Part-8

Python Iterables, Iterators, and Generators

Iterables

- Iterables are objects on which you can perform iteration, such as lists, tuples, dictionaries, and sets.
- Iterables have built-in `__iter__` and `__next__` methods that allow you to create an iterator object to access their el ements.

Iterators

- Iterators are objects that implement the `__iter__` and `__next__` methods, which allow you to traverse through the el ements of an iterable. (489:57 - 490:10)
- When you reach the end of an iterable, a `StopIteration` exception is raised, which you can handle using a `try-excep t` block.

Creating Custom Iterators

- You can create your own custom iterator by overriding the `__iter__` and `__next__` methods in a class.
- This allows you to define the behavior of the iterator, such as the range of values it should return.

Generators

- Generators are a special type of function that use the `yield` statement instead of `return` to create iterative functions.
- Generators are memory-efficient and can generate infinite sequences, making them useful for processing large dat a sets.
- Generators can be used to create complex processing pipelines by breaking down the logic into smaller, reusable pi eces.

Scope in Python

- Local scope: Variables defined within a function are only accessible within that function.
- Enclosed scope: Variables from an outer function can be accessed by an inner function.
- Global scope: Variables defined outside of any function can be accessed from anywhere in the code.
- Built-in scope: Python's built-in functions, modules, and objects are available throughout the code.

Python Modules

- Modules are Python files with a `.py` extension that contain reusable code, such as functions and variables.
- You can import modules using various syntax options, such as `import module_name`, `from module_name import function_name`, or `import module_name as alias.`
- Python also has built-in modules, such as `math`, `datetime`, and `random` which provide a wide range of utility functions.

Map, Filter, and Reduce Functions

Map

- The `map()`function applies a given function to each item in an iterable and returns an iterator with the transformed values. (554:05 - 554:28)
- You can use both custom functions and lambda functions with `map().`

Filter

- The `filter()`function creates a new iterable containing only the elements for which a given function returns `True.`
- You can use both custom functions and lambda functions with `filter().`

Reduce

- The `reduce()`function applies a function of two arguments cumulatively to the elements of a sequence, from left to right, to reduce the sequence to a single value. (552:46 - 553:12)
- You need to import the `functools` module to use the `reduce()` function.

Function	Description	Syntax
`map()`	Applies a function to each item in an iterable	`map(function, iterable)`

`filter()`	Creates a new iterable with elements that pass a given condition	`filter(function, iterable)`
`reduce()`	Applies a function of two arguments cumulatively to the elements of a sequence	`reduce(function, sequence)`