

Python Programming – Day 1

Focus: Foundations that matter

Goal: Write your first clean Python programs

What is python?

- Python is a **high-level programming language**
- Created by **Guido van Rossum**
- Designed for **readability & simplicity**
- Used by beginners and professionals alike



Guido Van Rossum

Why Python?

Python is popular because it is:

- Easy to learn
- Powerful
- Widely used

Use Cases:

- Backend (FastAPI, Django)
- Automation & scripting
- Data analysis & ML
- Cybersecurity & tooling



Python vs C

Python

- High-level language
- Easy to read & write
- Uses **indentation** instead of braces
- Dynamically typed
- Garbage collected
- Slower execution (but faster development)

```
1 print("Hello world")
2
3
```

C

- Low-level language
- Complex syntax
- Uses `{}` and `;`
- Statically typed
- Manual memory management
- Very fast execution

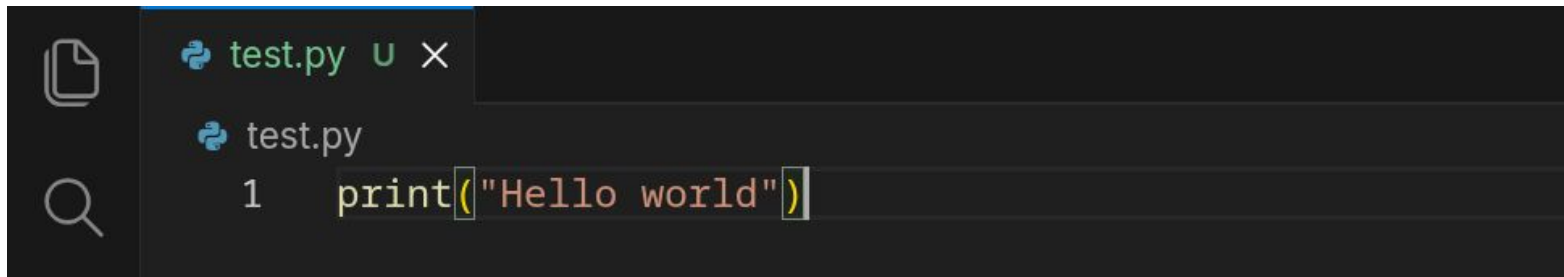
```
1 #include <stdio.h>
2
3 int main(){
4     printf("Hello world");
5     return 0;
6 }
7
```

Ways to Use Python

1. System Python

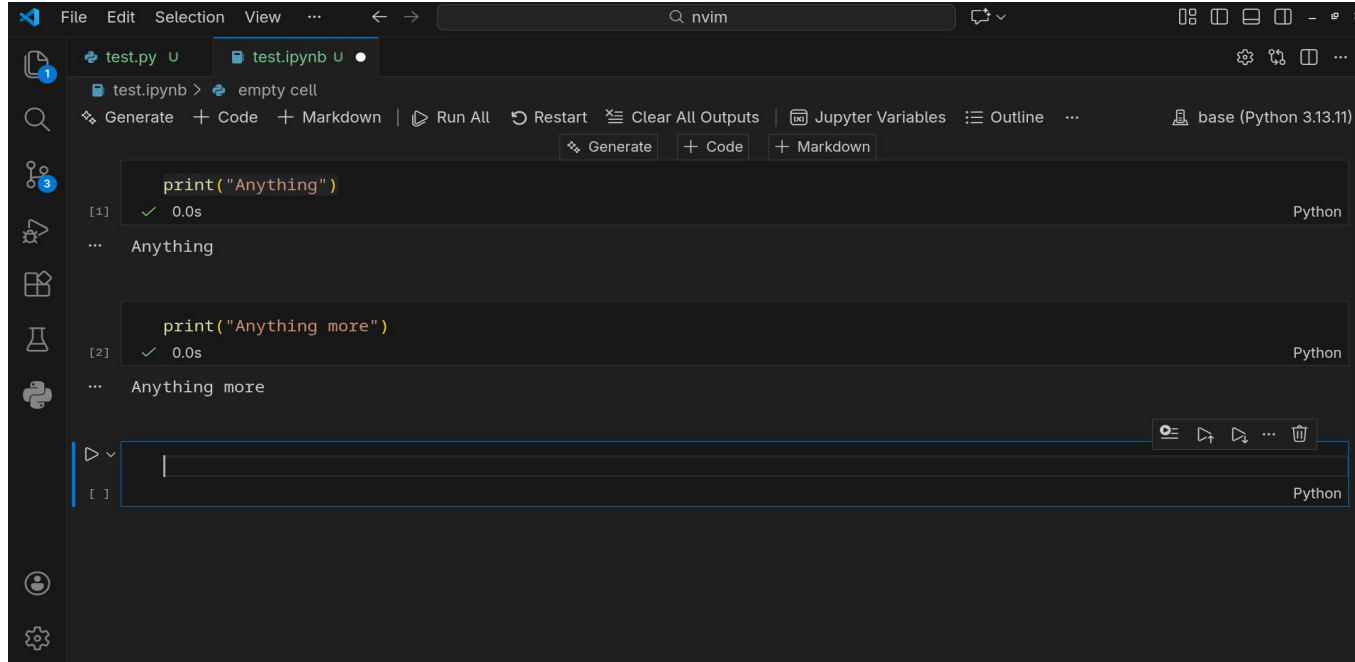
```
(base) → nvim git:(master) python
Python 3.13.11 | packaged by Anaconda, Inc. | (main, Dec 10 2025, 21:28:48) [GCC 14.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

2. IDE



Ways to Use Python

3. Notebook



What is a Virtual Environment?

A **virtual environment** is:

- An isolated Python workspace
- Keeps dependencies separate per project

Why it matters:

- Avoids version conflicts
- Clean & professional workflow
- Industry standard



```
> python -m venv venv  
> source venv/bin/activate (Mac/Linux)  
> venv\Scripts\activate (Windows)  
> deactivate
```

PIP vs UV

Pip (Most used)

- Traditional Python package manager
- Widely used
- Slower dependency resolution

Uv (less in used)

- Modern & fast
- Written in Rust
- Better dependency handling

Your First Python Code

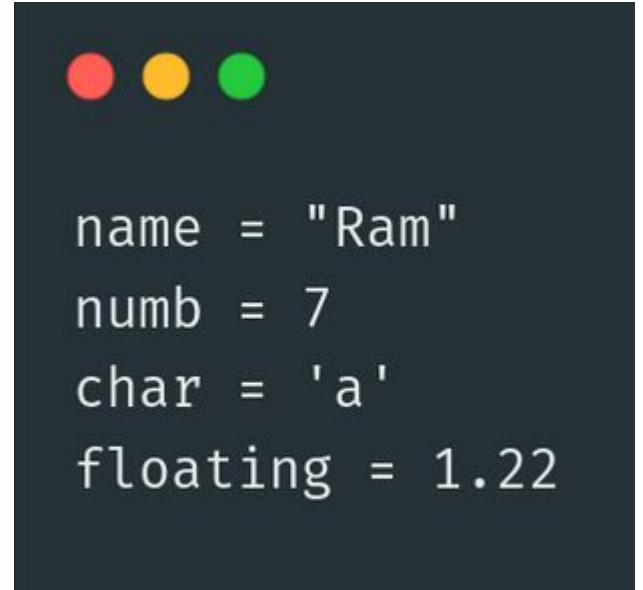
- `print()` outputs text to the screen
- This is your first Python program 🎉



```
print("Hello, Python!")
```

Variables & Data Types

- Variables store data.
- Common Python data types:
 - `int` → 10, -5
 - `float` → 3.14, 0.5
 - `str` → "hello"
 - `bool` → True / False



```
name = "Ram"  
numb = 7  
char = 'a'  
floating = 1.22
```

Input & Output


- Output:



```
print("Enter your name")
```

Input & Output

- Input:



```
a = input('Enter your name')
```

Type Casting

- Convert one type to another



```
age = int(input("Enter age: "))  
height = float("5.9")
```

Control flow

- Control flow is the way to **control the order in which code executes** in a program.

Key Components:

- Conditional statements: `if`, `elif`, `else`
- Loops: `for`, `while`
- Loop controls: `break`, `continue`, `pass`

Control flow (if-elif-else)



```
age = 20
```

```
if age < 13:  
    print("Child")  
elif age < 20:  
    print("Teenager")  
else:  
    print("Adult")
```

Logical Operators

What are Logical Operators?

- Logical operators are used to **combine or modify conditions** in Python.

Common Operators:

- **and** → True if **both** conditions are True
- **or** → True if **at least one** condition is True
- **not** → Reverses a condition

Logical Operators



```
age = 20
```

```
citizen = True
```

```
if age ≥ 18 and citizen:  
    print("Eligible to vote")
```

Practice Task

- **Even / Odd Checker**
 - 1. Prompt user to enter a number
 - 2. Read the number and store in variable NUM
 -
 - 3. IF NUM modulo 2 equals 0 THEN
 - PRINT "Number is Even"
 - ELSE
 - PRINT "Number is Odd"

HomeWork Question

Write a Python program that:

1. Takes three numbers as input from the user
2. Determines and prints:
 - The **largest number**
 - Whether it is **even or odd**

-

Day 2

By Pujan Neupane

Loops, Data Structures & Built-ins

By Pujan Neupane

What Are Loops?

Loops let us **repeat actions** without writing the same code again.

Why loops matter:

- Reduce code repetition
- Handle large data easily
- Automate repetitive tasks

For Loop

Used to iterate over a sequence (list, string, range, etc.)

Key idea:

- Runs **for each item** in a sequence



```
for i in range(start, stop, step):  
    print(i)
```

while Loop

Runs **while a condition is true**

Key idea:

- Condition-based repetition



```
while(condition):  
    # code
```


Loop Control Keywords

Break: Stops the loop immediately

Continue : Skips current iteration, moves to next

Pass : Does Nothing

Break: Control Keywords



```
for i in range(1, 6):  
    if i == 3:  
        break  
    print(i)
```

Continue : Control Keywords



```
for i in range(1, 6):  
    if i == 3:  
        continue  
    print(i)
```

pass : Control Keywords



```
for i in range(1, 4):  
    if i == 2:  
        pass  
    print(i)
```

Question: What will be printed by this code?

- a. 0 1 2
- b. 0 1
- c. 1 2 3
- d. 0 1 2 3 4



```
for i in range(5):  
    if i == 2:  
        break  
    print(i)
```

Task

- Write a program to print **all numbers from 1 to 10 except 5**, and stop **if the number is 8**.

Data Structures in Python

Used to **store and organize data**

Main types:

- List
- Tuple
- Set
- Dictionary

List

- Ordered collection
- Mutable (can be changed)



```
aList = []  
aList = list()
```


List: Example



```
# List example
fruits = ["apple", "banana", "mango"]
print(fruits)
print(fruits[0])    # apple
fruits[1] = "orange"
print(fruits)
```

Tuple

- Ordered collection
- Immutable (cannot be changed)



```
# Tuple example
coordinates = (10, 20)

# Access by index
print(coordinates[0]) # 10

# coordinates[0] = 50 (ERR)
```

Set

- Unordered collection
- No Duplicates
- mutable



```
# Set example
numbers = {1, 2, 3, 3, 4}

print(numbers) # duplicates removed automatically

# Membership testing
print(3 in numbers) # True
print(5 in numbers) # False
```

Dictionary

- Key–Value Pairs
- Mutable

Used for:

- Fast lookups
- Mapping relationships



```
student = {  
    "name": "Pujan",  
    "age": 20,  
    "course": "Python"  
}  
print(student["name"])  
student["age"] = 21  
print(student)
```

Indexing & Slicing

Indexing


- Access single element

Slicing

- Access a range of elements

Works with:

- Lists
- Tuples
- Strings



```
text = "Python"  
nums = [10, 20, 30, 40, 50]  
print(text[0])  
print(nums[2])  
print(text[0:3])  
print(nums[1:4])
```

Mutability vs Immutability

Mutable

- Can change after creation
- Example: List, Dictionary

Immutable

- Cannot change after creation
- Example: Tuple, String



```
# Mutable example (List)
a = [1, 2, 3]
a[0] = 99
print(a)

# Immutable example (Tuple)
b = (1, 2, 3)
# b[0] = 99      error
print(b)
```

Task: CODE

Question: Create a **list of fruits**, a **tuple of colors**, and a **set of numbers**.
Then:

- Add a new fruit to the list
- Try changing a tuple element (observe error)
- Add a new number to the set

Built-in Functions

`len()` – count items (already done)

`type()` – check data type (already done)

`range()` – generate numbers (already done)

`sorted()` – sort data

`enumerate()` – index + value

`sum()` – total

`min()` – smallest value

`max()` – largest value

`sorted()` – sort data



```
numbers = [5, 2, 9, 1, 3]
```

```
sorted_numbers = sorted(numbers)
```

```
print(sorted_numbers)
```

`enumerate()` – Index + Value



```
fruits = ["apple", "banana", "mango"]  
  
for index, value in enumerate(fruits):  
    print(index, value)
```

`sum()` – Total



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

```
total = sum(numbers)
```

```
print(total)
```

`min()` – Smallest Value from the list



```
numbers = [10, 5, 30, 2]
```

```
print(min(numbers))
```

`max()` – Largest Value from the list



```
numbers = [10, 5, 30, 2]
```

```
print(max(numbers))
```

Task: CODE

Question: Given a list of marks `[40, 55, 70, 85]`, write a program to print:

- Total marks
- Highest mark
- Lowest mark
- Marks sorted in ascending order

Day 3

By Pujan Neupane

Function && Clean Code

By Pujan Neupane

What is a Functions?

A **function** is a reusable block of code that performs a specific task.

Why functions matter:

- Reduce code repetition
- Improve readability
- Easier debugging and maintenance

Defining a Function



```
def function_name():  
    # code block  
    pass
```

Example of a Function



```
def greet():  
    print("Hello, World!")
```

```
greet()
```

Parameters & Return Values

- Parameters allow data to be passed into a function.

Key points:

- `return` sends data back to the caller
- A function can return multiple values



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

```
result = add(4, 6)  
print(result)
```

Default Arguments

- Default arguments provide a fallback value.
- Why use defaults?
 - Prevent errors
 - Make functions flexible




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

```
greet()  
greet("Pujan")
```

Keyword Arguments


- Arguments passed using parameter names.



```
def user_info(name, age):  
    print(name, age)  
  
user_info(age=20, name="Pujan")
```

*args (Variable Arguments)


- Used to pass multiple positional arguments.



```
def total(*args):  
    return sum(args)  
  
print(total(1, 2, 3, 4))
```

****kwargs** (Keyword Arguments)

- Used to pass multiple keyword arguments.



```
def profile(**kwargs):  
    for key, value in kwargs.items():  
        print(key, value)  
  
profile(name="Pujan", role="Student")
```


Question: Use args and kwargs

- Magic Number Machine

Write a function that:

- Accepts **any number of integers** using `*args`
- Prints:
 - the **total count** of numbers
 - the **sum**
 - the **largest number**



```
magic_numbers(5, 10, 3, 25, 7)
```

Question: Use kwargs

- Character Profile Builder

Write a function that:

- Accepts **character details** using ****kwargs**
- Prints a **game-style profile**

```
character(name="Pujan", level=5, power="Python", hp=100)
```

Character Profile

Name : Pujan

Level : 5

Power : Python

HP : 100

Lambda Functions

- Anonymous one-line functions.



```
lambda arguments: expression
```

Lambda Functions example:

- Anonymous one-line functions.



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

Scope (Local vs Global)

Local variables:

- Defined inside a function
- Accessible only within that function

Global variables:


- Defined outside functions
- Accessible everywhere

A dark-themed code editor window with three colored window control buttons (red, yellow, green) at the top left. The code inside is as follows:

```
x = 10
def demo():
    x = 5
    print(x)
demo()
print(x)
```

Docstrings

- Docstrings describe what a function does.



```
def add(a, b):  
    """Returns the sum of two numbers"""  
    return a + b
```

Function Naming

Good naming rules:

- Use lowercase
- Use underscores
- Be descriptive

Examples:

- `calculate_total()`
- `get_user_data()`

Avoid:

- `func1()`
- `x()`

Reusability & Best Practices

Best practices:

- Keep functions small
- Do one task per function
- Avoid hardcoding values

Reusable functions:

- Save time
- Improve scalability

OOP

- By Pujan Neupane

What is OOP?

Object-Oriented Programming (OOP) is a way of structuring programs using **classes** and **objects**.

Why OOP?

- Organizes code better
- Improves reusability
- Models real-world entities

Class & Object

Class

A blueprint for creating objects.

Object

An instance of a class.



```
class User:  
    pass  
u1 = User()
```

`__init__` Method


- Special method (constructor)
- Runs automatically when an object is created



```
class User:  
    def __init__(self, name):  
        self.name = name
```

Instance vs Class Variables

- **Instance variable** → unique per object
- **Class variable** → shared among all objects



```
class User:
    role = "member" # class

def __init__(self, name):
    self.name = name
    # instance variable
```

Theory Question:

- Why do we use the `__init__` method?
- What is the difference between a class and an object?

Coding Question:

- Create a `Car` class with attributes: `brand`, `model`, `year`.
 - Create 2 objects and print their details.
 - Add a method `age()` to the `Car` class to calculate the age of the car (current year - year of the car).

soln:

```
# Car Class Example
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def display(self):
        print(f"{self.brand} {self.model} ({self.year})")

    def age(self, current_year):
        return current_year - self.year

# Objects
car1 = Car("Toyota", "Corolla", 2018)
car2 = Car("Honda", "Civic", 2020)


car1.display()
car2.display()
current_year = 2025
print(f"Car1 Age: {car1.age(current_year)} years")
print(f"Car2 Age: {car2.age(current_year)} years")
```


Magic Methods

- Magic methods start and end with double underscores.
- Examples:
 - `__init__`
 - `__str__`
 - `__repr__`

`__str__` Method

- Defines **user-friendly** string output



```
class User:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"User name is {self.name}"
```

`__repr__` Method

- Used for **developer/debugging** representation



```
class User:
    def __repr__(self):
        return f"User(name={self.name})"
```

Question: code

- Create a `Book` class with attributes: `title`, `author`.
 - Use `__str__` to print: `"Book: <title> by <author>"`
 - Use `__repr__` to print developer-friendly format:
`"Book(title='<title>', author='<author>')"`

What happens if you print the object without defining `__str__`?

soln:

```
# Book Class Example
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"Book: {self.title} by {self.author}"


    def __repr__(self):
        return f"Book(title='{self.title}',
author='{self.author}')"

b1 = Book("1984", "George Orwell")

print(b1)          # calls __str__
print(repr(b1))    # calls __repr__
```

Inheritance

- Inheritance allows a class to acquire properties of another class.



```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    pass
```

Polymorphism

- Same method name, different behavior.

```
class Dog:
    def speak(self):
        print("Bark")

class Cat:
    def speak(self):
        print("Meow")
```

Method Overriding

- Child class modifies parent method.



```
class Dog(Animal):  
    def speak(self):  
        print("Dog barks")
```


Operator Overloading

- Redefining operators using magic methods.

Examples:

- `+` \rightarrow `__add__`
- `==` \rightarrow `__eq__`

Operator Overloading : Example


```
class Number:
    def __init__(self, value):
        self.value = value
    def __add__(self, other):
        return Number(self.value + other.value)

    def __str__(self):
        return str(self.value)
n1 = Number(10)
n2 = Number(20)
print(n1 + n2)
```

Question:

Create a **Distance** class:

- Attributes: **feet**, **inches**
- Overload the **+** operator to **add two distances**.
- 12 inches = 1 foot (handle overflow properly)
- Add a **__str__** method to display in "**X ft Y in**" format



```
d1 = Distance(5, 8)
d2 = Distance(3, 9)
print(d1 + d2)
```

Files, Errors & Standard Library

- By Pujan Neupane

File Handling – Basics


Reading a Text File:



```
with open("data.txt", "r") as f:  
    content = f.read()  
    print(content)
```

File Handling – Basics

Write a Text File:



```
with open("data.txt", "w") as f:  
    f.write("Hello, Python!\n")
```

File Handling – Basics

File Modes:

`r` → Read

`w` → Write (overwrite)

`a` → Append


`rb` / `wb` → Binary modes



```
with open("log.txt", "a") as f:  
    f.write("New log entry\n")
```

File Handling – Basics

Write in JSON Files



```
import json

data = {
    "name": "Pujan",
    "role": "Cybersecurity"
}

with open("user.json", "w") as f:
    json.dump(data, f, indent=4)
```


File Handling – Basics

Reading JSON Files



```
with open("user.json", "r") as f:  
    data = json.load(f)  
    print(data["name"])
```

Question

- JSON is used to save the player's game progress.
- Player data is stored as key–value pairs.
- The data is written to a file so it can be loaded later.



```
{  
  "name": "Hero",  
  "level": 3,  
  "score": 1500  
}
```

Question

Solution:



```
import json

# create player data
player = {
    "name": "Hero",
    "level": 1,
    "score": 0
}

# write to json file
with open("savegame.json", "w") as file:
    json.dump(player, file, indent=4)

print("Game saved successfully!")
```

Exception Handling – Basics



```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ValueError:
    print("Invalid input")
except ZeroDivisionError:
    print("Cannot divide by zero")
```

Exception Handling – Basics



```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ValueError:
    print("Invalid input")
except ZeroDivisionError:
    print("Cannot divide by zero")
```

Exception Handling – Basics

Try

Except

Else

finally

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for exception handling. The code uses the try-except-else-finally structure. The 'except' block catches a FileNotFoundError and prints a message. The 'finally' block prints a message indicating execution is finished. The line 'print("Execution finished")' is highlighted with a light green background.

```
try:  
    f = open("data.txt")  
except FileNotFoundError:  
    print("File not found")  
else:  
    print(f.read())  
finally:  
    print("Execution finished")
```

Exception Handling – Task

Write a Python program that:

- Takes a number as input
- Converts it to an integer
- Handles **invalid input** using `ValueError`

Exception Handling – Task



```
try:
    number = int(input("Enter a number"))
except ValueError:
    print("invalid input")
else:
    print(number)
```


MODULES :

datetime Module :

Why **datetime**?

- Work with current date & time
- Create timestamps for logs
- Format date/time for humans or systems

Datetime



```
from datetime import datetime
now = datetime.now()
print(now)
print(now.strftime("%Y-%m-%d %H:%M:%S"))
```

os Module (Operating System Interaction)

Why **os**?

- Interact with file system
- Manage folders & paths
- Environment-level operations

os Module (Operating System Interaction)



```
# Get Current Working Directory
import os
print(os.getcwd())
#Create a Directory
os.mkdir("logs")
#Check If File or Folder Exists
print(os.path.exists("logs"))
```

`sys` Module (System-Level Access)

Why `sys`?

- Access Python runtime info
- Read command-line arguments
- Control program execution

`sys` Module (System-Level Access)



```
import sys  
print(sys.argv)
```

Practice – Log Generator



```
from datetime import datetime  
import uuid
```

```
log_id = uuid.uuid4()  
time = datetime.now()
```

```
with open("app.log", "a") as f:  
    f.write(f"[{time}] {log_id} - User logged in")
```

asyncio

- **asyncio** is a Python library for **asynchronous programming**
- It allows programs to run **multiple tasks concurrently**
- Uses **event loop**, **async**, and **await**
- Best for **I/O-bound tasks** (network calls, APIs, file operations)

asyncio



```
import asyncio

async def func():
    print("Hello World!")
    await asyncio.sleep(2)
    print("Taranga ma aau")

asyncio.run(func())
```

Pygame


- By Pujan Neupane

What is Pygame?

- Python library for making games
- Built on top of SDL
- Used for 2D games, simulations, and visuals
- Handles:
 - Graphics
 - Keyboard & mouse input
 - Sound
 - Game loop

Installing Pygame

- `pip install pygame`



```
import pygame
print(pygame.__version__)
```

Basic Pygame Structure

Every Pygame program has:

- Initialize pygame
- Create a game window
- Game loop
- Event handling
- Quit pygame

Init pygame



```
import pygame
```

```
pygame.init()
```

```
screen = pygame.display.set_mode((800, 600))
```

```
pygame.display.set_caption("Game")
```

Game Loop



```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
```

Colors & Drawing



```
WHITE = (255, 255, 255)
```

```
BLUE = (0, 0, 255)
```

```
screen.fill(WHITE)
```

```
pygame.draw.rect(screen, BLUE, (100, 100, 50, 50))
```

```
pygame.display.update()
```


Keyboard Input



```
keys = pygame.key.get_pressed()
```

```
if keys[pygame.K_LEFT]:
```

```
    x -= 5
```

```
if keys[pygame.K_RIGHT]:
```

```
    x += 5
```

Frame Rate





```
clock = pygame.time.Clock()  
clock.tick(60)
```

Project day

- Pujan Neupane


Mentors



Sandip Sapkota

Student | Web Developer

+977-9748706471 | sandipsapkota9844@gmail.com




Sandip Sapkota · 1st


Software Engineer @Sayapatri Group | Back-End Development |
Typescript and Go Lang

Dang District, Lumbini, Nepal · [Contact info](#)


[304 connections](#)

 Anurag Sharma, Akhileshwar pandit, and 163 other mutual connections

[Message](#) [More](#)



Sayapatri Group



Tribhuvan University, IOE,
Purwanchal Campus

Mentors



sujal karki · 1st

Undergraduate student at IOE Purwanchal Campus

Biratnagar, Kosi Zone, Nepal · [Contact info](#)

[56 connections](#)



Rajesh Thapa, Tilak Thapa, and 42 other mutual connections

[Message](#)

[More](#)



Tribhuvan University, IOE,
Purwanchal Campus



Mentor



Prachit Bhujel · 1st

Computer Engineering Student | ML and AI enthusiast

Biratnagar, Kosi Zone, Nepal · [Contact info](#)

[168 connections](#)



Anurag Sharma, Rajesh Thapa, and 88 other mutual connections

[Message](#)


[More](#)



Tribhuvan University, IOE,
Purwanchal Campus



Mentors





Suwarna Pyakurel


Space Robotics and tech enthusiast

Undergraduate Engineer

<https://github.com/suwarna-wave>

 suwarnapyakurel5@gmail.com







Suwarna Pyakurel

• 1st


Outreach Coordinator at ICRTAI | Strategic Public Relations Planning, Event Management | Technical Documentation

Kathmandu, Bāgmatī, Nepal · [Contact info](#)


1,250 followers · [500+ connections](#)

  Anurag Sharma, Aarzu Gautam, and 325 other mutual connections

[Message](#) [More](#)





TARANGA, IOE Purwanchal Campus




Tribhuvan University, IOE, Purwanchal Campus


Mentors






Pratyooos Panta  · 1st

Undergrad Computer Engineering Student into Full Stack WebDev
Kathmandu, Bāgmatī, Nepal · [Contact info](#)



[Personal Portfolio](#) 

975 followers · [500+ connections](#)

 Anurag Sharma, Aarzu Gautam, and 243 other mutual connections

[Message](#) [More](#)

Open to work
Full Stack Developer and Back End Developer roles
[Show details](#)

 **TARANGA, IOE Purwanchal Campus**
 **Tribhuvan University, IOE, Purwanchal Campus**

Tutor



Hello, my name is Pujan Neupane.
Nice to meet you.


Pujan Neupane [Add verification badge](#)

ML & Cyber security Enthusiast | Web Developer | C, C++ & Python Learner

Nawalparasi District, Nepal · [Contact info](#)

500+ connections

[Open to](#) [Add profile section](#) [Enhance profile](#) [Resources](#)

Open to work 

Web Developer, Python Developer, Game Developer and Machine Learning Engineer roles

[Show details](#)

 Cyber Alert Nepal

 Tribhuvan University, IOE,
Purwanchal Campus

Project Time

Time : 1 hr

Project idea:

Python Password Security Tool


Description

You are required to build a **menu-driven Python program** that helps users:


1. Check the **strength of a password**
2. Simulate a **secure login system**

The program should use **basic Python concepts only** and must run in the **terminal**.

Project Base setup :



```
Enter password: Test@123
Password Strength: STRONG (4/4)
```




```
def check_password(password):
    score = 0
    # TODO: add checks
    return score


def login():
    pass

while True:
    print("1. Check Password")
    print("2. Login")
    print("3. Exit")
```

Project Base setup :



```
Enter password: Test@123  
Password Strength: STRONG (4/4)
```



```
def check_password(password):  
    score = 0  
    # TODO: add checks  
    return score  
  
def login():  
    pass  
  
while True:  
    print("1. Check Password")  
    print("2. Login")  
    print("3. Exit")
```

Thank you

- By Pujan Neupane