**Lambda Functions**

* A **lambda function** is a small, quick function that we write in one line. It's like a shortcut for creating simple functions without giving them a name.
* You use lambda when we need a short function for a quick task.
* When we need a simple, one-time function.
* Often used with functions like map(), filter(), or sorted().

# Regular function

def add(a, b):

return a + b

print(add(5, 3)) # Output: 8

# Lambda function (same as above)

add\_lambda = lambda a, b: a + b

print(add\_lambda(5, 3)) # Output: 8

**Example with filter**

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

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print(even\_numbers) # Output: [2, 4]

**Regular Functions**

* A **function** is a block of reusable code. We give it a name and call it whenever we need that functionality.
* Functions can do anything from calculations to printing messages.
* To avoid repeating code.
* To organize our program into smaller parts.

# Define a function

def greet(name):

return f"Hello, {name}!"

# Call the function

print(greet("Harini")) # Output: Hello, Harini!

**Loops**

Loops are used to repeat actions. There are two main types of loops in Python:

1. **For Loop**: Use this when we know the exact number of repetitions or are working with a collection (like a list or dictionary).

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

# Output:

# apple

# banana

# cherry

**2.While Loop**: Use this when you want to repeat something **until** a condition is false.

count = 0

while count < 3:

print("Counting:", count)

count += 1

# Output:

# Counting: 0

# Counting: 1

# Counting: 2

**Break and Continue**:

* break: Stops the loop entirely.

for num in range(5):

if num == 3:

break

print(num) # Output: 0 1 2

* continue: Skips the current iteration and moves to the next.

for num in range(5):

if num == 3:

continue

print(num) # Output: 0 1 2 4

**Django**

* **What is Django?**
  + Django is like a **toolbox for web development**. It helps us to build websites quickly without worrying too much about low-level details like connecting to databases or handling user authentication.
  + It’s great for big projects because it provides a lot of built-in features.
* **Why Django?**
  + Ready-made features like user login, forms, and database management.
  + Follows the **"Don’t Repeat Yourself" (DRY)** principle, so we write less repetitive code.

**Flask**

* **What is Flask?**
  + Flask is like a **minimalist web development kit**. It gives us the basics to build a website and lets us to add features only when you need them.
  + It’s lightweight and ideal for small or custom projects.
* **Why Flask?**
  + Gives more control because it doesn’t include unnecessary features by default.
  + Easy to learn and use for beginners.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Django** | **Flask** |
| **Complexity** | More features, suitable for big apps | Lightweight, flexible |
| **Learning** | Steeper because of built-in tools | Easier for beginners |
| **Use case** | Large-scale apps with many features | Small, custom projects |