PYTHON ASSIGNMENT - 25

1) . What is the difference between enclosing a list comprehension in square brackets and

parentheses?

2) What is the relationship between generators and iterators?

3) What are the signs that a function is a generator function?

4) What is the purpose of a yield statement?

5) What is the relationship between map calls and list comprehensions? Make a comparison and

contrast between the two.

ANSWERS

1.Difference between square brackets and parentheses in list comprehensions:

- Using square brackets (**[]**) creates a list comprehension, and it produces a new list.
- new_list = [x**2 for x in range(5)] # [0, 1, 4, 9, 16]

Using parentheses (**()**) creates a generator expression, and it produces a generator object.
generator = (x**2 for x in range(5))
The main difference is that a list comprehension creates and returns a list, while a generator expression creates an iterator (generator) that generates values on-the-fly.

2 Relationship between generators and iterators:

- All generators are iterators, but not all iterators are generators.
- Generators are a specific type of iterator created using a function with the **yield** statement.

- Iterators, in general, are objects that implement the **__iter__** and **__next__** methods or use the **iter** and **next** functions.

3. **Signs that a function is a generator function:**
   a. The presence of the **yield** statement is a clear sign that a function is a generator function.
   b. Generator functions use **yield** to produce a series of values, and they maintain their state between calls.
4. **Purpose of a yield statement:**
   a. The **yield** statement is used in a generator function to produce a value to the caller without terminating the function's execution.
   b. When a generator function encounters a **yield** statement, it suspends its state and returns the yielded value to the caller.
   c. The function can later be resumed from where it left off, maintaining its local variables and state.
5. **Relationship between map calls and list comprehensions:**
   a. Both **map** calls and list comprehensions are used to apply a function to each element of an iterable.
   b. **Map:**
   c. result_map = map(lambda x: x**2, [1, 2, 3, 4]) # <map object>

List comprehension: result_list_comp = [x**2 for x in [1, 2, 3, 4]] # [1, 4, 9, 16]

- **Comparison:**
  - **Creation:**
    - **map** returns a map object and requires converting to a list if a list is desired.
    - List comprehension directly creates a list.
  - **Readability:**
    - List comprehensions are often considered more readable and concise.
  - **Lazy evaluation:**
    - **map** is lazily evaluated, producing values on demand.
    - List comprehensions are eagerly evaluated, creating the entire list at once.

- **Functionality:**
    - List comprehensions can include conditional statements, making them more versatile in certain cases.
- **Contrast:**
    - List comprehensions are more concise and often considered more readable.
    - **map** is lazily evaluated, potentially saving memory for large iterables.
    - List comprehensions can include conditionals more naturally.
- **Choosing between them:**
    - Use list comprehensions when you want a concise and readable way to create a list.
    - Use **map** when you need lazy evaluation or want to apply a function element-wise without creating a list.