**DO FOLLOW DEEKSHITHA TECH TALKS <3**

## 1️⃣ Introduction to Python

### What is Python?

- Python is a **high-level** language → easy to read & write (English-like).
- **Interpreted** → code runs line by line (no compilation needed).
- **Object-Oriented** → everything is treated as an object.

### Why Python is popular?

- Simple syntax
- Huge library support
- Used by top companies (Google, Amazon, Netflix)

### Where Python is used?

- 🌐 Web Development (Django, Flask)
- 🤖 AI & Machine Learning
- 📊 Data Science
- ⚙️ Automation & Testing
- ☁️ Cloud & DevOps

## 2️⃣ Data Types & Variables

### What is a Variable?

- A **variable** is a name used to store data in memory.

```
x = 10
name = "Python"
```

---

### 🔢 int (Integer)

- Stores **whole numbers**

```
a = 10
b = -5
```

---

### 🔢 float

- Stores **decimal numbers**

```
price = 99.5
pi = 3.14
```

---

### 🔢 complex

- Stores numbers with **real + imaginary part**

```
c = 2 + 3j
```

---

### 🔤 str (String)

- Stores **text**

```
name = "Deekshitha"
```

---

### 📦 list

- Ordered, mutable (can change)

```
numbers = [1, 2, 3]
```

---

### 📦 tuple

- Ordered, immutable (cannot change)

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

---

### 📦 set

- Unordered, unique values only

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

---

### 📦 dict (Dictionary)

- Stores data as **key : value**

```
student = {"name": "A", "age": 20}
```

---

### 🔍 type()

- Tells the datatype

```
type(10)   # int
```

### 🆔 id()

- Memory address of variable

```
id(x)
```

## 3️⃣ OPERATORS IN PYTHON (DETAILED)

### 👉 What are Operators?

Operators are **symbols used to perform operations** on variables and values.
They are very important in **coding rounds & interviews**.

---

### ➕ Arithmetic Operators

Used to perform **mathematical calculations**

| Operator | Meaning | Example | Output |
|---|---|---|---|
| + | Addition | 10 + 5 | 15 |
| - | Subtraction | 10 - 5 | 5 |
| * | Multiplication | 10 * 5 | 50 |
| / | Division | 10 / 4 | 2.5 |
| % | Modulus (remainder) | 10 % 3 | 1 |
| // | Floor division | 10 // 3 | 3 |
| ** | Power | 2 ** 3 | 8 |

### 📌 Interview Tip:

- / always gives **float output**
- % is used a lot in **even/odd**, **palindrome**, **digit problems**

---

### 🔍 Relational (Comparison) Operators

Used to **compare two values**
Output is always **True or False**

| Operator | Meaning | Example | Output |
|---|---|---|---|
| > | Greater than | 10 > 5 | True |
| < | Less than | 5 < 3 | False |
| == | Equal to | 5 == 5 | True |
| != | Not equal | 5 != 3 | True |
| >= | Greater or equal | 5 >= 5 | True |
| <= | Less or equal | 3 <= 5 | True |

📌 Used in **if-else conditions**

---

### 🔗 Logical Operators

Used to **combine multiple conditions**

| Operator | Meaning | Example | Output |
|---|---|---|---|
| and | Both true | (5>2 and 10>3) | True |
| or | Any one true | (5>2 or 10<3) | True |
| not | Reverse result | not(5>2) | False |

📌 **Interview Example:**

```
if age > 18 and citizen == True:
   print("Eligible")
```

---

### 📝 Assignment Operators

Used to **assign or update values**

| Operator | Meaning | Example | Result |
|---|---|---|---|
| = | Assign | x = 10 | 10 |
| += | Add & assign | x += 5 | 15 |
| -= | Subtract & assign | x -= 3 | 12 |
| *= | Multiply & assign | x *= 2 | 24 |
| /= | Divide & assign | x /= 4 | 6.0 |

📌 Used to **optimize code** in loops

---

### 🔎 Membership Operators

Used to **check presence of value in a collection**

| Operator | Meaning | Example | Output |
|---|---|---|---|
| in | Present | 'a' in 'apple' | True |
| not in | Not present | 5 not in [1,2,3] | True |

📌 Used in **strings, lists, sets, dictionaries**

---

### ⚙️ Bitwise Operators

Operate on **binary numbers** (0 & 1)

| Operator | Meaning | Example | Result |
|---|---|---|---|
| & | AND | 5 & 3 | 1 |
| ` | ` | OR | `5 |
| ^ | XOR | 5 ^ 3 | 6 |
| ~ | NOT | ~5 | -6 |
| << | Left shift | 5 << 1 | 10 |
| >> | Right shift | 5 >> 1 | 2 |

📌 Mostly asked in **bit manipulation questions**

---

🔥 **Interview Summary**

✔ Arithmetic ⟶ calculations
✔ Relational ⟶ comparisons
✔ Logical ⟶ multiple conditions
✔ Assignment ⟶ value update
✔ Membership ⟶ searching
✔ Bitwise ⟶ low-level optimization

## 4️⃣ Control Statements

**if statement**

```
if age > 18:
    print("Eligible")
```

---

**if-else**

```
if x > 0:
    print("Positive")
else:
    print("Negative")
```

---

**Loops**

**for loop**

```
for i in range(5):
    print(i)
```

**while loop**

```
while x < 5:
    x += 1
```

---

**break**

- Stops loop

**continue**

- Skips current iteration

**pass**

- Does nothing (placeholder)

---

## 5️⃣ Functions

**What is a function?**

- Reusable block of code

```
def add(a, b):
    return a + b
```

## 5️⃣ FUNCTIONS IN PYTHON (DETAILED)

### 👉 What is a Function?

A **function** is a **reusable block of code** that performs a specific task.
Functions help to:

- Reduce code repetition
- Improve readability
- Make debugging easy

**Basic Function Example**

```
def add(a, b):
    return a + b


print(add(10, 5))   # Output: 15
```

---

🗒️ **Function Syntax**

```
def function_name(parameters):
    statements
    return value
```

---

📌 **Types of Arguments (VERY IMPORTANT FOR INTERVIEWS)**

**1️⃣Positional Arguments**

- Values are passed **in order**

```
def sub(a, b):
    return a - b


sub(10, 5)   # a=10, b=5
```

⚠️ Order matters

---

**2️⃣Keyword Arguments**

- Values passed using **parameter name**

```
sub(b=5, a=10)
```

✔ Order does NOT matter

---

**3️⃣Default Arguments**

- Default value is used if argument not passed

```
def greet(name="User"):
    print("Hello", name)


greet()       # Hello User
greet("Ram")  # Hello Ram
```

---

**4️⃣Variable Length Arguments**

***args (Non-keyword arguments)**

- Accepts multiple values as a **tuple**

```
def total(*args):
```

```
    return sum(args)


total(1,2,3,4)
```

---

****kwargs (Keyword arguments)**

- Accepts multiple values as a **dictionary**

```
def info(**kwargs):
    print(kwargs)


info(name="A", age=20)
```

📌 Used in **real-time projects & frameworks**

---

🔹 **Return vs Print**

| **return** | **print** |
|---|---|
| Sends value back | Displays output |
| Can be reused | Cannot reuse |

---

⚡ **Lambda Function (Anonymous Function)**

**What is Lambda?**

- One-line function
- No function name
- Used for **short operations**

```
square = lambda x: x * x
print(square(5))   # 25
```

**Lambda with multiple arguments**

```
add = lambda a, b: a + b
```

📌 Used with map(), filter(), reduce()

---

6️⃣ **OOPS CONCEPTS IN PYTHON (DETAILED)**

👉 **What is OOPS?**

**Object Oriented Programming System**
Organizes code using **classes & objects**

---

### 🧱 Class

A **class** is a **blueprint** or template.

```
class Student:
    pass
```

---

### 📦 Object

An **object** is an **instance of a class**

```
s1 = Student()
```

---

### 🔧 Constructor (__init__)

- Automatically called when object is created
- Used to initialize data

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
s = Student("Ram", 20)
```

📌 self refers to **current object**

---

### 🧩 OOPS Pillars (VERY IMPORTANT)

#### 1️⃣ Encapsulation

- Binding data and methods together
- Data hiding using private variables

```
class Bank:
    def __init__(self):
        self.__balance = 1000
```

---

#### 2️⃣ Inheritance

- Child class acquires parent properties

```
class Parent:
    pass


class Child(Parent):
    pass
```

✔ Code reuse

---

**3️⃣ Polymorphism**

- Same function name, different behavior

```
print(len("Python"))
print(len([1,2,3]))
```

---

**4️⃣ Abstraction**

- Hiding implementation details
- Achieved using abstract classes

📌 Focus on **what**, not **how**

**7️⃣ EXCEPTION HANDLING (DETAILED)**

👉 **What is an Exception?**

An **exception** is a **runtime error** that stops the normal flow of program execution.

Common exceptions:

- ZeroDivisionError
- ValueError
- TypeError
- IndexError

---

**Basic Example**

```
try:
    x = 10 / 0
except:
    print("Error occurred")
```

---

**try–except–else–finally**

```
try:
    x = int(input())
except:
    print("Invalid input")
else:
    print("Success")
finally:
    print("Always executed")
```

**finally**

✔ Always executes
✔ Used for closing files, database connections

---

**Custom Exception**

Used when **built-in exceptions are not enough**

```
age = -5
if age < 0:
    raise Exception("Age cannot be negative")
```

📌 **Interview Tip:**
Explain **why** custom exceptions are needed (business rules)

---

## 8️⃣ FILE HANDLING (DETAILED)

👉 **What is File Handling?**

Used to **store data permanently** in files.

---

**open() function**

```
f = open("file.txt", "r")
```

**File Modes**

| Mode | Meaning |
|---|---|
| r | Read |

| Mode | Meaning |
|---|---|
| w | Write (overwrite) |
| a | Append |
| r+ | Read & write |

---

**read(), write(), append()**

f.read()

f.write("Hello")

---

**with keyword (BEST PRACTICE)**

- Automatically closes file
- Prevents memory leaks

```
with open("file.txt") as f:
    print(f.read())
```

📌 **Interview Tip:**
Always say **with keyword is safer**

---

## 9️⃣ MODULES & PACKAGES (DETAILED)

### 👉 Module

A **module** is a Python file (.py) that contains:

- functions
- variables
- classes

Example:

```
import math

print(math.sqrt(25))
```

---

### 👉 Package

A **package** is a **collection of modules** organized in folders.

Example:

```
mypackage/
```

module1.py

module2.py

📌 Used to **organize large projects**

---

## 🔟 DATA STRUCTURES (DETAILED)

**List**

- Dynamic
- Ordered
- Mutable

lst = [1,2,3]

---

**Stack (LIFO)**

Last In First Out

Operations:

- push
- pop

stack = []

stack.append(10)

stack.pop()

---

**Queue (FIFO)**

First In First Out

from collections import deque

q = deque()

q.append(1)

q.popleft()

---

**Dictionary**

- Key-value pair
- Fast lookup

d = {"name":"A", "age":20}

---

**Comprehension**

Short & efficient way to create collections

[x*x for x in range(5)]

📌 Saves lines of code → interviewer likes this

---

## 1️⃣1️⃣ REGULAR EXPRESSIONS (REGEX)

👉 **What is Regex?**

Used for **pattern matching** in strings

Examples:

- Email validation
- Mobile number check
- Password validation

import re

---

**search()**

Searches pattern anywhere

re.search("a", "apple")

---

**match()**

Matches from beginning only

re.match("a", "apple")

---

**findall()**

Returns all matches

re.findall("\d", "a1b2")

---

## 1️⃣2️⃣ MULTITHREADING vs MULTIPROCESSING

**Multithreading**

- Multiple threads in one process
- Best for **I/O tasks** (file, network)

import threading

---

**Multiprocessing**

- Multiple processes
- Uses multiple CPU cores
- Best for **CPU-intensive tasks**

import multiprocessing

📌 **Interview Question:**
Why multiprocessing over multithreading?
👉 Due to **GIL limitation**

---

## 1️⃣3️⃣ PYTHON FOR PLACEMENTS (MOST IMPORTANT)

**Frequently Asked Topics**

✔ Strings
✔ Arrays
✔ Loops
✔ Patterns
✔ Functions

---

**String Example**

```
s = "python"

print(s[::-1])
```

---

**Pattern Example**

```
for i in range(1,5):
    print("*"*i)
```

---

**Logic Building**

- Think before coding
- Break problem into steps
- Use dry run

---

**Time Complexity (VERY IMPORTANT)**

| Complexity | Meaning |
|---|---|
| O(1) | Constant time |
| O(n) | Linear |
| $O(n^2)$ | Nested loops |

Example:

```
for i in range(n):   # O(n)
for i in range(n):
    for j in range(n):  # O(n²)
```

📌 Interviewers LOVE when you mention time complexity

---

👉 **What is a String?**

A **string** is a sequence of **characters** enclosed in:

- single quotes ' '
- double quotes " "

```
s = "Python"
```

Strings are **immutable** → cannot be changed once created.

---

🔹 **String Indexing**

Each character has an index (starts from 0)

```
s = "Python"
print(s[0])   # P
print(s[-1])  # n
```

---

🔹 **String Slicing**

Used to extract part of a string

```
s = "Python"
print(s[1:4])   # yth
print(s[::-1])  # nohtyP (reverse)
```

📌 **Interview Favorite:** Reverse a string using slicing

---

🔹 **Common String Methods (VERY IMPORTANT)**

| Method | Use |
| --- | --- |
| upper() | Convert to uppercase |
| lower() | Convert to lowercase |
| strip() | Remove spaces |
| replace() | Replace text |
| split() | Split string |
| join() | Join strings |
| find() | Find index |
| count() | Count occurrences |

```
s = " hello python "
print(s.strip())
print(s.upper())
```

---

🔹 **String Operations**

```
a = "Py"
b = "thon"
print(a + b)     # Concatenation
print(a * 3)     # Repetition
```

---

🔹 **Important String Interview Programs**

**Reverse a String**

```
s = "python"
print(s[::-1])
```

---

**Check Palindrome**

```
s = "madam"
print(s == s[::-1])
```

---

**Count Vowels**

```
s = "hello"
```

```
count = 0
for c in s:
    if c in "aeiouAEIOU":
        count += 1
print(count)
```

---

**Remove Duplicate Characters**

```
s = "programming"
print("".join(set(s)))
```

---

### 🎯 String Interview Tips

✔ Strings are immutable
✔ Slicing is powerful
✔ Practice reverse, palindrome, frequency

---

### 📦 ARRAYS IN PYTHON (DETAILED)

Python does not have built-in arrays like C/Java.
We mainly use **Lists** as arrays.

---

### 👉 What is an Array (List)?

An **array** stores multiple values of **same or different types**.

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

Lists are:
✔ Ordered
✔ Mutable
✔ Dynamic

---

### 🔹 Array Indexing & Slicing

```
arr = [10,20,30,40]
print(arr[0])     # 10
print(arr[1:3])   # [20,30]
```

---

### 🔹 Common List (Array) Methods

| Method | Use |
| --- | --- |
| append() | Add element |
| insert() | Insert at index |
| remove() | Remove element |
| pop() | Remove last |
| sort() | Sort list |
| reverse() | Reverse list |
| len() | Length |

```
arr.append(50)
arr.sort()
```

---

- **Traversing an Array**

```
for i in arr:
    print(i)
```

---

- **Important Array Interview Programs**

**Find Largest Element**

```
arr = [10, 25, 3]
print(max(arr))
```

---

**Sum of Elements**

```
print(sum(arr))
```

---

**Remove Duplicates**

```
arr = [1,2,2,3]
print(list(set(arr)))
```

---

**Reverse an Array**

```
print(arr[::-1])
```

---

🔹 **List Comprehension (VERY IMPORTANT)**

Short way to create arrays

squares = [x*x for x in range(5)]

---

⏱️ **Time Complexity (INTERVIEW MUST)**

| **Operation** | **Complexity** |
|---|---|
| Access | O(1) |
| Search | O(n) |
| Insert/Delete | O(n) |

🎯 **FINAL INTERVIEW TIP**

✔ Write clean code
✔ Explain logic step-by-step
✔ Don’t panic
✔ Think loud