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) In Python, enclosing a list comprehension in square brackets (`[]`) creates a list, while enclosing it in parentheses (`()`) creates a generator expression.

- List comprehension (with square brackets): It generates and returns a list containing the results of applying an expression to each item in an iterable.

Example:

```

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

squares = [x \*\* 2 for x in numbers]

# squares is now [1, 4, 9, 16, 25]

```

- Generator expression (with parentheses): It creates an iterator, which generates values on-the-fly when needed, without storing them in memory. This is more memory-efficient for large data sets.

Example:

```

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

square\_generator = (x \*\* 2 for x in numbers)

# square\_generator is an iterator

```

2) Generators are a type of iterator in Python. An iterator is an object that can be iterated (looped) over, and it typically represents a stream of data that can be generated lazily. A generator is a specific type of iterator that is created using a special function called a generator function. In essence:

- A generator is an iterator, but not all iterators are generators.

- Generators are defined using functions with the `yield` statement, which allows them to yield values one at a time.

3) A function is a generator function in Python if it contains the `yield` statement at least once. The presence of the `yield` statement differentiates generator functions from regular functions. When a generator function is called, it doesn't execute the function body immediately. Instead, it returns a generator object, which can be used to control the execution of the function and retrieve values using the `next()` function or in a `for` loop.

4) The purpose of a `yield` statement in a generator function is to yield a value to the caller and temporarily pause the function's execution. This allows the function to be resumed from where it left off when the next value is requested. It essentially transforms the function into an iterator that can produce values lazily, making it memory-efficient for handling large data sets or infinite sequences.

5) Map calls and list comprehensions are both used for transforming data in Python, but they have some differences:

- Map:

- The `map` function applies a given function to each item in one or more iterables (e.g., lists) and returns an iterator that produces the results one by one.

- It's primarily used for one-to-one transformations of data.

- It requires a separate function to be defined and passed as an argument.

Example:

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

squares = map(lambda x: x \*\* 2, numbers)

# squares is an iterator

```

- List Comprehension:

- A list comprehension generates a new list by applying an expression to each item in an iterable.

- It's used for creating a new list, not just transforming values.

- The transformation logic is embedded within the comprehension itself.

Example:

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

squares = [x \*\* 2 for x in numbers]

# squares is a list

```

In summary, both `map` and list comprehensions can be used for similar transformations, but list comprehensions are more concise and result in a new list, while `map` returns an iterator. Generator expressions, as mentioned earlier, are similar to list comprehensions but generate values lazily like generators, making them more memory-efficient for large datasets.