Python Advanced Assignment-15

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

Ans.

Python 3.8 introduced several new features and optimizations. Some of the key features include:

- Walrus operator (`:=`): This allows assignment expressions, which assign values to variables as part of expressions. Example:

```
if (n := len(my_list)) > 10:
    print(f"List is too long ({n} elements)")
```

- Positional-only parameters: The ability to define function parameters that can only be passed positionally (i.e., not as keyword arguments). This is done using a `/` in the function signature. Example:

```
def func(a, b, /, c, d):
    return a + b + c + d
```

- f-strings enhancements: F-strings (formatted string literals) now support the `=` specifier to provide more detailed output.

```
x = 10
print(f'{x=}')  # Outputs: x=10
```

- `math.prod()`: A new function to calculate the product of elements in an iterable, similar to `sum()` but for multiplication.

```
import math
result = math.prod([1, 2, 3, 4])  # Outputs: 24
```

- `reversed()` on dictionaries: Reversing the order of dictionaries is now supported using `reversed()`.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
reversed_dict = reversed(my_dict)
```

- `typed_dict` and `Final`: Introduced as part of PEP 589, `TypedDict` allows for dictionaries with a specified type signature, and `Final` is used to denote that a value or method cannot be overridden.

- `__import__` improvements: More granular control of the import mechanism and better performance for certain use cases.

2. What is monkey patching in Python?

Ans.

Monkey patching in Python refers to the dynamic modification or extension of classes or modules at runtime. It allows you to change or add attributes, methods, or functions of a class or module after it has been loaded.

Example:

```
class A:
    def greet(self):
        print("Hello from A")


def new_greet():
    print("Hello from the patched version!")


# Monkey patching the greet method
A.greet = new_greet


# Now it will use the new method
obj = A()
obj.greet()  # Outputs: Hello from the patched version!
```

Monkey patching should be used carefully, as it can lead to code that's difficult to debug and maintain.

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

Ans.

- Shallow copy: A shallow copy creates a new object, but it copies only the reference to the original nested objects. If the original object contains references (like lists or dictionaries inside it), a shallow copy will not copy the nested objects—changes to the nested objects in the original will reflect in the shallow copy as well.

Example:

```
import copy
original = [[1, 2, 3], [4, 5, 6]]
```

```
shallow_copy = copy.copy(original)

shallow_copy[0][0] = 100

print(original)  # [[100, 2, 3], [4, 5, 6]]
```

- Deep copy: A deep copy creates a new object and recursively copies all objects, including nested ones. It creates independent copies of all objects, so changes in the deep copy do not affect the original object.

Example:

```
import copy

original = [[1, 2, 3], [4, 5, 6]]

deep_copy = copy.deepcopy(original)

deep_copy[0][0] = 100

print(original)  # [[1, 2, 3], [4, 5, 6]]
```

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

Ans.

In Python, an identifier (the name of a variable, function, or class) can have an unlimited length, although in practice, it is constrained by memory limitations and readability considerations. However, it's advised to keep identifiers reasonably short and meaningful.

5. What is generator comprehension?

Ans.

Generator comprehension is similar to list comprehension but instead of creating a list, it creates a generator object that generates items lazily (on demand). This is more memory-efficient, especially for large datasets, because it doesn't store all the items in memory at once.

Syntax is the same as list comprehension, but you use parentheses `()` instead of square brackets `[]`.

Example:

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

print(gen)  # <generator object>

print(list(gen))  # Outputs: [0, 1, 4, 9, 16]
```

Each value is generated only when required, making it ideal for scenarios involving large data or infinite sequences.