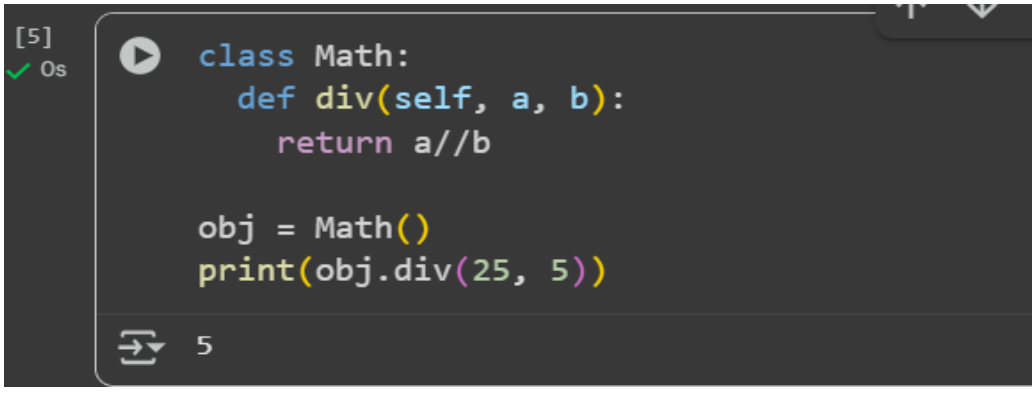
Functions

1. What is the difference between a function and a method in Python?

Function → A block of reusable code defined with **def** or **lambda**. It stands alone and can be called independently.
For example,

Method → A function that is associated with an **object** (part of a class). It's called using object.method(), and usually operates on that object's data.
For example,



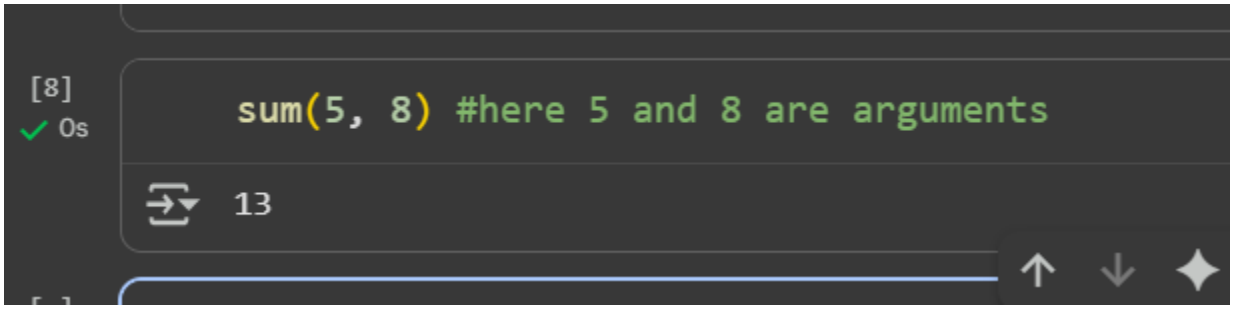
2. Explain the concept of function arguments and parameters in Python.

Function Parameters:

Parameters are the placeholders specified inside the parentheses of a function definition.Each parameter represents a variable that the function intends to work with.
For example,

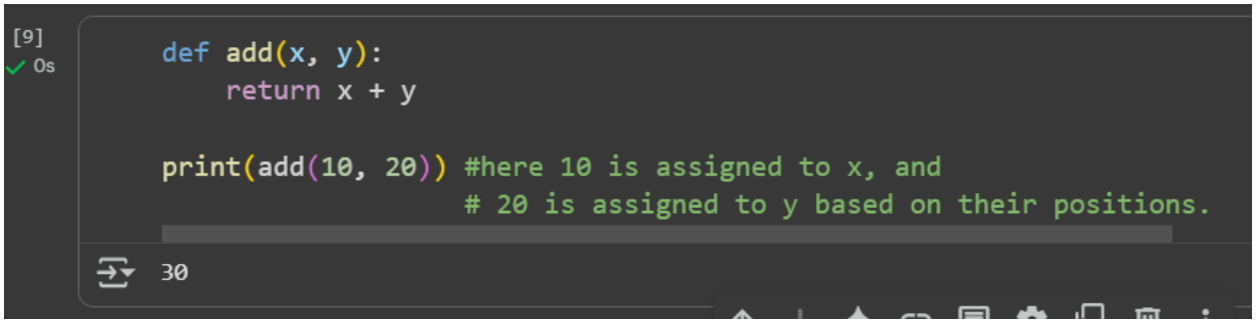
Function Arguments:

Arguments are the actual values passed to the function when it is called. They fill in the parameters defined in the function, providing the specific data the function will use.
For example,

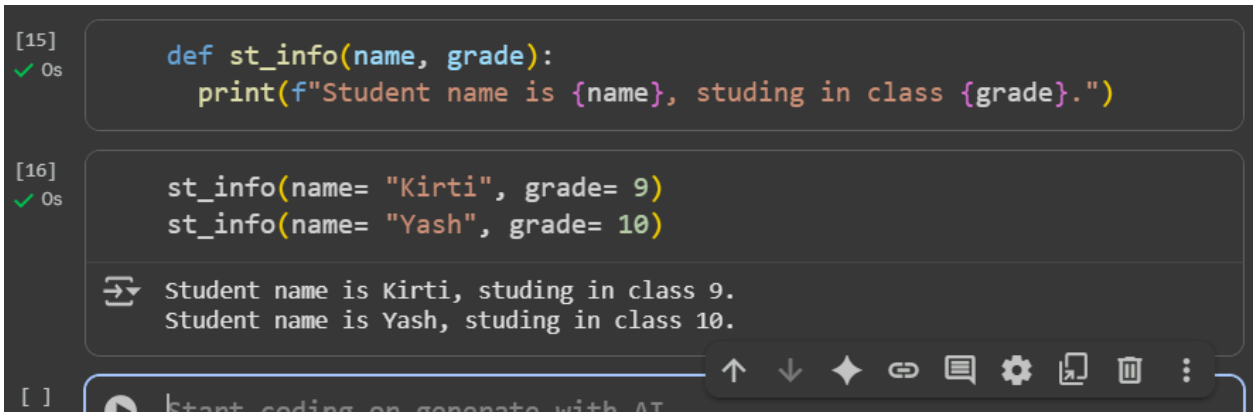


Types of Arguments:

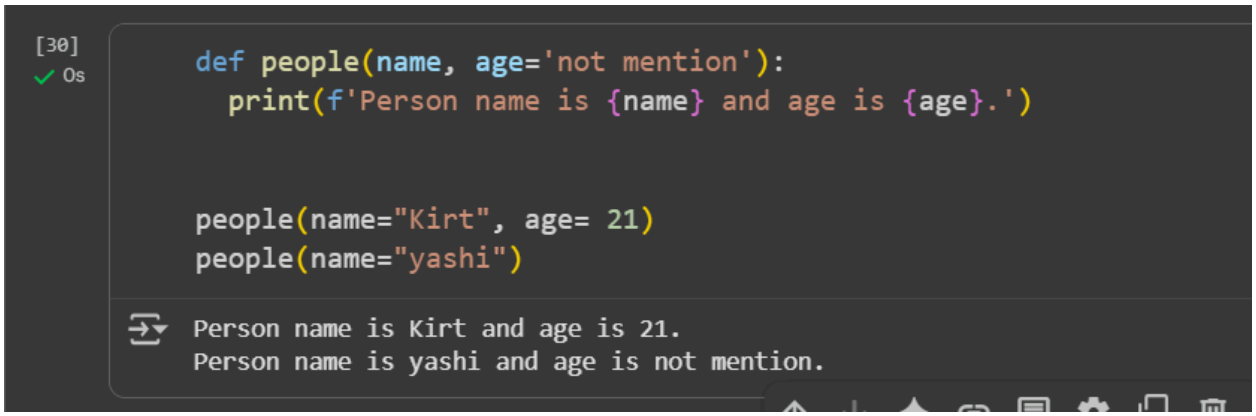
1. Positional Arguments: It's mapped to parameters by position/order.
For example,



2. Keyword Arguments: Explicitly assign values to parameters using param=value syntax, so order doesn't matter.
For example,



3. Default Arguments: Parameters have default values, so arguments can be omitted during the call.



4. Arbitrary Positional Arguments (*args):
Allows passing a variable number of positional arguments.
For example,

5. Arbitrary Keywords Arguments (*kwargs):

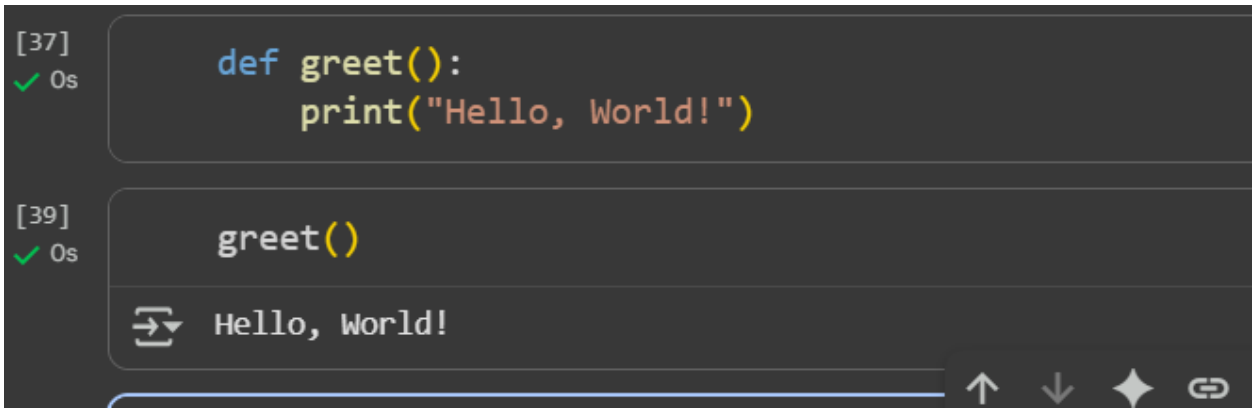
Allows passing a variable number of keyword arguments.

For example,

3. What are the different ways to define and call a function in Python?

There are several ways to define and call functions in Python. The most common methods are :

1. Standard Function Definition Using def :
Functions are defined using the def keyword, followed by the function name, parentheses (which may include parameters), a colon, and an indented block of code (the function body).
For example,



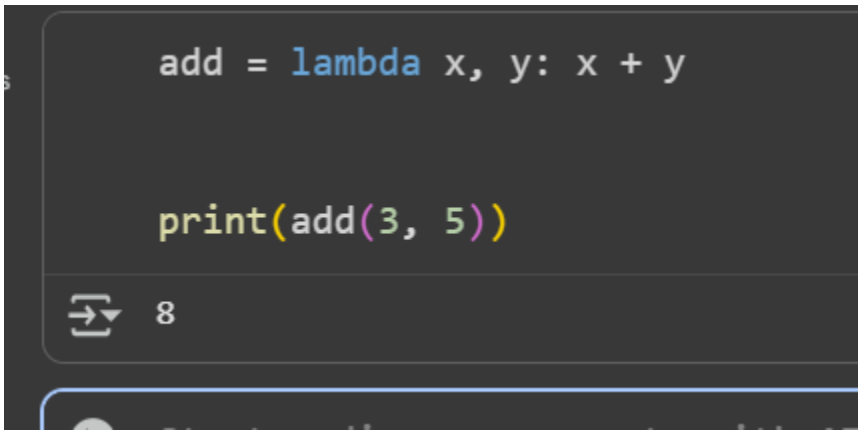
2. Function with Parameters: Functions can accept inputs via parameters, allowing them to operate on different values.

3. Function with Default Parameters:

Parameters can have default values, so callers can omit those arguments.

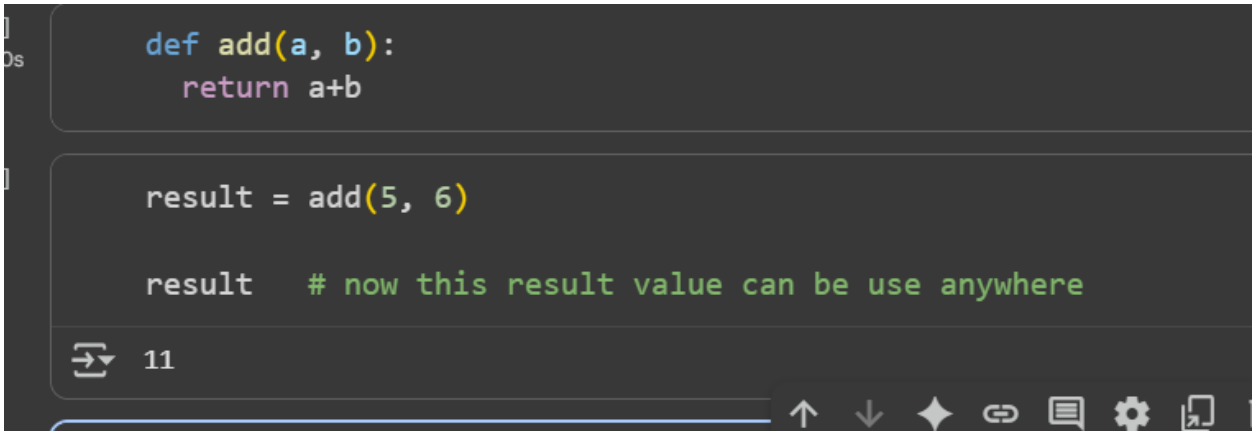
4. Lambda Functions (Anonymous Functions):

Python supports small anonymous functions with the lambda keyword, typically used for short, simple operations.



4. What is the purpose of the `return` statement in a Python function?

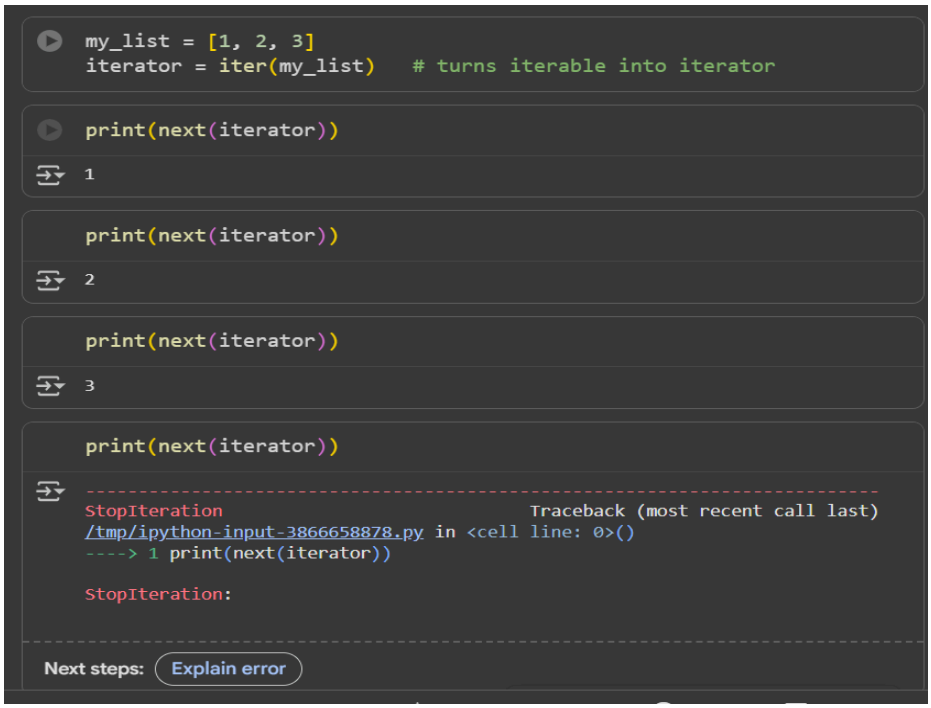
The purpose of the **'return'** statement in a Python function is that it provides output (a result) that can be stored in a variable or used directly in expressions.
For example,



5. What are iterators in Python and how do they differ from iterables?

An **Iterator** is an object that produces the next value of an iterable on demand. It remembers its current position and moves step by step until elements are exhausted.

For example,

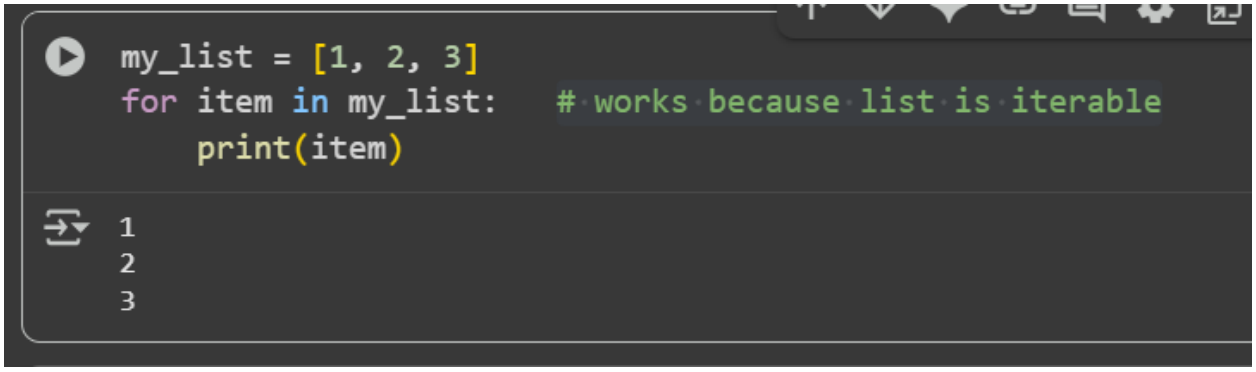


It implements two methods:

1. iter( ) → returns the iterator object itself.

2. next( ) → returns the next element; raises **StopIteration** when no elements are left.

While an **Iterable** is any Python object that can return its elements one by one when asked like list, tuple, string, set, dict, etc.



6. Explain the concept of generators in Python and how they are defined.

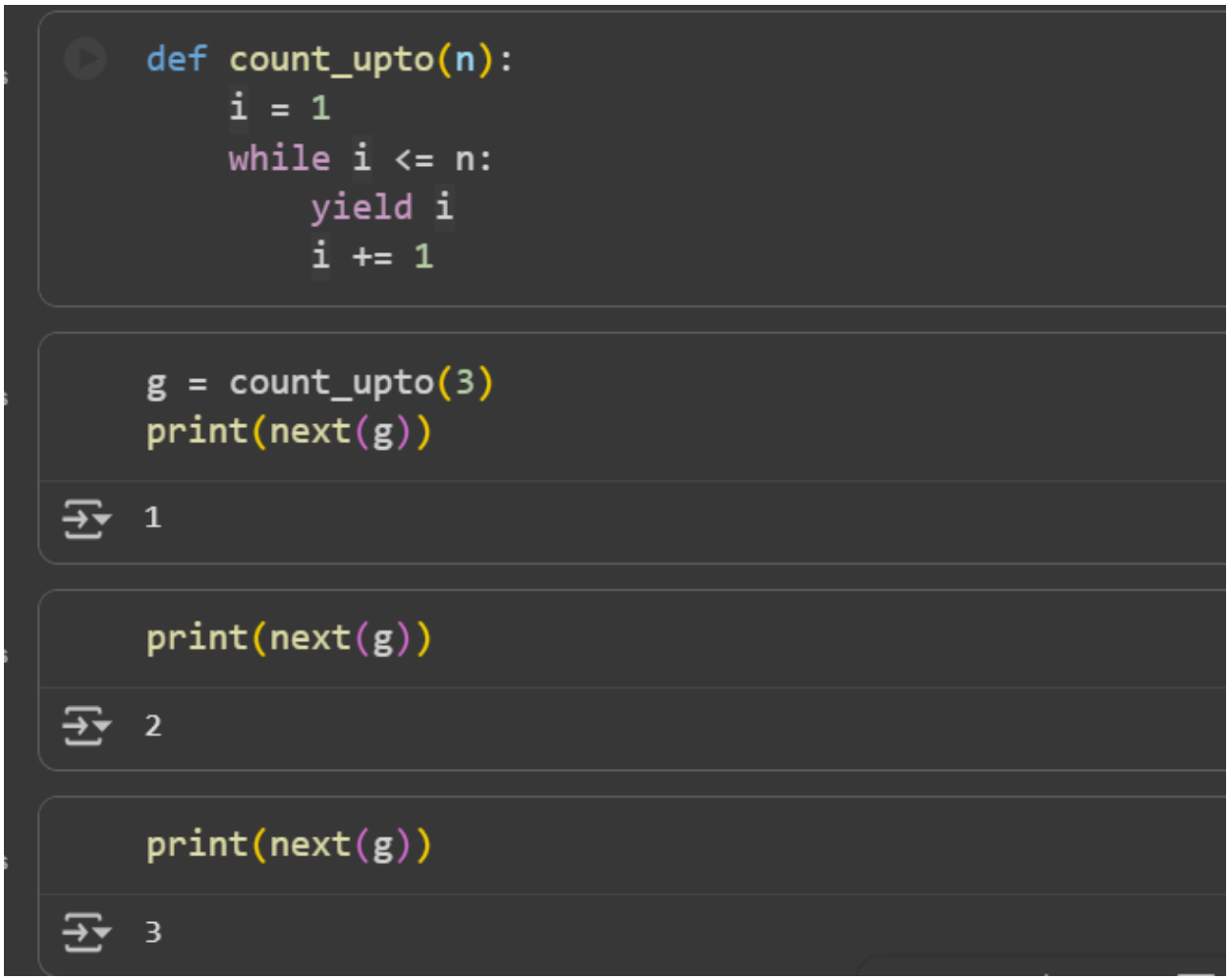
Generators are a special kind of iterator in Python. They allow you to produce values one at a time instead of creating and storing the entire sequence in memory. Useful when dealing with large data sets or infinite sequences.

There are two ways to define Generators:

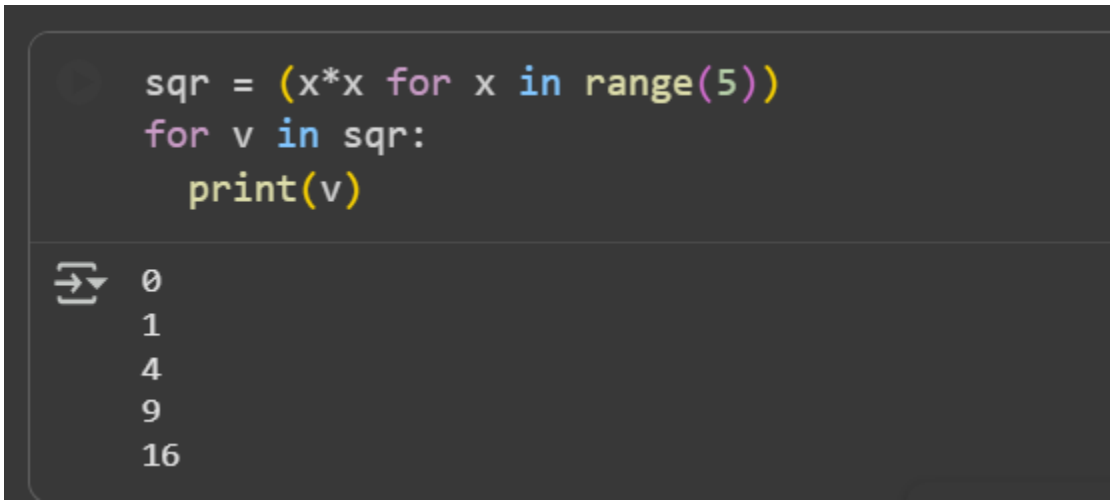
1. Generator Functions :

These are regular functions but use the yield keyword to produce a value and temporarily suspend execution. When resumed, they pick up where they left off.

For example,



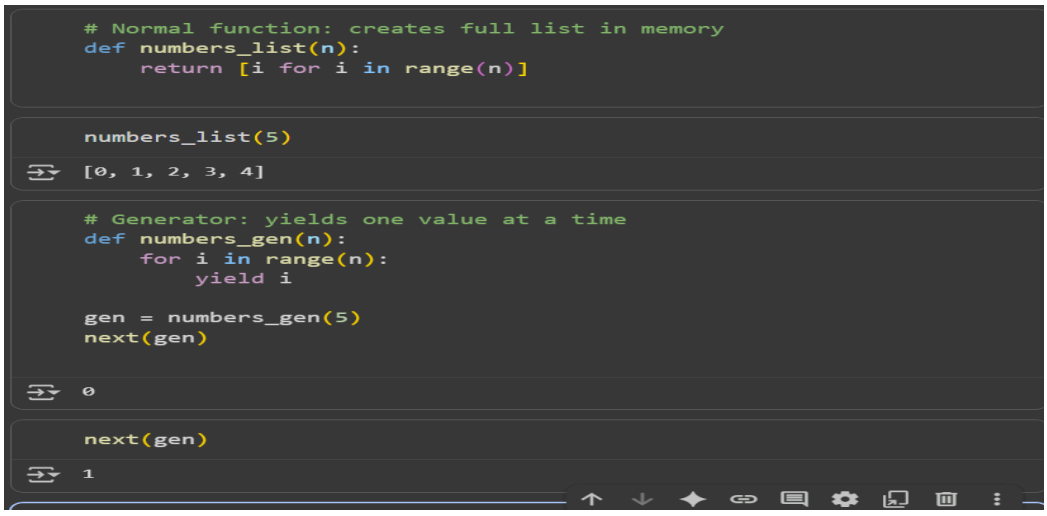
2. Generator Expressions:
These are similar to list comprehensions but use parentheses () instead of square brackets []. They create generator objects instead of lists.
For example,



7. What are the advantages of using generators over regular functions?

The advantages of using generators over regular functions are:

1. Generators don't store the entire result in memory. They produce values one at a time.
For example,



2. Values are computed only when needed, not in advance in the generators.

3. Regular functions can't return infinite sequences while Generators can represent infinite streams because they yield step by step.

4. Faster for streaming data (files, network, logs) since they don't create large intermediate structures.

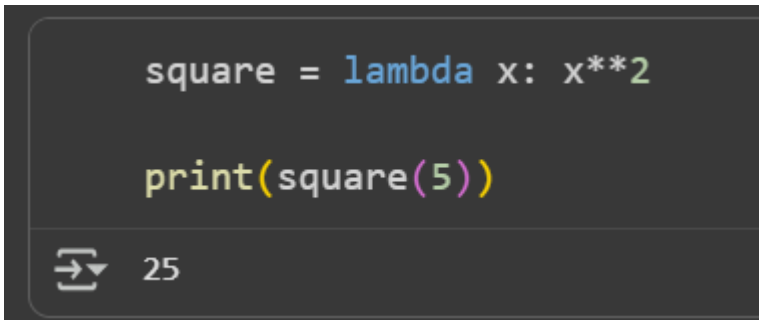
5. Yield makes iterator logic concise.

8. What is a lambda function in Python and when is it typically used?

A **lambda function** in Python is a small, anonymous function defined using the **lambda** keyword instead of **def.**

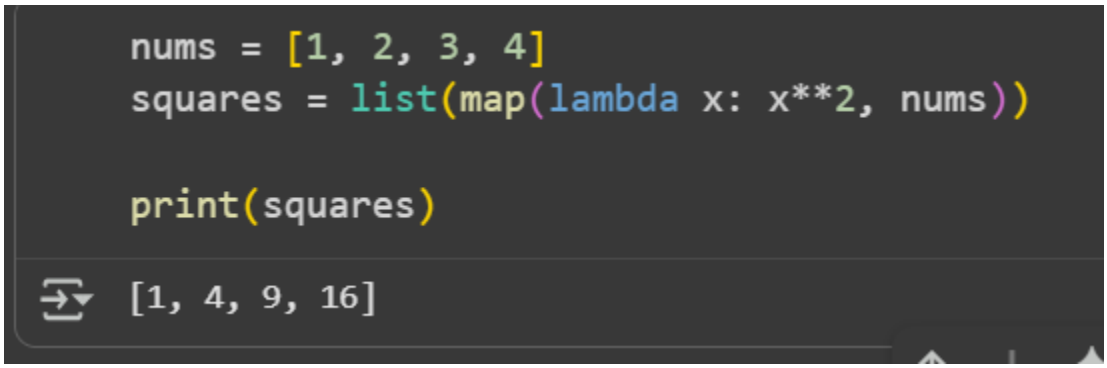
Syntax: lambda arguments: expression → It takes any number of arguments but only one expression, which is evaluated and returned.

For example,

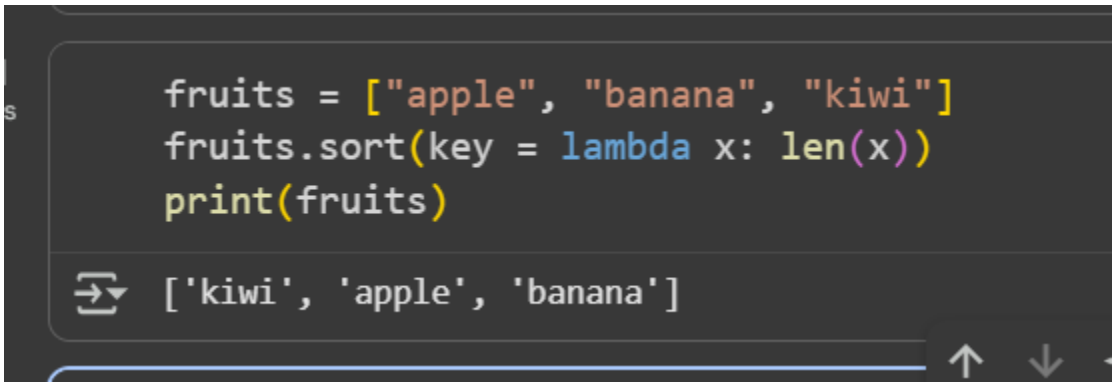


Lambda functions are usually used for quick, short-term operations where a full function definition would be unnecessary.
Common use cases:

1. Inside built-in functions like map(), filter(), reduce().
Passing as arguments to higher-order functions like map(), filter(), and reduce() to perform inline operations.
For example,



2. Sorting with a key function:



3. Filtering items:

```
nums = [10, 15, 20, 25, 30]

evens = list(filter(lambda x: x % 2 == 0, nums))

print(evens)
```

```
[10, 20, 30]
```

4. When function definition is not worth it:
Lambda are ideal for concise, one-time-use functions in places where defining a regular function would be overkill.

9. Explain the purpose and usage of the `map()` function in Python.

The purpose of **map() function** in Python is to apply a given function to every item in an iterable and return a map object.
Syntax: map(function, iterable)

The usage are:
1. Using as a Normal Function

For example,

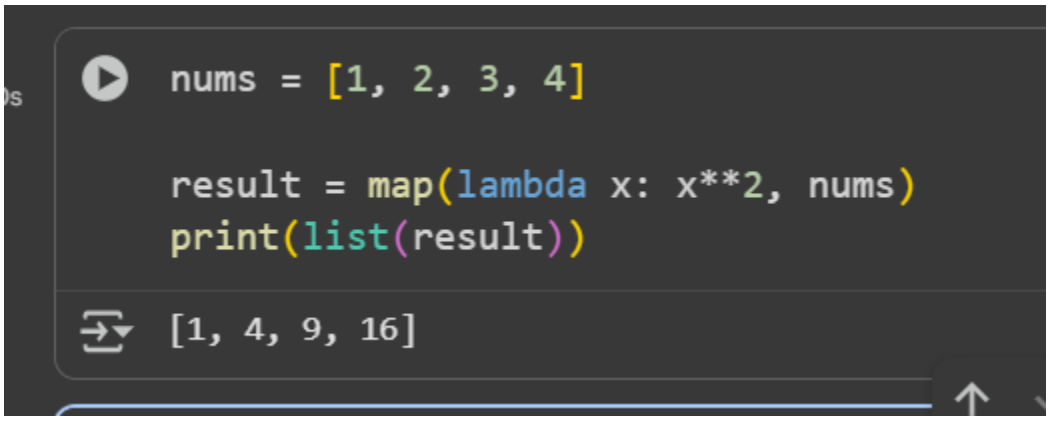
```
def square(x):
    return x**2

nums = [1, 2, 3, 4]
result = map(square, nums)
print(list(result))
```

```
[1, 4, 9, 16]
```

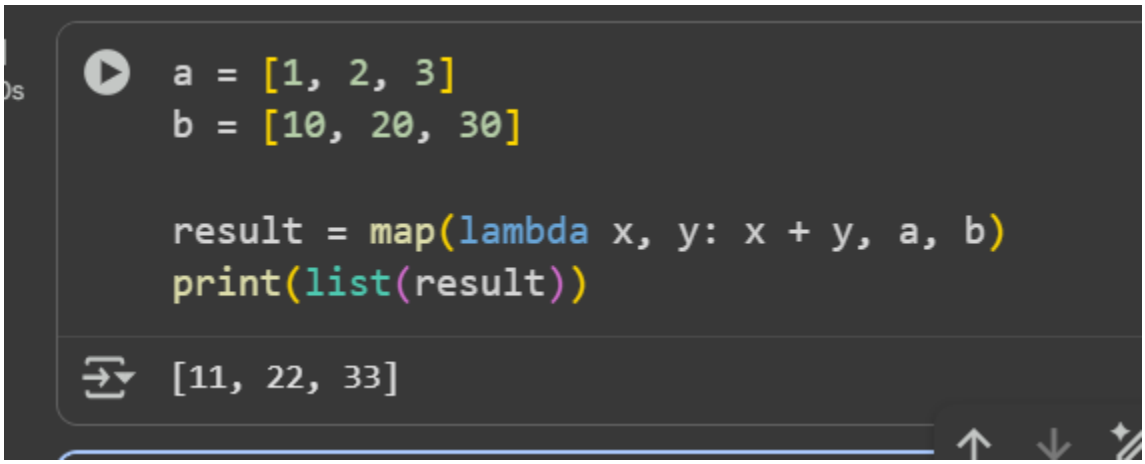
2. Using lambda

For example,



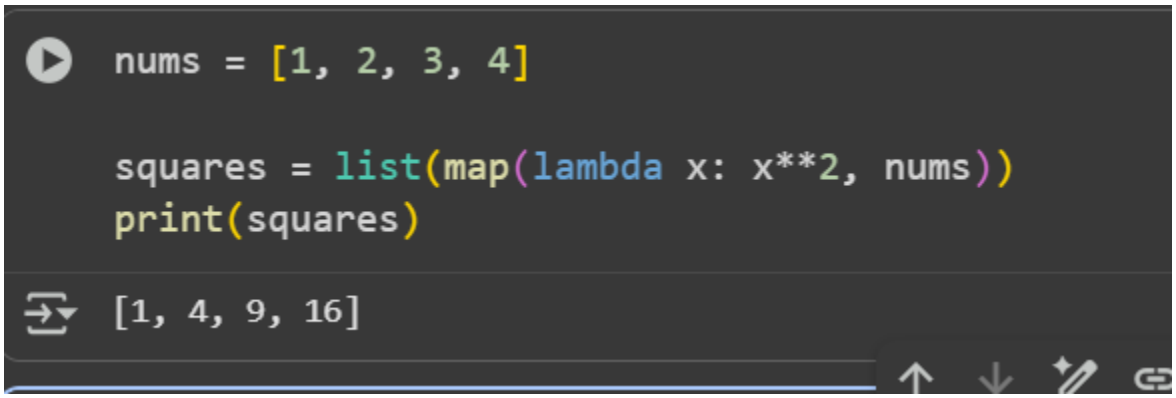
3. With Multiple Iterables

If you pass more than one iterable, map( ) applies the function in parallel (like zipping them). For example,

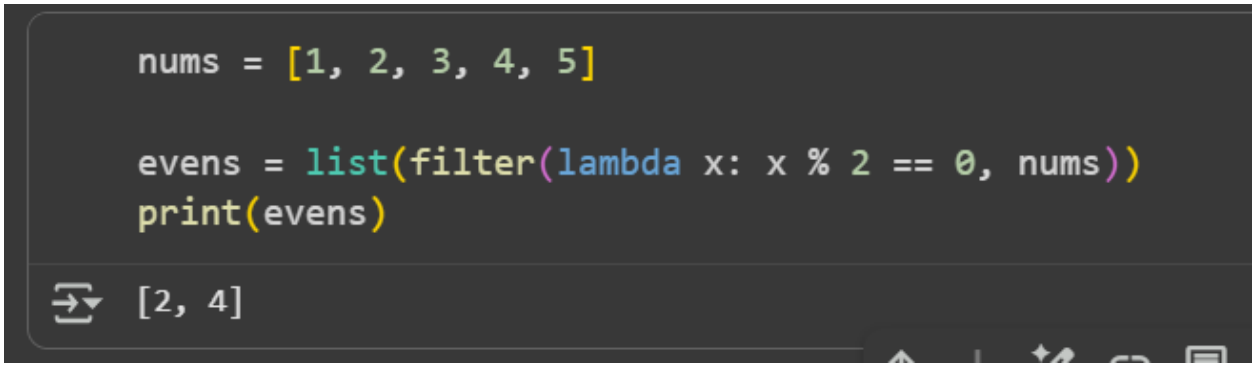


10. What is the difference between `map()`, `reduce()`, and `filter()` functions in Python?

map() : It applies a function to each element of an iterable. As an output a new iterable (map object) with transformed elements.
For example,



filter() : It selects elements from an iterable based on a condition (True/False). As an iterable (filter object) with only the elements that pass the text.
For example,



reduce() : It repeatedly applies a function to pairs of elements, reducing the iterable to a single value. As a result one final result comes.
For example,

11. Using pen & Paper write the internal mechanism for sum operation using reduce function on this given list:[47,11,42,13].