**🎯 Topic: Functions in Python**

## **✅ 1. What is a Function?**

A function is a **block of reusable code** that performs a specific task.  
 It lets you **write once, use many times**.

📦 Think of it like a **machine**: You give input, it gives output.

## **✅ 2. Defining a Function (with def)**

### **🔧 Syntax:**

def function\_name():

# code block

### **📘 Example:**

def greet():

print("Hello, welcome to Python!")

greet() # Calling the function

🧠 Output:

Hello, welcome to Python!

## **✅ 3. Function with Parameters (Passing Inputs)**

### **📘 Example:**

def greet(name):

print(f"Hello {name}, welcome!")

greet("Gowtham") # Output: Hello Gowtham, welcome!

✅ This helps make your function **dynamic**.

## **✅ 4. Return Values from Functions**

A function can **return a result** using the return keyword.

### **📘 Example:**

def add(a, b):

return a + b

result = add(5, 3)

print("Sum:", result)

🧠 Output:

Sum: 8

✅ You can **store** and **reuse** the result.

## **✅ 5. \*args → Accept Multiple Positional Arguments**

\*args allows you to pass **any number of values** into a function.

### **📘 Example:**

def add\_all(\*args):

total = 0

for num in args:

total += num

return total

print(add\_all(1, 2, 3, 4)) # Output: 10

🧠 Internally, args is a **tuple**.

## **✅ 6. \*\*kwargs → Accept Multiple Keyword Arguments**

\*\*kwargs allows passing **named arguments** (like key=value pairs).

### **📘 Example:**

def print\_info(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

print\_info(name="Gowtham", age=30)

🧠 Internally, kwargs is a **dictionary**.

🖨 Output:

name: Gowtham

age: 30

## **✅ Real-Life Example: Profile Generator**

### **📘 Code:**

def create\_profile(\*\*kwargs):

print("User Profile:")

for key, value in kwargs.items():

print(f"{key.capitalize()}: {value}")

create\_profile(name="Nandini", age=28, city="Madurai", profession="Designer")

## **✅ 7. Default Parameter Values**

You can assign default values to parameters.

### **📘 Example:**

def greet(name="Guest"):

print(f"Hello, {name}!")

greet() # Hello, Guest!

greet("Nila") # Hello, Nila!

## **🧠 Summary Table:**

| **Concept** | **Description** | **Syntax Example** |
| --- | --- | --- |
| def | Define a function | def greet(): |
| () | Call the function | greet() |
| return | Send result back from function | return a + b |
| \*args | Multiple unnamed values | def fun(\*args): |
| \*\*kwargs | Multiple named values | def fun(\*\*kwargs): |
| default | Pre-filled values if nothing passed | def greet(name="Guest"): |

## **🧪 Mini Project Idea (for your video)**

### **🎯 Bill Calculator:**

def calculate\_bill(\*items):

total = sum(items)

return total

def show\_user(\*\*details):

print("Customer Info:")

for k, v in details.items():

print(f"{k}: {v}")

show\_user(name="Rahul", city="Chennai")

print("Total Bill:", calculate\_bill(100, 250, 75))

## **✅ What does return do in a function?**

The return statement is used to **send data (output) back** from a function to the place where it was called.

### **📘 Example 1: Using return**

def add(a, b):

return a + b

result = add(10, 5)

print("Result is:", result)

🧠 What happens here:

* return a + b → sends the result (15) back to the caller
* You store it in result and can use it later

## **❓ Why not just use print()?**

Because:

* **print() only shows the output** on the screen
* **It does NOT give the value back to the code**

### **📘 Example 2: Using print() only**

def add(a, b):

print(a + b)

x = add(10, 5)

print("x:", x)

🖨 Output:

15

x: None

⚠️ Because print() just prints and returns **nothing** (None)

## **✅ Key Differences:**

| **Feature** | **return** | **print()** |
| --- | --- | --- |
| Gives value | ✅ Yes, sends output to the caller | ❌ No, just displays on screen |
| Can reuse | ✅ Yes (store, calculate, compare) | ❌ No |
| For real apps | ✅ Must use return | 🚫 Only for debugging/output |

### **🎯 Think of it like this:**

| **Concept** | **Example** |
| --- | --- |
| return | Giving a **gift back** (you can use it) |
| print() | Just **showing the gift** (can’t use it again) |

### **✅ When to use return?**

* Calculators
* API results
* Data processing
* Any function where you **need to use the result again**

## **🧼 What is a Pure Function?**

A **pure function** is a function that:

1. **Always returns the same output** for the same input
2. **Does not change anything outside itself** (no side effects)

🧠 It’s like a **math formula**:  
 Input goes in → Output comes out → That’s it!  
 No print, no database updates, no file writes.

## **✅ Pure Function Example:**

def add(a, b):

return a + b

💡 Every time you call add(2, 3), it **always gives 5**.  
 It doesn't touch anything outside the function.

## **❌ Impure (Normal) Function Example:**

total = 0

def add\_to\_total(amount):

global total

total += amount

print("Total is:", total)

😵 This is **not a pure function** because:

* It **modifies a global variable** (total)
* It **prints** (causes a side effect)

Even if you give the same input, the output will **change every time** based on total.

## **🎯 Summary: Pure vs Normal (Impure) Function**

| **Feature** | **Pure Function** | **Impure Function (Normal)** |
| --- | --- | --- |
| Same input = same output | ✅ Yes | ❌ Not always |
| Changes outside data | ❌ Never | ✅ Can modify global or external |
| Side effects | ❌ None | ✅ Can print, write, update, etc |
| Easy to test/debug | ✅ Yes | ❌ Harder |

## **🧠 When to Use Pure Functions?**

* ✅ In **data processing**
* ✅ For **predictable, reusable code**
* ✅ In **functional programming** and **unit testing**

## **✅ Difference Between Function and Method**

| **Feature** | **Function** | **Method** |
| --- | --- | --- |
| Definition | A block of code that **performs a task** | A function that is **associated with an object or class** |
| Called with | Just by name (e.g. add(5, 3)) | With object or class (e.g. name.upper()) |
| Belongs to | **Independent** (not tied to a class) | **Belongs to an object/class** |
| Defined using | def keyword | Defined using def **inside a class** |
| Example | def greet(): | def greet(self): inside a class |

### **About the Author**

**Gowtham SB** is a **Data Engineering expert, educator,** **and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

𝐒𝐨𝐜𝐢𝐚𝐥𝐬

🎥𝐘𝐨𝐮𝐓𝐮𝐛𝐞 - https://www.youtube.com/@dataengineeringvideos

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - <https://instagram.com/dataengineeringtamil>

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - [https://instagram.com/](https://instagram.com/dataengineeringtamil)thedatatech.in

🤝𝐂𝐨𝐧𝐧𝐞𝐜𝐭 𝐟𝐨𝐫 𝟏:𝟏 - https://topmate.io/dataengineering/

💼𝐋𝐢𝐧𝐤𝐞𝐝𝐈𝐧 - https://www.linkedin.com/in/sbgowtham/

🌐𝐖𝐞𝐛𝐬𝐢𝐭𝐞 - https://codewithgowtham.blogspot.com

💻𝐆𝐢𝐭𝐇𝐮𝐛 - http://github.com/Gowthamdataengineer

💬𝐖𝐡𝐚𝐭𝐬 𝐀𝐩𝐩 - https://lnkd.in/g5JrHw8q

📧𝐄𝐦𝐚𝐢𝐥 - atozknowledge.com@gmail.com

📱𝐀𝐥𝐥 𝐌𝐲 𝐒𝐨𝐜𝐢𝐚𝐥𝐬 - <https://lnkd.in/gf8k3aCH>