comprehension?**

**Generator comprehension** is similar to list comprehension, but instead of creating a list, it returns a generator object. The generator produces items one at a time, which is more memory-efficient than creating a complete list upfront, especially for large datasets.

Generator comprehensions use parentheses () instead of square brackets [].

Example:

gen = (x \* x for x in range(5))

for val in gen:

print(val) # Output: 0, 1, 4, 9, 16

Here, (x \* x for x in range(5)) creates a generator that computes x \* x for each value of x in the range.

Generators are useful when you don't need all the items at once and want to save memory by generating items lazily, one at a time.