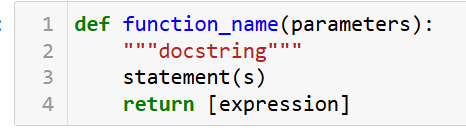
**Functions:**

**Definition**:A function is a structured, reusable code block utilized for carrying out a specific task. Functions facilitate the creation of modular code, simplifying maintenance and reusability.

**Syntax**:



**Components of a Function:**

**Name of the Function:** Describes the function and should be meaningful.

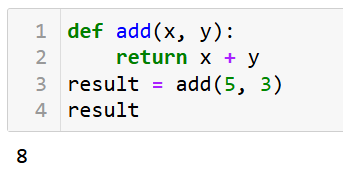
**Parameters (Arguments)**: Variables passed into the function. Functions can have multiple parameters or none at all.

**Docstring**: A string that contains information about the purpose and behavior of the function.

**Code Bloc**k: Contains the actual code to be executed within the function.

**Return Statement**: Determines the value that the function will return. If not included, the function defaults to returning None.

Example:



**Iterators and Generators:**

**Iterators:**

**Definition:** An iterator is an object that allows traversal through a countable number of values. It is used to iterate over iterable objects like lists, tuples, sets, etc

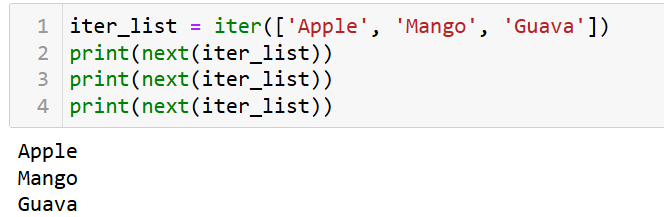
**Key Points**:

**Lazy Evaluation**:

* **On-Demand Evaluation**: Values are generated only when requested, avoiding the need for storing all elements at once.
* **Memory Efficiency**: Only one memory location is used for processing values, reducing RAM usage, especially beneficial for large datasets.
* **No Repeated Evaluation**: The expression's evaluation is stored in memory until the item is explicitly called.

Using an iterator-

* **iter()** function is used to create an iterator containing an iterable object.
* **next(**) function is used to call the next element in the iterable object.
* After the iterable object is completed, to use them again reassign them to the same object.



**Generators:**

**Definition**: Generators are a simple way to create iterators using functions and the yield keyword instead of returning values.

**Key Points**:

* **yield Keyword**:
  + **Yield vs. Return**: Unlike the return statement, which exits the function, yield returns a value and pauses the function’s state, allowing it to resume from where it left off.
  + **Multiple Yields**: Functions can yield multiple values over time, producing a sequence of results.

**Lazy Evaluation**:

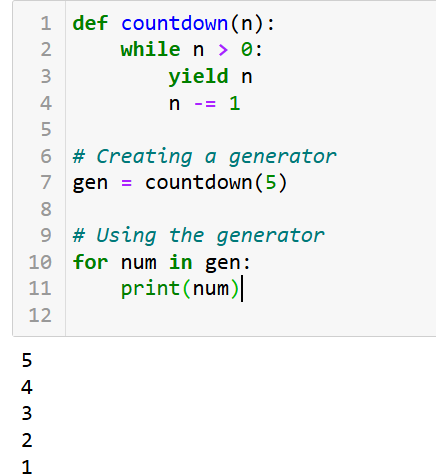
* **On-Demand Data**: Generators produce values one at a time and only when required, avoiding the need to generate or store all values upfront.
* **Memory Efficiency**: Since values are generated as needed and not all at once, generators use minimal memory.

**Characteristics**:

* **Iterables**: Generators return data in an iterable format, allowing the sequence to be iterated over, much like a list or tuple.

**Advantages**:

* **Efficient Memory Usage**: Generators do not store the entire sequence in memory, making them suitable for large datasets or infinite sequences.
* **Simpler Syntax**: Using yield simplifies the creation of iterators, making the code more concise and readable.



**Lambda , map, reduce and filter:**

**Lambda Functions:**

**Definition**: Lambda functions are small, anonymous functions defined using the lambda keyword. They are used for creating small, throwaway functions without the need to formally define a function using def

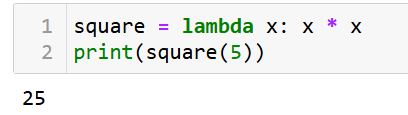
**Key Points**:

* **Anonymous**: Lambda functions are not bound to a name.
* **Single Expression**: They can contain only one expression, which is evaluated and returned.

**Syntax**:

***lambda arguments: expression***

**Example**:



**Map Functions:**

**Definition**: The map function applies a given function to all items in an input iterable (like a list) and returns an iterator with the results.

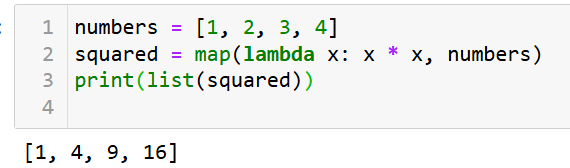
**Key Points**:

* **Transformation**: Used to transform each item in an iterable by applying the specified function.

**Syntax**:

***map(function, iterable)***

Example:



**Reduce Functions:**

**Definition:**The reduce function from the functools module applies a given function cumulatively to the items of a sequence, from left to right, to reduce the sequence to a single value.

**Key Points**:

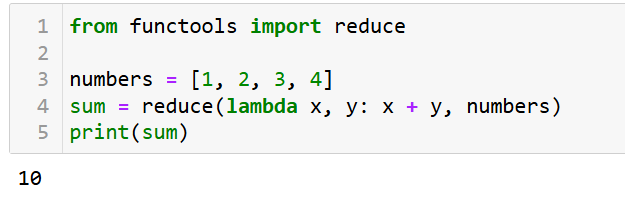
* **Cumulative Operation**: It reduces an iterable to a single cumulative value by applying the function cumulatively.

**Syntax**:

***from functools import reduce***

***reduce(function, iterable)***

Example:



**Filter Functions:**

**Definition:**The filter function constructs an iterator from elements of an iterable for which a specified function returns True.

**Key Points**:

* **Selection**: Used for filtering elements from an iterable based on a condition defined by the function.

**Syntax**:

***filter(function, iterable)***

Example:

