1 Printing Text

```
print("Hello World")
print("Welcome to Python")
print("I am learning Python")
```

2 Printing Numbers

```
print(10)
print(3.14)
print(2026)
```

3 Printing Multiple Things

```
print("My name is", "Mayank")
print("Age:", 14)
print("Class", 9, "Student")
```

4 Printing Variables

```
name = "Mayank"
age = 14
print(name)
print(age)
print("Name:", name)
print("Age:", age)
```

5 Basic Arithmetic using `print()`

```
print(5 + 3)
print(10 - 4)
print(6 * 2)
print(8 / 2)
print(10 % 3)
```

6 Printing Expressions

```
print(2 + 3 * 4)
print((2 + 3) * 4)
```

```
print(100 / 5 + 10)
```

7️⃣ Printing Boolean Values

```
print(True)
print(False)
print(5 > 3)
print(10 == 5)
```

8️⃣ Printing New Lines

```
print("Hello\nWorld")
print("Python\nis\nEasy")
```

9️⃣ Printing with `\t` (Tab Space)

```
print("Name\tAge\tClass")
print("Mayank\t14\t9")
```

🔟 Printing Quotes

```
print("He said, 'Python is easy'")
print('She said, "I like coding"')
```

1️⃣1️⃣ Printing Repeated Text

```
print("Hello " * 5)
print("Python\n" * 3)
```

1️⃣2️⃣ Printing Simple Patterns

```
print("*")
print("**")
print("***")
print("* * *")
print("* * *")
print("* * *")
```

---

1 3 Printing Calculations with Text

```
print("Sum is:", 5 + 5)
print("Product is:", 4 * 6)
```

---

1 4 Printing Type of Data

```
print(type(10))
print(type(3.14))
print(type("Python"))
```

---

1 5 Simple Info Program

```
name = "Mayank"
school = "CBSE"
print("Name:", name)
print("School Board:", school)
```

Python Data Types – Simple Programs

---

1 Integer (`int`)

```
a = 10
print(a)
print(type(a))
b = -25
print(b)
print(type(b))
```

---

2 Float (`float`)

```
x = 3.14
print(x)
print(type(x))
y = 10.0
print(y)
print(type(y))
```

---

3 String (`str`)

```
name = "Mayank"
print(name)
print(type(name))
msg = "Python is easy"
print(msg)
print(type(msg))
```

---

4 Boolean (`bool`)

```
a = True
print(a)
print(type(a))
b = False
print(b)
print(type(b))
print(5 > 3)
print(type(5 > 3))
```

📃 Python List – Definition & Examples

◆ Definition of List

A list is a collection of multiple values stored in a single variable.

- Lists are **ordered**
- Lists are **changeable (mutable)**
- Lists allow **duplicate values**
- Lists are written using **square brackets `[ ]`**

◆ Simple List Example

```
marks = [80, 90, 85]
print(marks)
```

◆ List with Different Data Types

```
data = [10, 3.5, "Python", True]
print(data)
```

◆ List of Strings

```
fruits = ["apple", "banana", "mango"]
print(fruits)
```

◆ List of Numbers

```
numbers = [1, 2, 3, 4, 5]
print(numbers)
```

◆ Duplicate Values in List

```
nums = [1, 2, 2, 3]
print(nums)
```

◆ Access List Elements

```
colors = ["red", "green", "blue"]
print(colors[0])
print(colors[1])
```

◆ Change List Value (Mutable)

```
colors = ["red", "green", "blue"]
colors[1] = "yellow"
print(colors)
```

◆ Length of List

```
marks = [70, 80, 90]
print(len(marks))
```

◆ Nested List

```
matrix = [[1, 2], [3, 4]]
print(matrix)
```

◆ Check List Data Type

```
items = ["pen", "pencil", "eraser"]
print(type(items))
```

📝 One-Line Exam Definition (Write This)

A list in Python is an ordered, mutable collection of items written using square brackets `[ ]`.

📄 Python Tuple – Definition & Examples

◆ Definition of Tuple

A tuple is a collection of multiple values stored in a single variable.

- Tuples are **ordered**
- Tuples are **unchangeable (immutable)**
- Tuples allow **duplicate values**
- Tuples are written using **round brackets `( )`**

◆ Simple Tuple Example

```
numbers = (10, 20, 30)
print(numbers)
```

◆ Tuple with Different Data Types

```
data = (10, 3.5, "Python", True)
```

```
print(data)
```

◆ Tuple of Strings

```
fruits = ("apple", "banana", "mango")
print(fruits)
```

◆ Tuple of Numbers

```
nums = (1, 2, 3, 4, 5)
print(nums)
```

◆ Duplicate Values in Tuple

```
values = (1, 2, 2, 3)
print(values)
```

◆ Access Tuple Elements

```
colors = ("red", "green", "blue")
print(colors[0])
print(colors[2])
```

◆ Length of Tuple

```
marks = (70, 80, 90)
print(len(marks))
```

◆ Nested Tuple

```
t = ((1, 2), (3, 4))
print(t)
```

◆ Single-Element Tuple (Important)

```
a = (5,)
print(a)
```

```
print(type(a))
```

❗ Without comma it is **not** a tuple:

```
b = (5)
print(type(b))
```

◆ Tuple is Immutable (Cannot Change)

```
t = (10, 20, 30)
# t[1] = 50   ✕ Error
print(t)
```

◆ Check Tuple Data Type

```
items = ("pen", "pencil", "eraser")
print(type(items))
```

📝 One-Line Exam Definition

A tuple in Python is an ordered, immutable collection of items written using round brackets `( )`.

🔁 Difference: List vs Tuple (Exam Favorite)

Feature	List	Tuple
Brackets	`[ ]`	`( )`
Changeable	Yes	No
Speed	Slower	Faster

🔍 How do we check which is slower or faster in Python?

We check **speed using time measurement** ⏱️
Python has a built-in module called **`time`** to do this.

🧠 Concept (Simple Words)

- We **run the same task**
- Once using **list**
- Once using **tuple**
- Then we **compare time taken**

Less time = **faster**
More time = **slower**

✅ Simple Example

⏱️ Checking speed of LIST

```
import time

start = time.time()

lst = [1, 2, 3, 4, 5] * 1000000

end = time.time()

print("List time:", end - start)
```

⏱️ Checking speed of TUPLE

```
import time

start = time.time()

tup = (1, 2, 3, 4, 5) * 1000000

end = time.time()

print("Tuple time:", end - start)
```

📌 Output (Example)

```
List time: 0.12
Tuple time: 0.08
```

☞ **Tuple took less time → Tuple is faster**

❓ WHY tuple is faster than list?

Reason	Explanation
Mutable	List can change → extra memory
Immutable	Tuple cannot change → fixed
Memory	List uses more memory
Speed	Tuple uses less memory → faster

🧪 Real-Life Example

- **List** = Whiteboard (you can erase & change → slow)
- **Tuple** = Printed paper (fixed → fast)

📝 Exam Answer (Write This)

Tuples are faster than lists because tuples are immutable and require less memory, while lists are mutable and slower.

◆ Example 1: All Data Types in One Program

```
a = 10                  # int
b = 3.5                 # float
c = "Python"            # string
d = (1, 2, 3)           # tuple
e = [4, 5, 6]           # list
print(a)
print(b)
print(c)
print(d)
print(e)
```

◆ Example 2: Printing Values with Their Types

```
x = 25
y = 7.8
name = "Mayank"
t = ("red", "green")
lst = [10, 20, 30]

print(x, type(x))
print(y, type(y))
print(name, type(name))
print(t, type(t))
print(lst, type(lst))
```

◆ Example 3: Mixed Data Types in List

```
data = [10, 2.5, "Hello", (1, 2), [3, 4]]
print(data)
```

◆ Example 4: Student Information

```
roll = 5
percentage = 89.5
name = "Rahul"
subjects = ("Math", "Science")
marks = [90, 85]

print(roll)
print(percentage)
print(name)
print(subjects)
print(marks)
```

◆ Example 5: Single Print with Multiple Values

```
a = 10
b = 5.5
c = "Total"
d = (1, 2)
e = [3, 4]

print(a, b, c, d, e)
```

📝 One-Line Understanding

✓ **int** → whole number
✓ **float** → decimal number
✓ **string** → text
✓ **tuple** → fixed collection
✓ **list** → changeable collection

📃 Python Dictionary – Definition, Examples & Main Points

◆ Definition of Dictionary

A dictionary is a collection of data stored in key–value pairs.
Each key is used to access its value.

- Written using **curly brackets** `{ }`
- Data is stored as **key : value**
- Keys are **unique**
- Values can be of **any data type**

◆ Simple Dictionary Example

```
student = {"name": "Mayank", "age": 14}
print(student)
```

◆ Dictionary with Numbers

```
marks = {"Math": 90, "Science": 85}
print(marks)
```

◆ Access Dictionary Values

```
student = {"name": "Mayank", "age": 14}
print(student["name"])
print(student["age"])
```

◆ Dictionary with Different Data Types

```
data = {
    "roll": 5,
    "percentage": 89.5,
    "name": "Rahul"
}
print(data)
```

◆ Dictionary with List & Tuple

```
info = {
    "subjects": ("Math", "Science"),
    "marks": [90, 85]
}
print(info)
```

◆ Change Dictionary Value

```
student = {"name": "Mayank", "age": 14}
student["age"] = 15
print(student)
```

◆ Add New Key–Value Pair

```
student = {"name": "Mayank"}
student["class"] = 9
print(student)
```

◆ Check Dictionary Type

```
d = {"a": 1, "b": 2}
print(type(d))
```

📝 Main Points (Very Important for Exam)

✓ Dictionary stores data in **key : value** form
✓ Written using **{ }**
✓ Keys are **unique**

✓ Values can be **duplicate**
✓ Dictionary is **changeable (mutable)**
✓ Keys are usually **string or number**
✓ Values can be **int, float, string, list, tuple, etc.**

📝 One-Line Exam Definition

A dictionary in Python is a mutable collection that stores data in key–value pairs using curly brackets `{ }`.

🔁 Example Comparison (Easy)

Feature	Dictionary
Brackets	`{ }`
Data stored as	key : value
Ordered (Python 3.7+)	Yes
Changeable	Yes

📄 Python Set – Definition & Examples

◆ Definition of Set

A set is a collection of unique values stored in a single variable.

- Written using **curly brackets `{ }`**
- **Duplicate values are not allowed**
- Set is **unordered**
- Set is **changeable (mutable)**

◆ Simple Set Example

```
numbers = {1, 2, 3, 4}
print(numbers)
```

◆ Set with Duplicate Values

```
nums = {1, 2, 2, 3}
print(nums)
```

☞ Output will remove duplicates:

```
{1, 2, 3}
```

◆ Set with Different Data Types

```
data = {10, 3.5, "Python"}
print(data)
```

◆ Create Empty Set (Important)

✕ Wrong way:

```
s = {}
print(type(s))   # Dictionary
```

✓ Correct way:

```
s = set()
print(type(s))   # Set
```

◆ Add Element to Set

```
fruits = {"apple", "banana"}
fruits.add("mango")
print(fruits)
```

◆ Remove Element from Set

```
fruits = {"apple", "banana", "mango"}
fruits.remove("banana")
print(fruits)
```

◆ Check Set Data Type

```
a = {1, 2, 3}
print(type(a))
```

◆ Set Cannot Be Indexed

✕ This is not allowed:

```
s = {10, 20, 30}
# print(s[0])   ✕ Error
```

📝 Main Points (Exam Focus)

✓ Set stores **unique values only**
✓ Duplicate values are **automatically removed**
✓ Set is **unordered** (no fixed position)
✓ Set is **mutable**
✓ Set elements are **not accessed by index**
✓ Used for **mathematical operations**

📝 One-Line Exam Definition

A set in Python is an unordered, mutable collection of unique elements written using curly brackets `{ }`.

🔁 Difference (Quick View)

Feature	Set
Duplicates	❌ Not allowed
Order	❌ Unordered
Indexing	❌ No
Brackets	`{ }`

🧠 Where Set Is Used?

- Removing duplicate values
- Common items (intersection)
- Unique roll numbers
- Membership checking

📃 Python Methods (Easy & Important)

◆ 1 LIST METHODS (`list`)

```
lst = [10, 20, 30]
```

Method	Use	Example
`append()`	Add element at end	`lst.append(40)`
`insert()`	Add at position	`lst.insert(1, 15)`
`remove()`	Remove element	`lst.remove(20)`
`pop()`	Remove last element	`lst.pop()`
`sort()`	Sort list	`lst.sort()`
`reverse()`	Reverse list	`lst.reverse()`

Method	Use	Example
`count()`	Count value	`lst.count(10)`
`len()`	Length of list	`len(lst)`

✓ Example:

```
lst.append(40)
print(lst)
```

◆ 2 TUPLE METHODS (`tuple`)

☞ Tuple is **immutable**, so **very few methods**

```
t = (10, 20, 10, 30)
```

Method	Use	Example
`count()`	Count value	`t.count(10)`
`index()`	Find position	`t.index(20)`
`len()`	Length	`len(t)`

✓ Example:

```
print(t.count(10))
```

◆ 3 SET METHODS (`set`)

```
s = {1, 2, 3}
```

Method	Use	Example
`add()`	Add element	`s.add(4)`
`remove()`	Remove element	`s.remove(2)`
`discard()`	Remove safely	`s.discard(5)`
`pop()`	Remove random	`s.pop()`
`clear()`	Remove all	`s.clear()`
`union()`	Combine sets	`a.union(b)`
`intersection()`	Common values	`a.intersection(b)`

✓ Example:

```
s.add(5)
print(s)
```

◆ 4 DICTIONARY METHODS (`dict`)

```
d = {"name": "Mayank", "age": 14}
```

Method	Use	Example
`keys()`	Get keys	`d.keys()`
`values()`	Get values	`d.values()`
`items()`	Key-value pairs	`d.items()`
`get()`	Get value	`d.get("name")`
`update()`	Add/update	`d.update({"class":9})`
`pop()`	Remove key	`d.pop("age")`
`clear()`	Remove all	`d.clear()`

✓ Example:

```
print(d.get("name"))
```

📝 EXAM SHORT NOTES (VERY IMPORTANT)

- **List** → many methods (mutable)
- **Tuple** → only `count()` & `index()`
- **Set** → no duplicates, unordered
- **Dictionary** → key-value based
