Python

**1. What is Python?**

Python is a popular programming language that is easy to read and write. People use it to build websites, automate tasks, analyze data, and create software. It’s known for being simple and powerful.

**2. What is the difference between a Mutable datatype and an Immutable data type?**

Mutable means the data can be changed after it is created. Example: A list in Python can be updated (like adding or removing items).  
Immutable means the data cannot be changed after it's created. Example: A string cannot be changed. You can only create a new one.

**3. What is the difference between a Set and Dictionary?**

## **1. List (like a grocery list – ordered and changeable)**

### **➤ What is it?**

A **list** is an **ordered collection** of items. You can **add**, **remove**, or **change** items.

### **➤ Syntax:**

python

CopyEdit

fruits = ['apple', 'banana', 'mango']

### **➤ Features:**

* Ordered (items have positions)
* Allows duplicate items
* Changeable (mutable)

### **➤ Example:**

python

CopyEdit

fruits = ['apple', 'banana', 'mango']  
print(fruits[0]) # apple  
fruits.append('grape')  
print(fruits) # ['apple', 'banana', 'mango', 'grape']

## **📦 2. Tuple (like your birth date – ordered but unchangeable)**

### **➤ What is it?**

A **tuple** is like a list, but you **cannot change** it after it’s created.

### **➤ Syntax:**

python

CopyEdit

person = ('John', 25, 'New York')

### **➤ Features:**

* Ordered
* Allows duplicates
* **Unchangeable** (immutable)

### **➤ Example:**

python

CopyEdit

person = ('John', 25, 'New York')  
print(person[1]) # 25

## **🔢 3. Set (like a basket of fruits – unordered and no duplicates)**

### **➤ What is it?**

A **set** is an **unordered collection** that does **not allow duplicates**.

### **➤ Syntax:**

python

CopyEdit

colors = {'red', 'blue', 'green'}

### **➤ Features:**

* Unordered
* **No duplicates**
* Changeable (you can add/remove items)

### **➤ Example:**

python

CopyEdit

colors = {'red', 'blue', 'green'}  
colors.add('yellow')  
colors.add('red') # red already exists, so nothing changes  
print(colors)

## **📚 4. Dictionary (like a contact book – key-value pairs)**

### **➤ What is it?**

A **dictionary** stores data in **key-value** pairs.

### **➤ Syntax:**

python

CopyEdit

student = {'name': 'Alice', 'age': 20, 'grade': 'A'}

### **➤ Features:**

* Unordered (as of Python 3.6+, maintains order of insertion)
* No duplicate keys
* Changeable

### **➤ Example:**

python

CopyEdit

student = {'name': 'Alice', 'age': 20}  
print(student['name']) # Alice  
student['age'] = 21 # Updating value  
student['city'] = 'Delhi' # Adding new key-value pair  
print(student)

### **🧠 Summary Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Ordered?** | **Changeable?** | **Duplicates Allowed?** | **Example Syntax** |
| List | ✅ Yes | ✅ Yes | ✅ Yes | ['a', 'b', 'c'] |
| Tuple | ✅ Yes | ❌ No | ✅ Yes | ('a', 'b', 'c') |
| Set | ❌ No | ✅ Yes | ❌ No | {'a', 'b', 'c'} |
| Dictionary | ✅\* Yes | ✅ Yes | ❌ Keys only | {'key': 'value', 'name': 'John'} |

# ---------- LIST ----------

print("🔵 LIST OPERATIONS")

fruits = ['apple', 'banana', 'mango']

# Accessing

print("First fruit:", fruits[0]) # 'apple'

# Adding items

fruits.append('orange') # Add to end

fruits.insert(1, 'grape') # Insert at position

print("After adding:", fruits)

# Removing items

fruits.remove('banana') # Remove by value

popped = fruits.pop() # Remove last

print("After removing:", fruits, "Popped:", popped)

# Updating

fruits[0] = 'kiwi' # Change value

print("After updating:", fruits)

# Looping

for fruit in fruits:

print("Fruit:", fruit)

# Check existence

print("Is mango in list?", 'mango' in fruits)

print("\n---------------------------\n")

# ---------- TUPLE ----------

print("🟣 TUPLE OPERATIONS")

person = ('Alice', 30, 'Doctor')

# Accessing

print("Name:", person[0])

# Looping

for item in person:

print("Item:", item)

# Count & index

print("Count of 30:", person.count(30))

print("Index of 'Doctor':", person.index('Doctor'))

# Tuples are immutable: person[1] = 31 will raise error

# If needed, convert to list -> update -> convert back

temp = list(person)

temp[1] = 31

person = tuple(temp)

print("After update:", person)

print("\n---------------------------\n")

# ---------- SET ----------

print("🟢 SET OPERATIONS")

colors = {'red', 'blue', 'green'}

# Adding

colors.add('yellow')

colors.add('red') # No effect (already exists)

print("After adding:", colors)

# Removing

colors.remove('blue') # Will throw error if not exists

colors.discard('purple') # Safe remove

print("After removing:", colors)

# Looping

for color in colors:

print("Color:", color)

# Set operations

a = {1, 2, 3}

b = {3, 4, 5}

print("Union:", a.union(b)) # {1,2,3,4,5}

print("Intersection:", a & b) # {3}

print("Difference (a - b):", a - b) # {1,2}

print("\n---------------------------\n")

# ---------- DICTIONARY ----------

print("🟠 DICTIONARY OPERATIONS")

student = {'name': 'John', 'age': 21}

# Accessing

print("Name:", student['name'])

# Adding/Updating

student['grade'] = 'A'

student['age'] = 22

print("After update:", student)

# Removing

del student['grade'] # Remove by key

removed = student.pop('age') # Remove and return

print("After removing:", student, "Removed age:", removed)

# Looping

for key in student:

print("Key:", key, "Value:", student[key])

# Using .items()

for key, value in student.items():

print(f"{key} => {value}")

# Check existence

print("Is 'name' in student?", 'name' in student)

**4. What is the difference between / and // operator in Python?**

/ gives the result as a decimal number (called float). Example: 5 / 2 = 2.5  
// gives the result as a whole number, removing the decimal part. Example: 5 // 2 = 2

**6. What is a pass in Python?**

The pass statement is used when you want to leave a part of the code empty but still make it valid.

**What is a break, continue, and pass in Python?**

break: Stops a loop completely.  
continue: Skips the current step and goes to the next one.  
pass: Does nothing. Used as a placeholder.

**10. What is slicing in Python?**

Slicing means picking a part of a list, string, or any collection. Example: text = "Hello"; text[1:4] gives "ell".

**11. What are Keywords in Python?**

Keywords are special words that Python uses for its own rules. You can't use them as names for your variables. Examples: if, else, while, for, class, def, return.

**12. What are local variables and global variables in Python?**

x = 10 # 🌍 Global variable

def show():

y = 5 # 🔒 Local variable

print("Inside function - x:", x) # Can access global x

print("Inside function - y:", y) # Local variable y

show()

# Outside function

print("Outside function - x:", x) # ✅ Works (global)

# print("Outside function - y:", y) # ❌ Error (y is local)

count = 0

def increment():

global count # Tell Python: I want to change the global 'count'

count += 1

increment()

print("Global count:", count) # Output: 1

1. 13. What are functions in Python?

A function is a block of code that performs a task. You can call it whenever needed.

🧠 In Python, there are 4 main types of functions:

Type Meaning

1. Built-in functions Already provided by Python

2. User-defined functions Created by you

3. Lambda functions Small, one-line anonymous functions

4. Recursion functions A function that calls itself

✅ 1. Built-in Functions

Python gives you many functions by default.

🧪 Examples:

python

Copy

Edit

print("Hello") # Prints text

len("Python") # Returns length

sum([1, 2, 3]) # Returns sum

max(5, 10) # Returns max

✅ 2. User-Defined Functions

You write your own functions using def.

🧪 Example:

python

Copy

Edit

def greet(name):

print("Hello", name)

greet("John")

✅ 3. Lambda Functions (Anonymous Function)

Used for small, quick tasks. Written in one line.

🧪 Example:

python

Copy

Edit

square = lambda x: x \* x

print(square(5)) # Output: 25

Same as:

python

Copy

Edit

def square(x):

return x \* x

✅ 4. Recursive Functions

A function that calls itself to solve problems like factorial, Fibonacci, etc.

🧪 Example:

python

Copy

Edit

def factorial(n):

if n == 1:

return 1

return n \* factorial(n - 1)

print(factorial(5)) # Output: 120

🔁 Bonus: Function Types by Return/Input Style

Type Example

No input, no return def say\_hello(): print("Hi")

With input, no return def greet(name): print("Hello", name)

No input, with return def get\_pi(): return 3.14

With input and return def add(a, b): return a + b

1. 14. What is the difference between append() and extend() methods?

append() adds one item to the list.  
extend() adds many items (like another list).

1. 15. What is Object Oriented Programming (OOP) and main features of OOPs?

# 🧠 Core Concepts of OOP (Explained Simply)

Imagine you’re building a simple system for a hospital.

Here’s what the system needs to handle:

Doctors, Nurses, and Patients are all people—but they behave differently.

Each person has common information: name, age, ID

But:

A Doctor can prescribe medicine

A Nurse can check vitals

A Patient can book an appointment

Let’s use OOP to model this.

✅ Step-by-Step Solution Using OOP

1. Create an abstract class called Person → for Abstraction

from abc import ABC, abstractmethod

class Person(ABC): # Abstract class

def \_\_init\_\_(self, name, age, id):

self.name = name

self.age = age

self.id = id

@abstractmethod

def perform\_duty(self):

pass

2. Create subclasses for Doctor, Nurse, and Patient

Each will inherit from Person → Inheritance

Each will implement perform\_duty() differently → Polymorphism

class Doctor(Person):

def perform\_duty(self):

print(f"Dr. {self.name} is prescribing medicine.")

class Nurse(Person):

def perform\_duty(self):

print(f"Nurse {self.name} is checking patient vitals.")

class Patient(Person):

def perform\_duty(self):

print(f"Patient {self.name} is booking an appointment.")

3. Encapsulation is already used

We grouped name, age, id, and behavior (perform\_duty) inside each class.

We could also make data private like this:

self.\_\_name = name # double underscore = private

4. Using all the classes

people = [

Doctor("John", 45, "D001"),

Nurse("Emily", 30, "N007"),

Patient("Alex", 25, "P123")

]

for person in people:

person.perform\_duty() # Polymorphism in action

💡 OUTPUT

csharp

Dr. John is prescribing medicine.

Nurse Emily is checking patient vitals.

Patient Alex is booking an appointment.

✅ Recap: What You Just Learned

OOP Concept Where It Was Used

Encapsulation Data + behavior grouped inside classes (name, age, perform\_duty)

Inheritance Doctor, Nurse, and Patient inherit from Person

Polymorphism perform\_duty() behaves differently for each object

Abstraction Person is an abstract class that hides the common structure

1. 16. What is the difference between a class and an object?

A class is a blueprint (like a car design). An object is something built from the blueprint (like the actual car).

1. 17. What is inheritance and different types of inheritance?

Inheritance lets one class use the features of another class. Types: Single, Multiple, Multilevel, Hierarchical.

1. 18. What is \_\_init\_\_?

It’s a special function that runs when an object is created. It's used to give values to the object.

1. 19. What is a lambda function?

What is a Lambda Function?

A lambda function is a small, anonymous (nameless) function written in one line.

It's used when you need a quick, simple function — often for sorting, filtering, or mapping.

Think of it as a shortcut to define a function.

✅ Syntax:

python

Copy

Edit

lambda arguments: expression

🔹 It can take any number of arguments, but only one expression.

🔹 The result of the expression is automatically returned.

📘 Real Example vs Lambda:

✅ Regular function:

python

Copy

Edit

def square(x):

return x \* x

print(square(5)) # 25

✅ Lambda version:

python

Copy

Edit

square = lambda x: x \* x

print(square(5)) # 25

💡 Both do the same thing, but lambda is shorter.

💡 Common Uses of Lambda:

1. With map() – apply function to each item:

python

Copy

Edit

numbers = [1, 2, 3, 4]

squares = list(map(lambda x: x \* x, numbers))

print(squares) # [1, 4, 9, 16]

2. With filter() – filter items:

python

Copy

Edit

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

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

print(evens) # [2, 4]

3. With sorted() – custom sort:

python

Copy

Edit

students = [("Alice", 80), ("Bob", 60), ("Charlie", 90)]

# Sort by scores

sorted\_students = sorted(students, key=lambda x: x[1])

print(sorted\_students)

❗ When not to use lambda:

If the function is complex or needs multiple lines, use def instead. Lambda is best for small tasks.

🧪 Try It Yourself:

python

Copy

Edit

double = lambda x: x \* 2

print(double(10)) # Output: 20

1. 20. What is self in Python?

self is used inside a class to refer to the current object. It helps the object remember its own data.

24. What is pickling and unpickling?

Pickling means saving Python objects into a file.  
Unpickling means loading them back.

1. 25. What are the generators in Python?

Generators are special functions that give back items one at a time, instead of all at once. They save memory.

1. 26. What is a map function in Python?

The map() function applies another function to each item in a list (or any sequence).

1. 27. What are Decorators?

🔄 1. What is a Generator in Python?

✅ A generator is a special type of function that:

Returns values one at a time using yield

Saves memory (doesn’t store all values in memory at once)

Can be looped through like a list

🧠 Think of it like:

A vending machine that gives you one item at a time, instead of a list handing everything at once.

🧪 Example:

python

Copy

Edit

def count\_up\_to(n):

i = 1

while i <= n:

yield i

i += 1

# Using the generator

for num in count\_up\_to(5):

print(num)

🧾 Output:

Copy

Edit

1

2

3

4

5

✅ Why use generators?

Saves memory (great for large data)

Stops where it left off (like pause/play)

More efficient for looping

⚠️ Key Points:

Use yield instead of return

Can’t go back once moved forward

You can use next() to get the next value manually

python

Copy

Edit

gen = count\_up\_to(3)

print(next(gen)) # 1

print(next(gen)) # 2

🎨 2. What is a Decorator in Python?

✅ A decorator is a function that adds extra features to another function without changing its code.

🧠 Think of it like:

Putting a gift inside a fancy box without touching the gift — the decoration is added outside.

🧪 Simple Example:

python

Copy

Edit

def decorator\_function(func):

def wrapper():

print("✨ Before the function")

func()

print("✨ After the function")

return wrapper

@decorator\_function

def say\_hello():

print("Hello!")

say\_hello()

🧾 Output:

pgsql

Copy

Edit

✨ Before the function

Hello!

✨ After the function

✅ Why use decorators?

To add logging, timing, authentication, etc. to functions

Reuse common code without rewriting

📦 Real-life Use:

You often see decorators in:

Flask/Django web frameworks

Logging or debugging tools

Authentication in APIs

🔁 Summary Table

Feature Generator Decorator

Keyword yield @decorator\_name

Purpose Generate values one by one Add features to existing functions

Saves memory? ✅ Yes ❌ Not related to memory

Use Case Large loops, file reading, streams Logging, access control, timing

1. 28. What are Iterators in Python?

What is an Iterator in Python?

An iterator is:

An object that can be looped through, one value at a time

It remembers where it left off

It uses two methods:

\_\_iter\_\_() → gets the iterator object

\_\_next\_\_() → gets the next value

🧠 Think of it like:

A TV remote:

\_\_iter\_\_() is picking the remote

\_\_next\_\_() is pressing "Next Channel"

✅ Simple Example with a List:

python

Copy

Edit

fruits = ['apple', 'banana', 'mango']

it = iter(fruits) # Create iterator

print(next(it)) # apple

print(next(it)) # banana

print(next(it)) # mango

# print(next(it)) # ❌ Error: No more items (StopIteration)

10. What are negative indexes and why are they used?  
Negative indexes allow access to sequence elements from the end, e.g., -1 is the last element5.

1. How do you copy an object in Python?  
Use the copy() method for shallow copies and the deepcopy() function from the copy module for deep copies5.

\*12. What do \*args and kwargs mean?  
\*args allows variable numbers of positional arguments; \*\*kwargs allows variable numbers of keyword arguments5.

How are arguments passed in Python: by value or by reference?  
Python uses "pass by object reference" (or "pass by assignment"), meaning references to objects are passed, not the actual objects5.

Explain exception handling in Python.  
Use try, except, else, and finally blocks to handle exceptions and errors gracefully5.

16. How is memory managed in Python?  
Python uses automatic memory management with reference counting and a garbage collector to reclaim unused memory35.

17. What is the difference between sorted() and sort()?  
sorted() returns a new sorted list from any iterable, while sort() modifies the list in place5.

18. What is the difference between compile-time and runtime errors?  
Compile-time errors occur during code compilation; runtime errors occur during execution. Python checks types at runtime5.

What is method overriding? How is it different from method overloading?  
Overriding: redefining a method in a subclass; overloading: same method name with different parameters (not natively supported in Python)5.