# Level 3: Data Structures

In Python, data structures are used to store and manage data.  
We’ll cover:

1. Strings
2. Lists
3. Tuples
4. Sets
5. Dictionaries

## 1. **Strings**

A string is a sequence of characters.

### Example:

name = "Vijay"

print(name[0]) # V (indexing)

print(name[-1]) # y (last char)

print(name[1:4]) # ija (slicing)

print(len(name)) # 5

print(name.upper()) # VIJAY

print(name.lower()) # vijay

print(name.replace("V", "R")) # Rijay

## 2. **Lists**

A list stores multiple items (mutable).

### Example:

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

print(fruits[0]) # apple

fruits.append("orange")

print(fruits) # ['apple', 'banana', 'cherry', 'orange']

fruits.remove("banana")

print(fruits) # ['apple', 'cherry', 'orange']

print(len(fruits)) # 3

## 3. **Tuples**

Like lists, but immutable (cannot change after creation).

### Example:

colors = ("red", "green", "blue")

print(colors[0]) # red

print(len(colors)) # 3

# colors[1] = "yellow" ❌ (error: cannot modify)

## 4. **Sets**

Stores unique items, unordered, no duplicates.

### Example:

numbers = {1, 2, 3, 3, 2, 4}

print(numbers) # {1, 2, 3, 4} (duplicates removed)

numbers.add(5)

print(numbers) # {1, 2, 3, 4, 5}

## 5. **Dictionaries**

Stores data in key-value pairs.

### Example:

student = {

"name": "Vijay",

"age": 28,

"city": "Nellore"

}

print(student["name"]) # Vijay

student["age"] = 29 # update

student["course"] = "Python" # add new

print(student)

Exercises for You

1. Create a string "Python Programming" and:
   * Print first 6 characters.
   * Convert to uppercase.
   * Replace "Python" with "Java".
2. Create a list of 5 cities and:
   * Add one new city.
   * Remove one city.
   * Print the final list.
3. Create a tuple of 3 subjects and print the second subject.
4. Create a set of numbers with duplicates, and show that duplicates are removed.
5. Create a dictionary for your profile with keys: name, age, city, skill. Print it.

### ✅ Task 1 (String operations)

language = "Python Programming"

print(language[0:6]) # prints first 6 chars → Python

lang\_upper = language.upper() # converts to UPPERCASE → PYTHON PROGRAMMING

repl = language.replace('Python', 'Java') # replaces Python with Java

print(repl) # Java Programming

### ✅ Task 2 (List operations)

cities = ['Nellore', 'Vijayawada', 'Tirupati', 'Visakhapatnam', "Guntur"]

cities.append('kakinada') # adds new city

cities.remove('Tirupati') # removes Tirupati

print(cities)

# Output: ['Nellore', 'Vijayawada', 'Visakhapatnam', 'Guntur', 'kakinada']

### ✅ Task 3 (Tuple operations)

subjects = ('english', 'telugu', 'kannada')

print(subjects[1]) # prints 'telugu' (index starts at 0)

### ✅ Task 4 (Set operations)

numbers = {1,23, 3, 3, 5, 6}

print(numbers)

# Output: {1, 3, 5, 6, 23} (duplicates like "3" are automatically removed)

### ✅ Task 5 (Dictionary operations)

details = {'name': 'Vijay', 'age': 28, 'skill': ['python', 'scala']}

print(details['skill'][0]) # prints 'python'

### 🔹 Problem Statement:

You are building a system that decides discounts based on a customer's age:

* If age is less than 18 → "You get 20% discount"
* If age is between 18 and 60 → "You get 10% discount"
* If age is greater than or equal to 60 → "You get 30% discount"

### Example Code:

age = 65

if age < 18:

print("You get 20% discount")

elif age < 60:

print("You get 10% discount")

else:

print("You get 30% discount")

### 🔹 Why Functions?

Functions help you group reusable code. Instead of writing the same logic again and again, you write it once inside a function and call it whenever needed.

### ✅ Simple Example:

def greet(name):

return f"Hello, {name}!"

print(greet("Vijay"))

print(greet("Aroha"))

Output:

Hello, Vijay!

Hello, Aroha!

### 🔹 Real-Time Problem Statement:

You are working in a banking project.  
You need a function that calculates Simple Interest given principal, rate, and time.

Formula:

Simple Interest = (Principal \* Rate \* Time) / 100

### ✅ Example Code:

def simple\_interest(principal, rate, time):

return (principal \* rate \* time) / 100

print(simple\_interest(10000, 5, 2)) # 1000

👉 Do you want me to take you step by step from basic to advanced functions (parameters, return values, default args, lambda, recursion, etc.), or jump straight into real-time function problems?

##### You said:

step by step from basic to advanced functions

###### ChatGPT said:

Perfect Vijay 👌 we’ll go step by step into Functions, starting from the basics and moving towards advanced concepts.

Here’s the roadmap for Functions:

1. Defining and Calling Functions
2. Function Parameters
   * Positional Arguments
   * Keyword Arguments
   * Default Arguments
   * Variable-Length Arguments (\*args, \*\*kwargs)
3. Return Statement
4. Scope (Local & Global Variables)
5. Nested Functions
6. Lambda (Anonymous Functions)
7. Recursion
8. Built-in Higher Order Functions (map, filter, reduce)
9. Decorators (Advanced)

### 🔹 Step 1: Defining and Calling a Function

A function is defined using the def keyword.

✅ Example:

def greet():

print("Hello, Welcome to Python Functions!")

# Calling the function

greet()

Output:

Hello, Welcome to Python Functions!

👉 This is the simplest function:

* No parameters
* No return value

Would you like me to continue to Step 2 (Function Parameters), or do you want me to give a few practice problems for Step 1 first?

##### You said:

step2

###### ChatGPT said:

Great 👍 Let’s move to Step 2: Function Parameters

Parameters allow us to pass data into functions so they can work dynamically.

### 🔹 2.1 Positional Arguments

Arguments are passed in the same order as parameters.

✅ Example:

def add(a, b):

print("Sum:", a + b)

add(10, 20) # a=10, b=20

Output:

Sum: 30

### 🔹 2.2 Keyword Arguments

We specify parameters by name, so order doesn’t matter.

✅ Example:

def introduce(name, age):

print(f"My name is {name} and I am {age} years old.")

introduce(age=25, name="Vijay")

Output:

My name is Vijay and I am 25 years old.

### 🔹 2.3 Default Arguments

We give a default value if no argument is passed.

✅ Example:

def greet(name="Guest"):

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

greet("Vijay")

greet()

Output:

Hello, Vijay!

Hello, Guest!

### 🔹 2.4 Variable-Length Arguments

When we don’t know how many values will be passed, we use:

#### (a) \*args → accepts multiple values as a **tuple**

def total\_sum(\*numbers):

print("Numbers:", numbers)

print("Sum:", sum(numbers))

total\_sum(10, 20, 30, 40)

Output:

Numbers: (10, 20, 30, 40)

Sum: 100

#### (b) \*\*kwargs → accepts multiple values as a **dictionary**

def person\_info(\*\*details):

print("Details:", details)

person\_info(name="Vijay", age=25, city="Nellore")

Output:

Details: {'name': 'Vijay', 'age': 25, 'city': 'Nellore'}

✅ Summary of Step 2:

* Positional → based on order
* Keyword → based on name
* Default → fallback value
* \*args → multiple positional values (tuple)
* \*\*kwargs → multiple keyword values (dictionary)