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

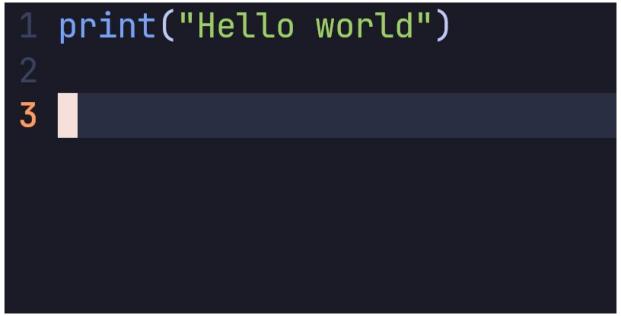
Use Cases:

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

Python vs C

Python

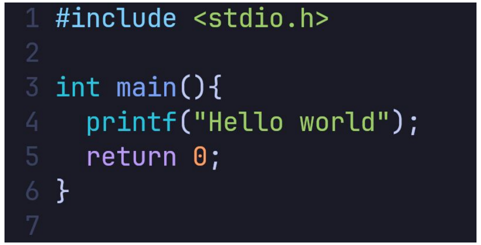
- High-level language
- Easy to read & write
- Uses **indentation** instead of braces
- Dynamically typed
- Garbage collected
- Slower execution (but faster development)



```python
print("Hello world")

```

C

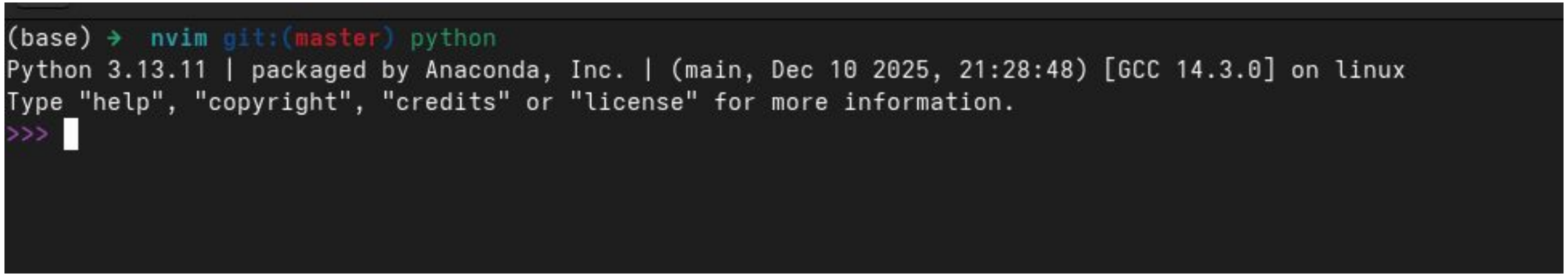
- Low-level language
- Complex syntax
- Uses `{}` and `;`
- Statically typed
- Manual memory management
- Very fast execution



```c
#include <stdio.h>

int main(){
  printf("Hello world");
  return 0;
}

```

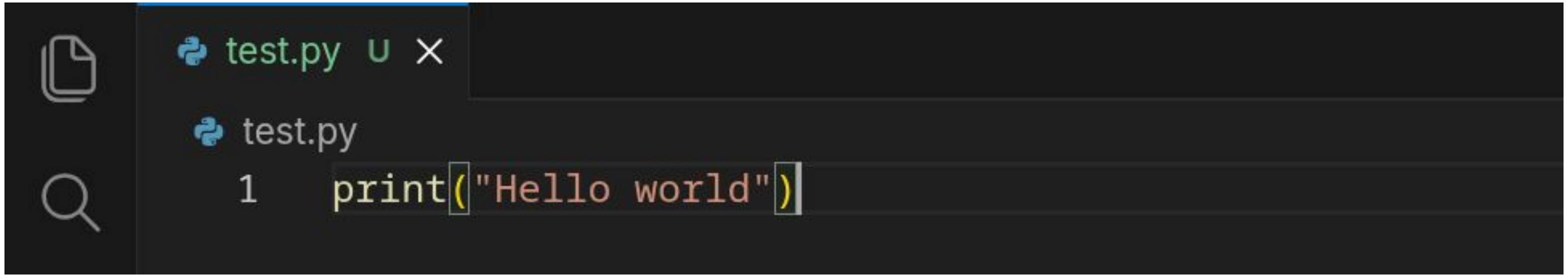
Ways to Use Python

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.
>>> 
```

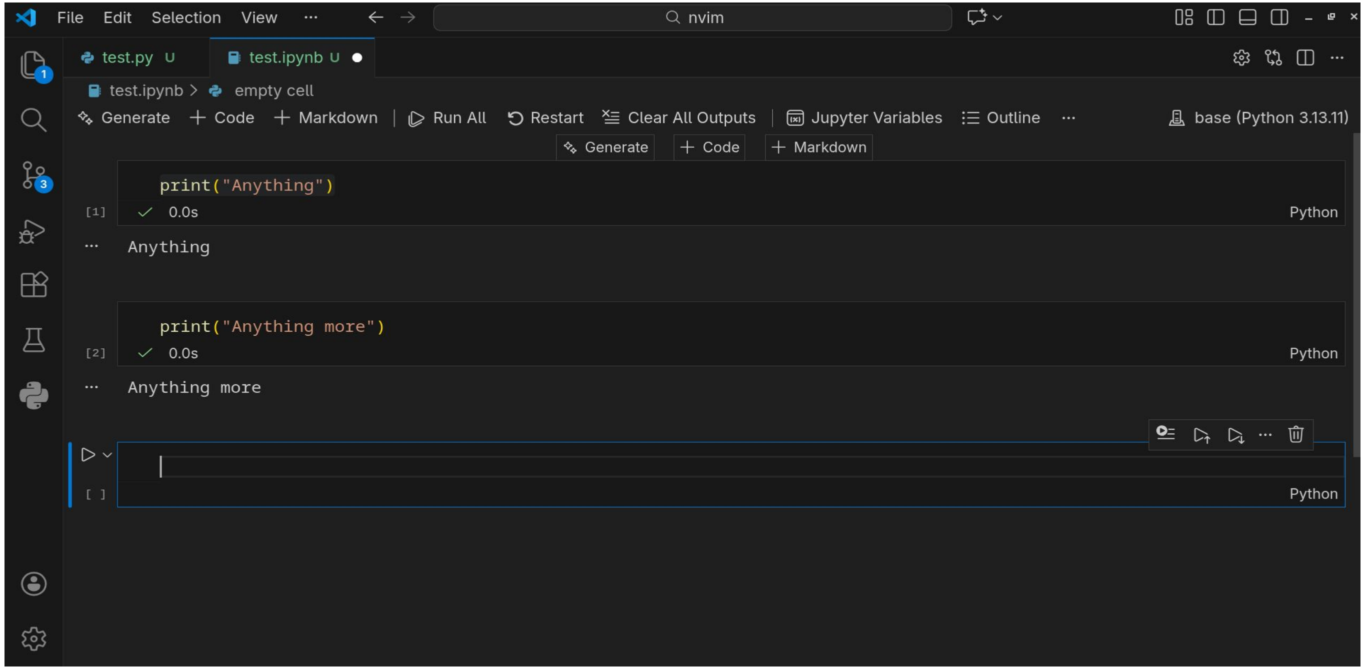
2. **IDE**



```
print("Hello world")
```

Ways to Use Python

3. Notebook



```
[1] print("Anything")
    ✓ 0.0s
... Anything

[2] print("Anything more")
    ✓ 0.0s
... Anything more
```

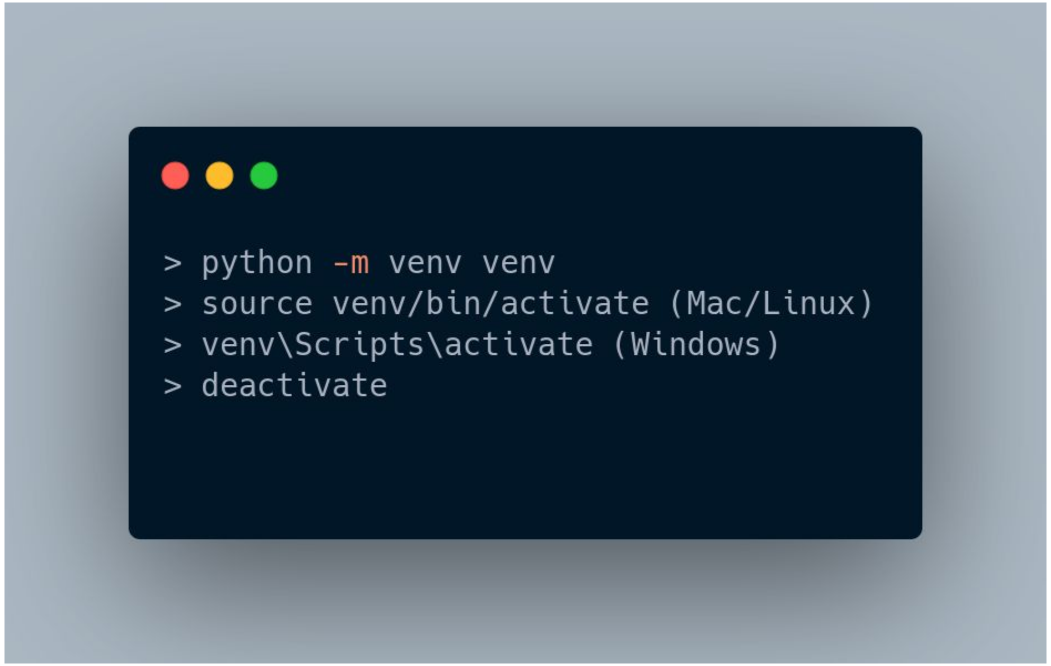
What is a Virtual Environment?

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)

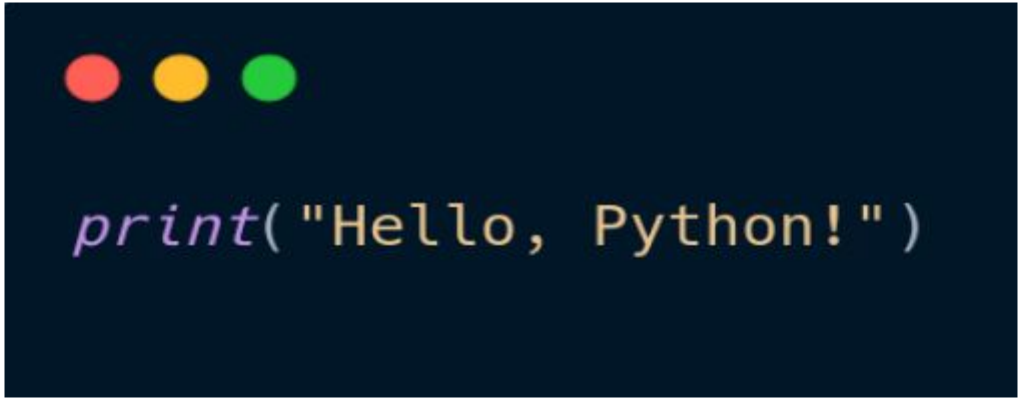
- Traditional Python package manager
- Widely used
- Slower dependency resolution

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 🎉

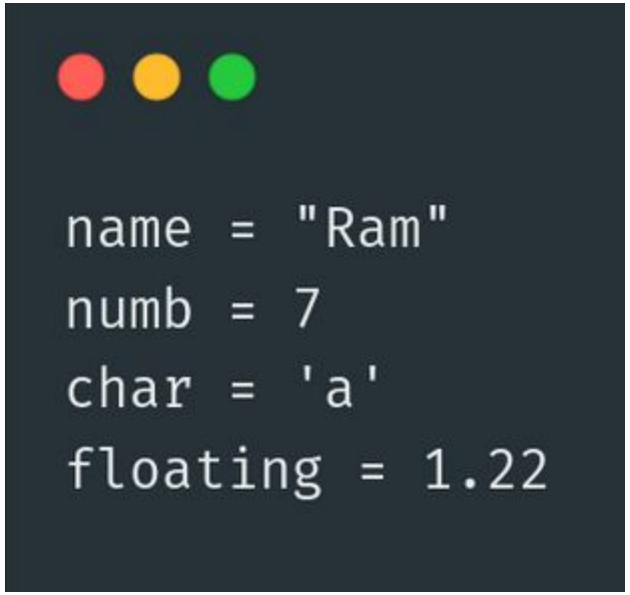


```python
print("Hello, Python!")
```

Variables && Data Types

- Variables store data.

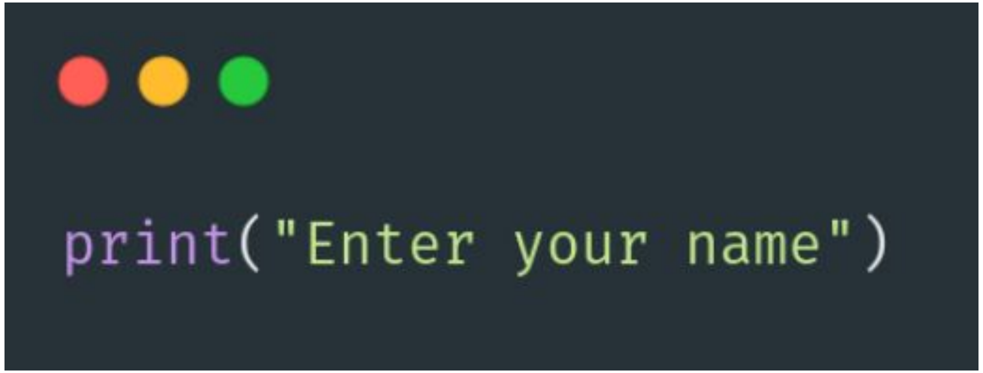
- Common Python data types:
  - `int` → 10, -5
  - `float` → 3.14, 0.5
  - `str` → "hello"
  - `bool` → True / False



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

Input & Output

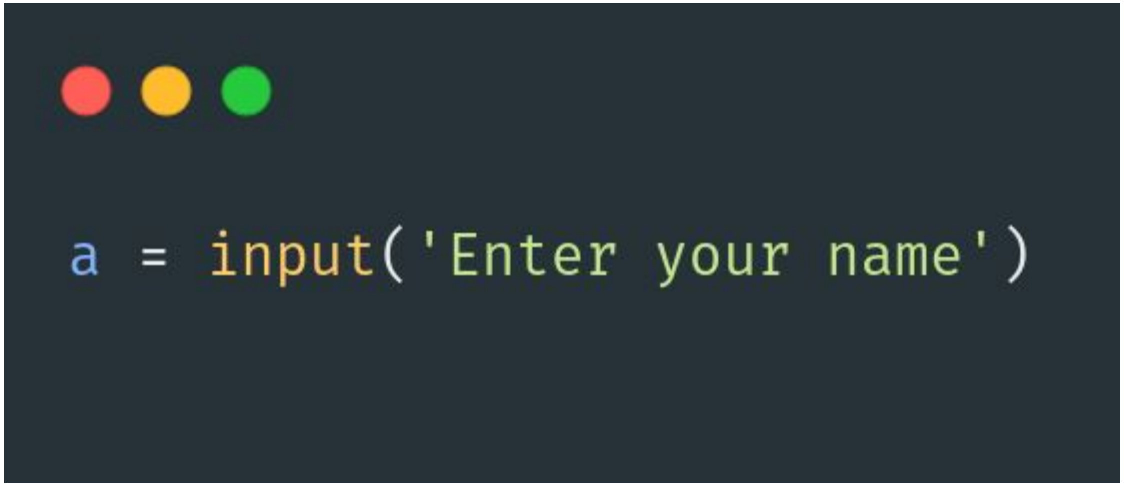
- Output:



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

Input & Output

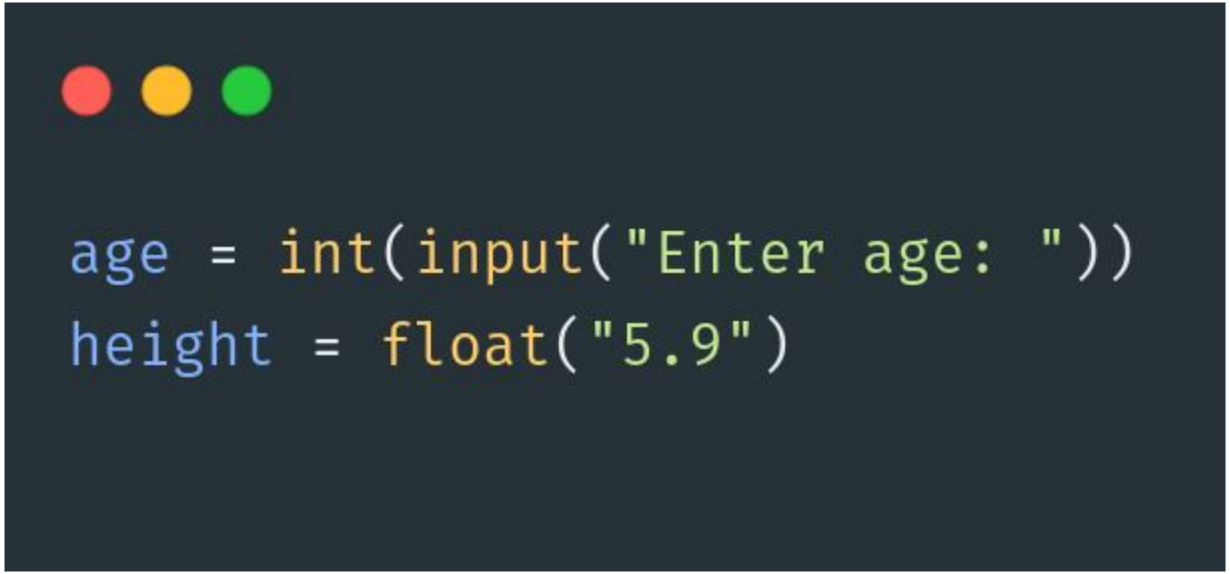
- Input:



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

Type Casting

- Convert one type to another



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

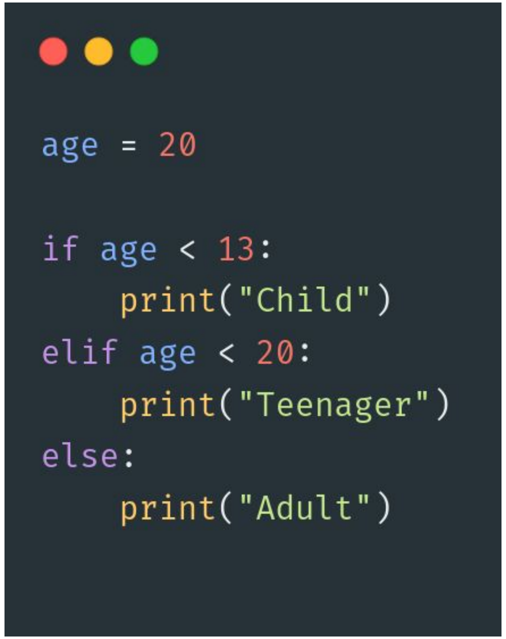
Control flow

- 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)



```python
age = 20

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

Logical Operators

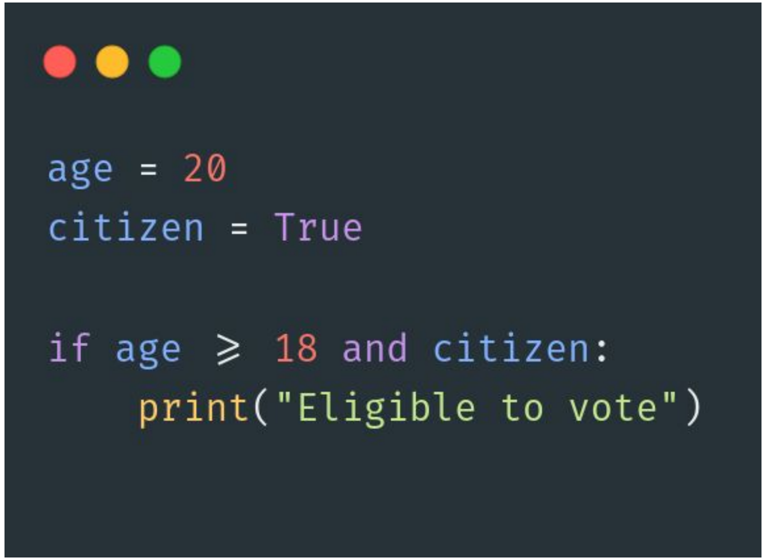
What are Logical Operators?

- 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:

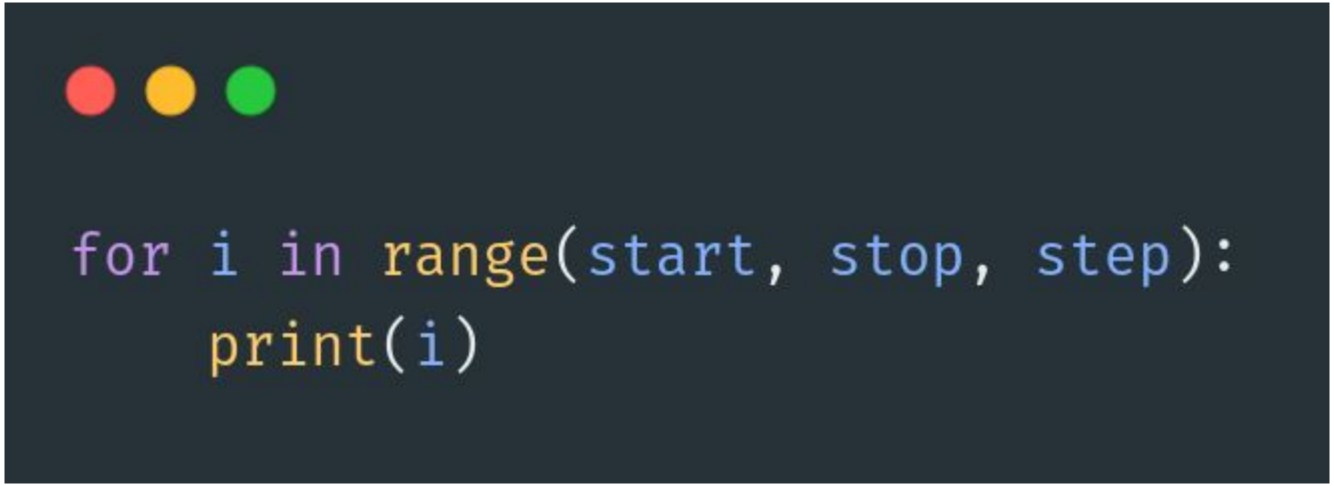
- Reduce code repetition
- Handle large data easily
- Automate repetitive tasks

For Loop

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

Key idea:

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



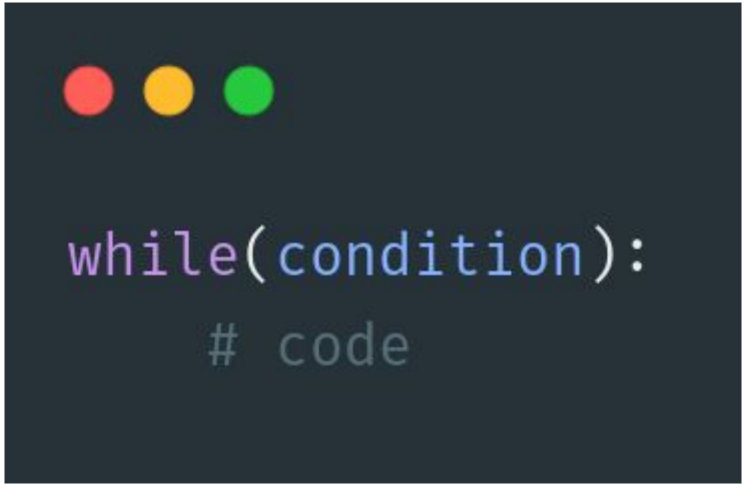
```
for i in range(start, stop, step):
    print(i)
```

while Loop

Runs **while a condition is true**

Key idea:

- Condition-based repetition



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

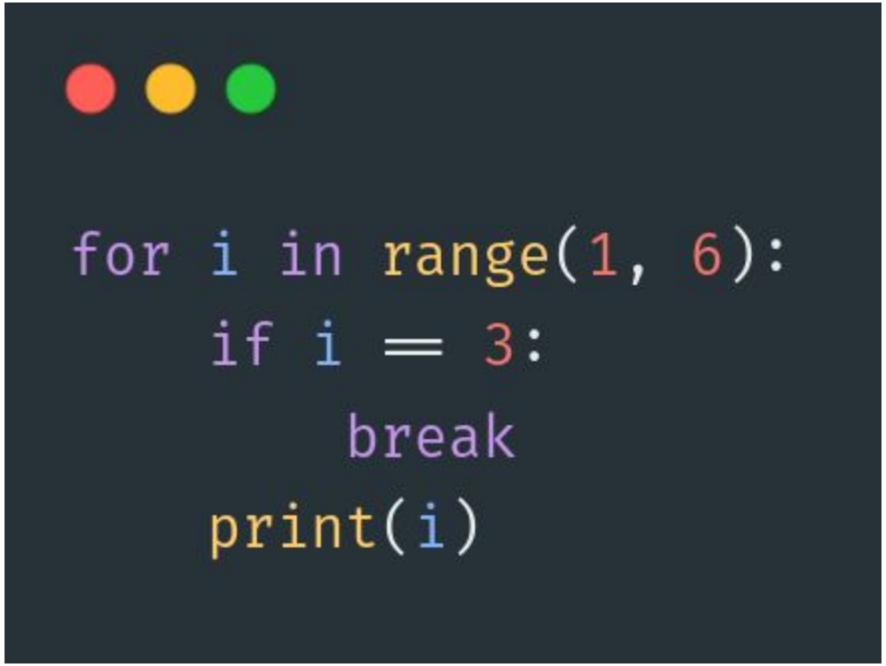
Loop Control Keywords

Break: Stops the loop immediately

Continue : Skips current iteration, moves to next

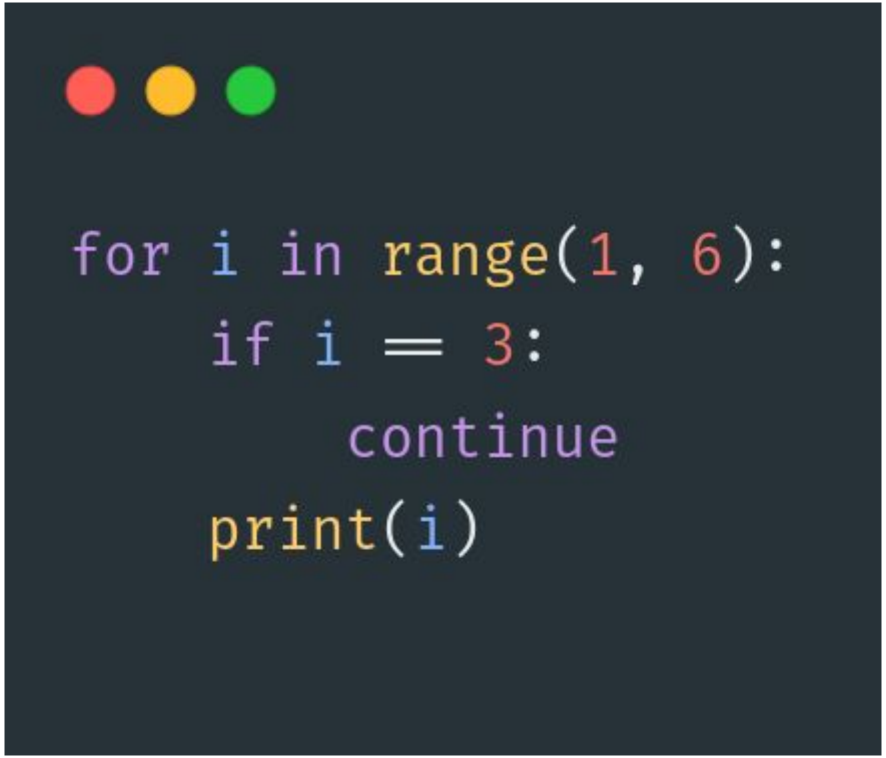
Pass : Does Nothing

Break: Control Keywords



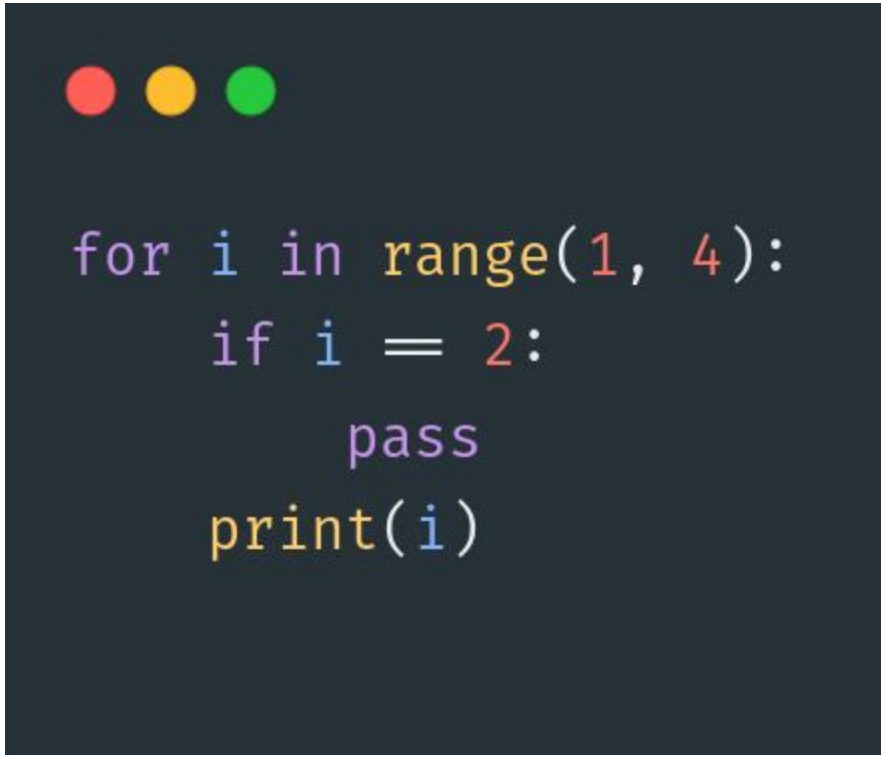
```python
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

Continue : Control Keywords



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

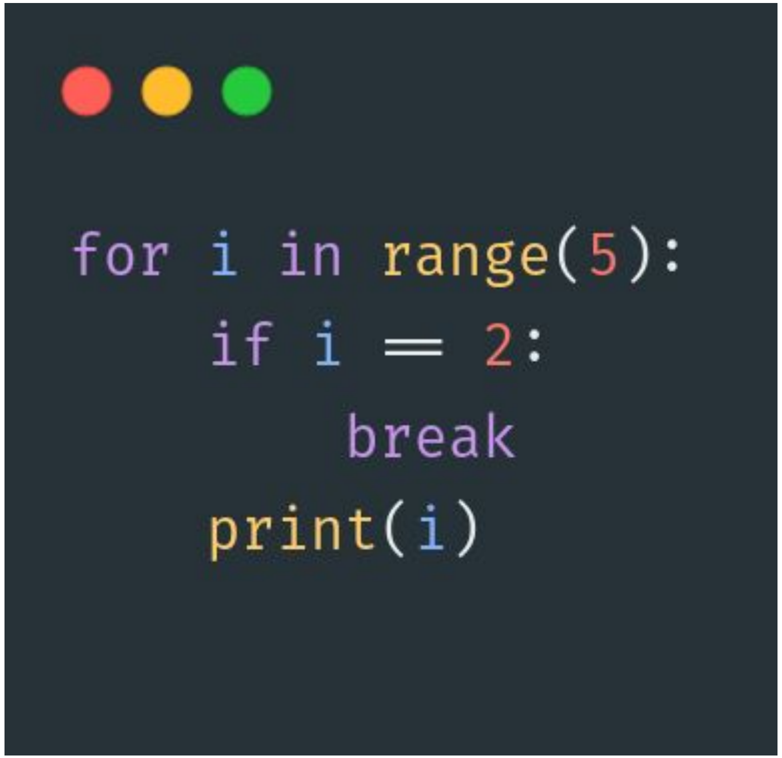
pass : Control Keywords



```python
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



```python
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

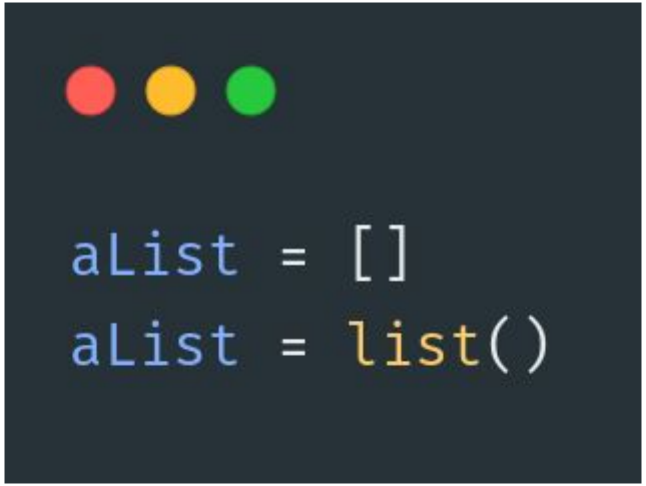
Used to **store and organize data**

Main types:

- List
- Tuple
- Set
- Dictionary

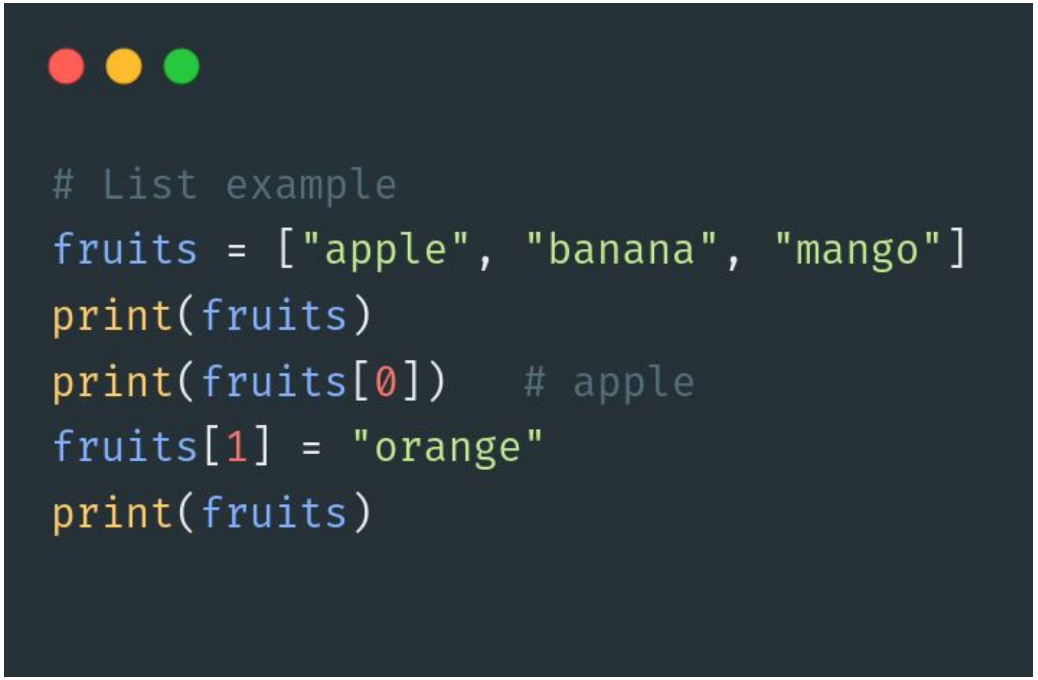
List

- Ordered collection
- Mutable (can be changed)



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

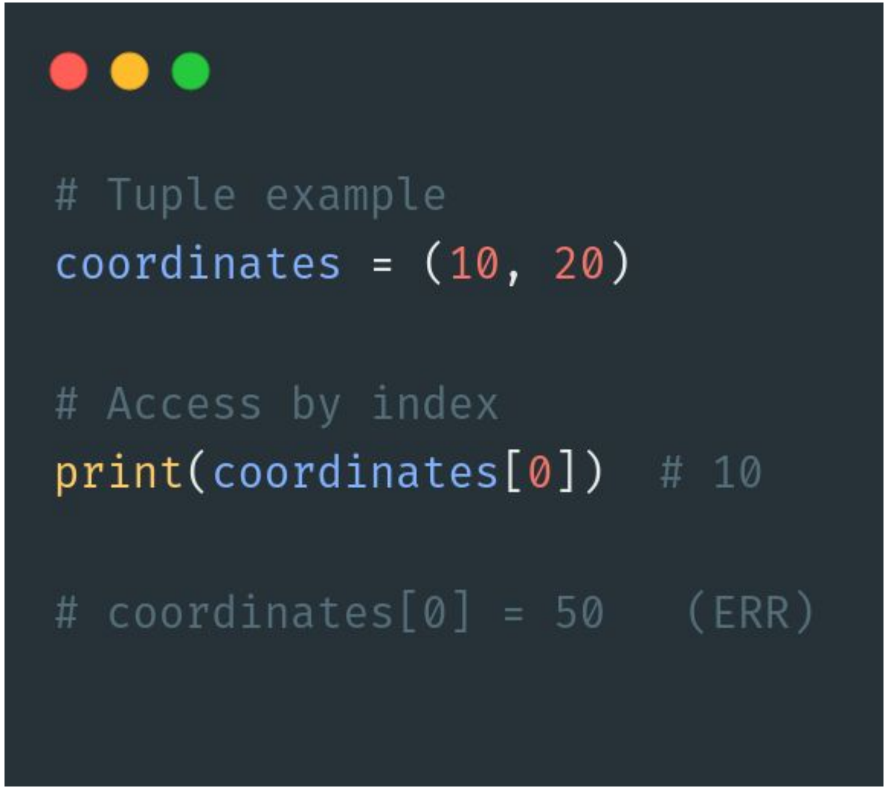
List: Example



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

Tuple

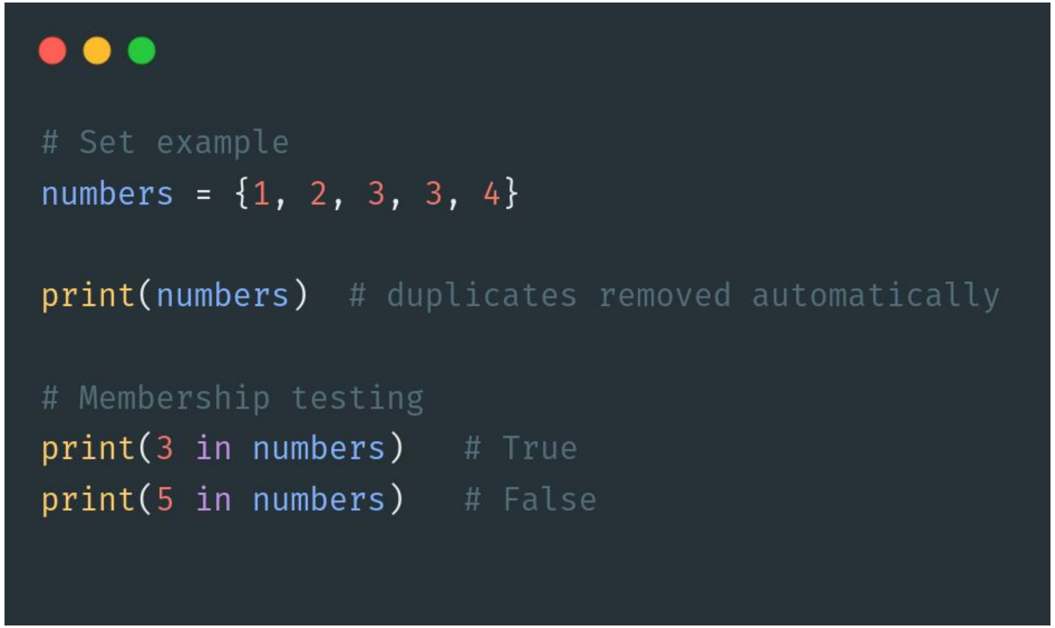
- Ordered collection
- Immutable (cannot be changed)



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

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

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

Set

- Unordered collection
- No Duplicates
- mutable



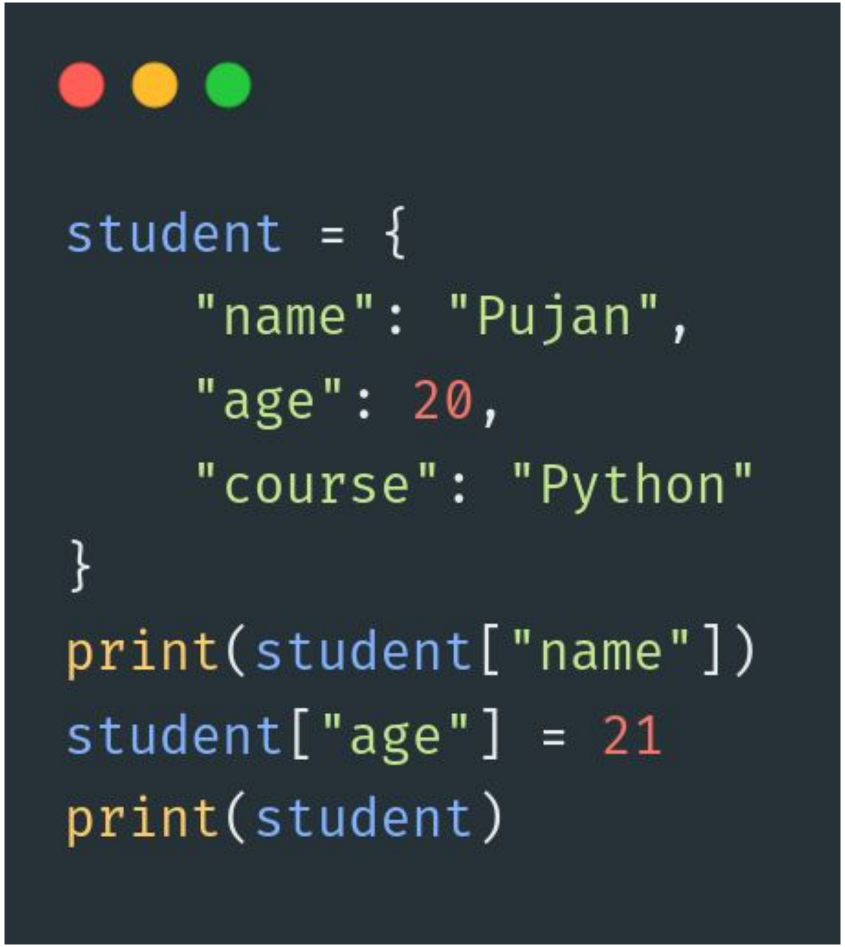
```python
# Set example
numbers = {1, 2, 3, 3, 4}

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



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

Indexing & Slicing

Indexing

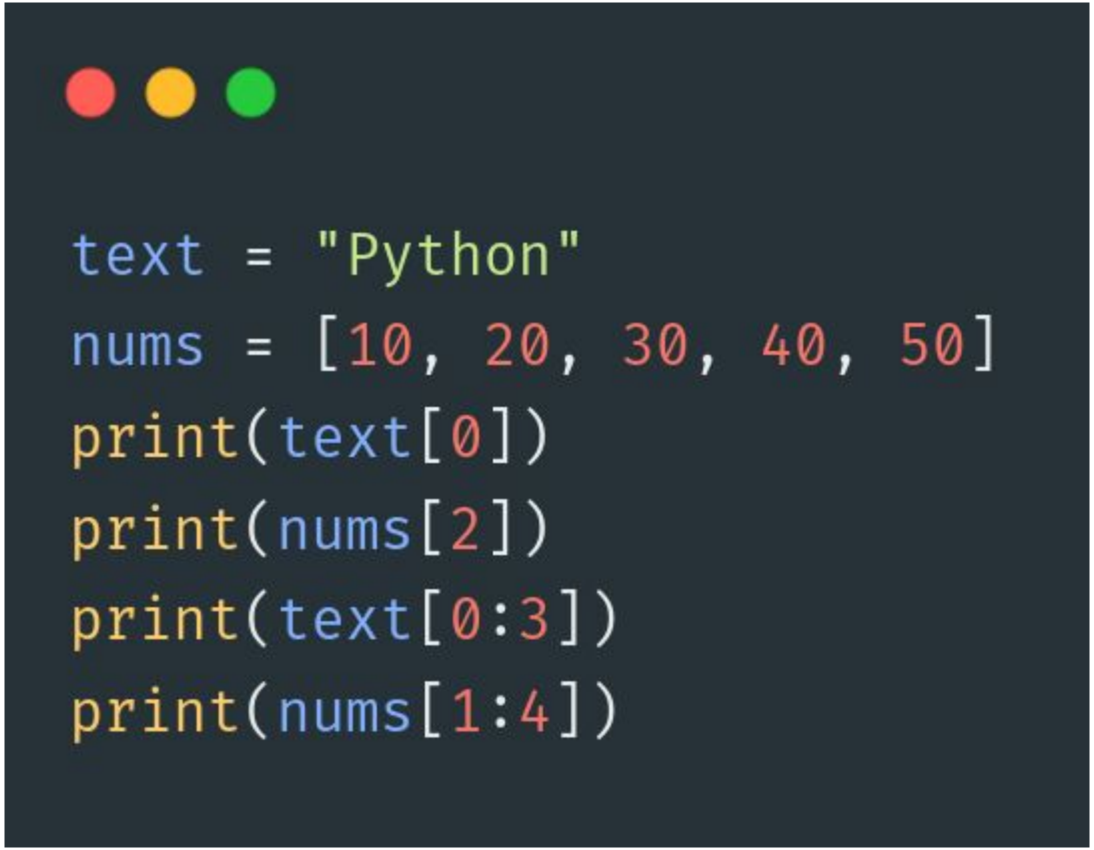
- Access single element

Slicing

- Access a range of elements

Works with:

- Lists
- Tuples
- Strings



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

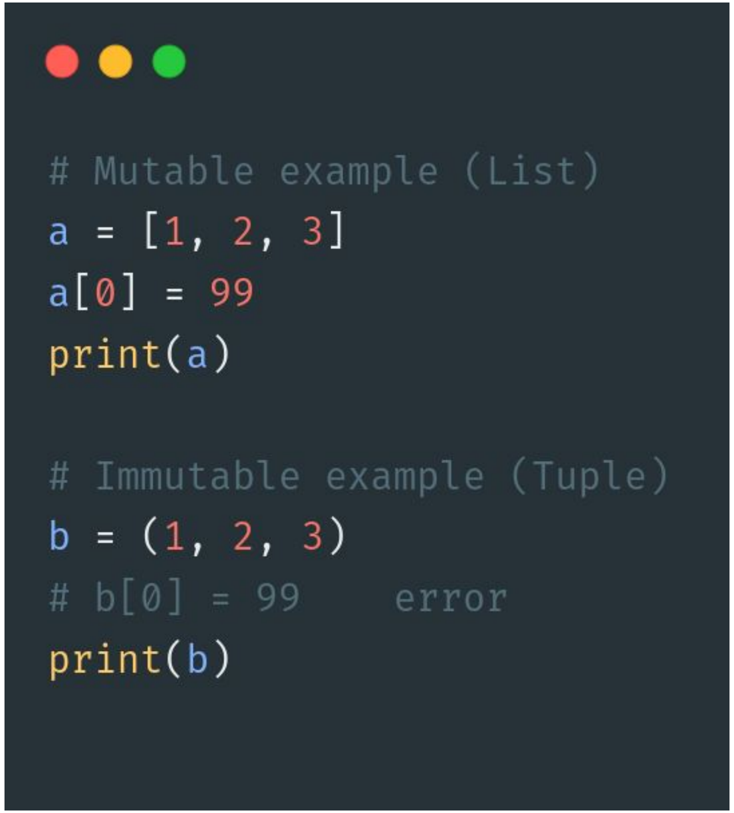
Mutability vs Immutability

Mutable

- Can change after creation
- Example: List, Dictionary

Immutable

- Cannot change after creation
- Example: Tuple, String



```python
# 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

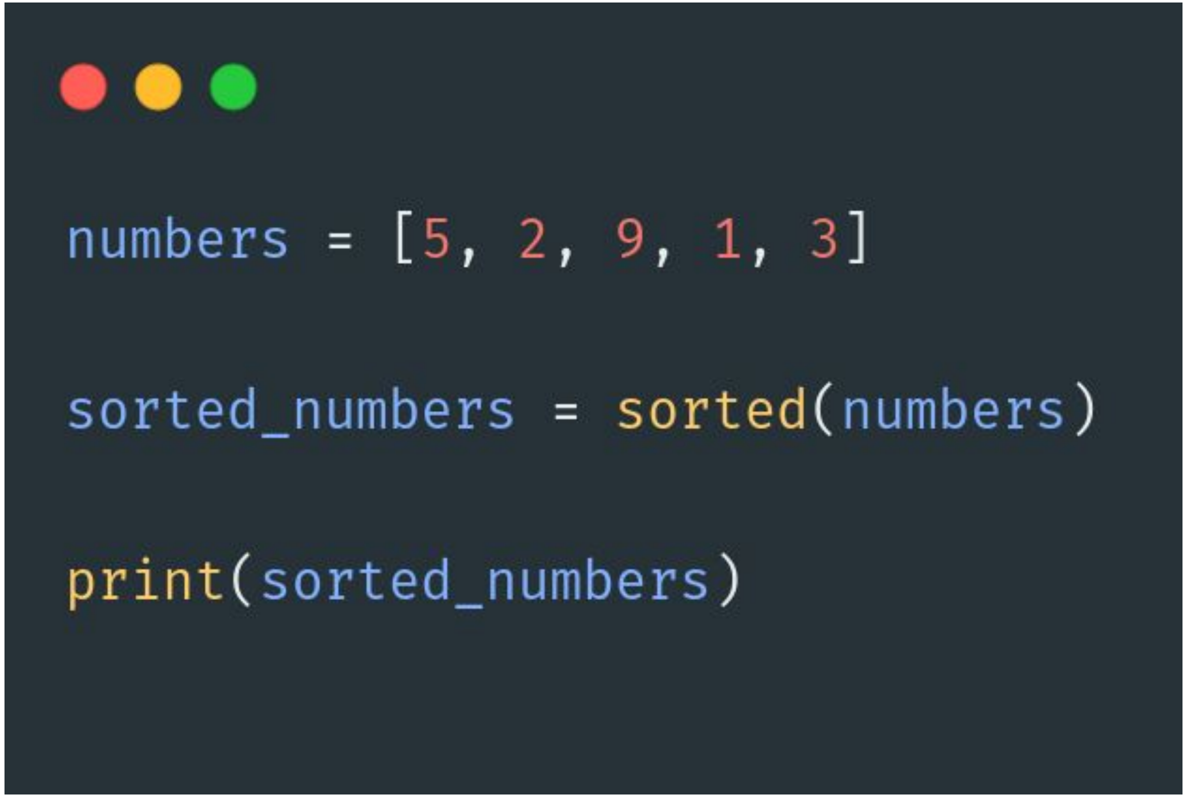
`enumerate()` – index + value

`sum()` – total

`min()` – smallest value

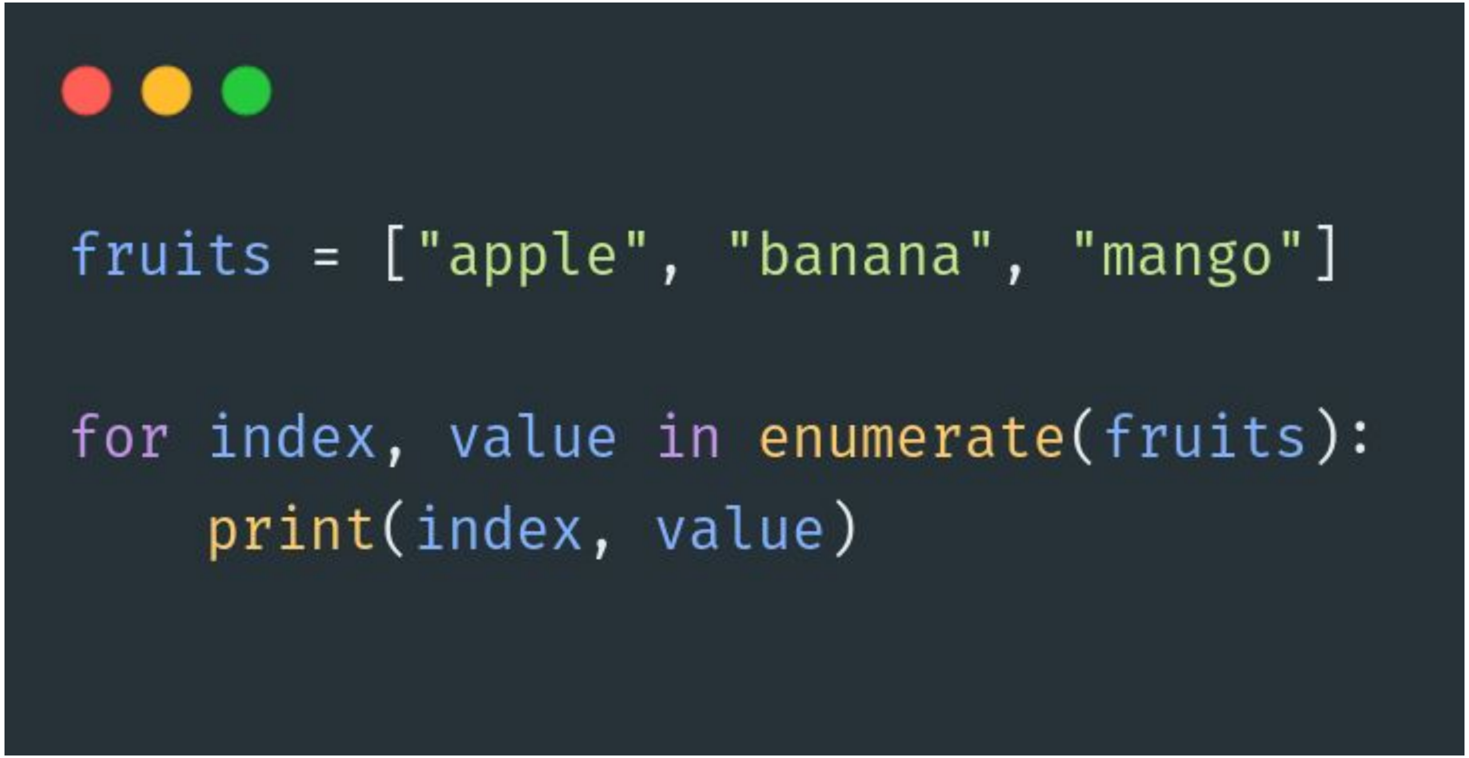
`max()` – largest value

`sorted()` – sort data



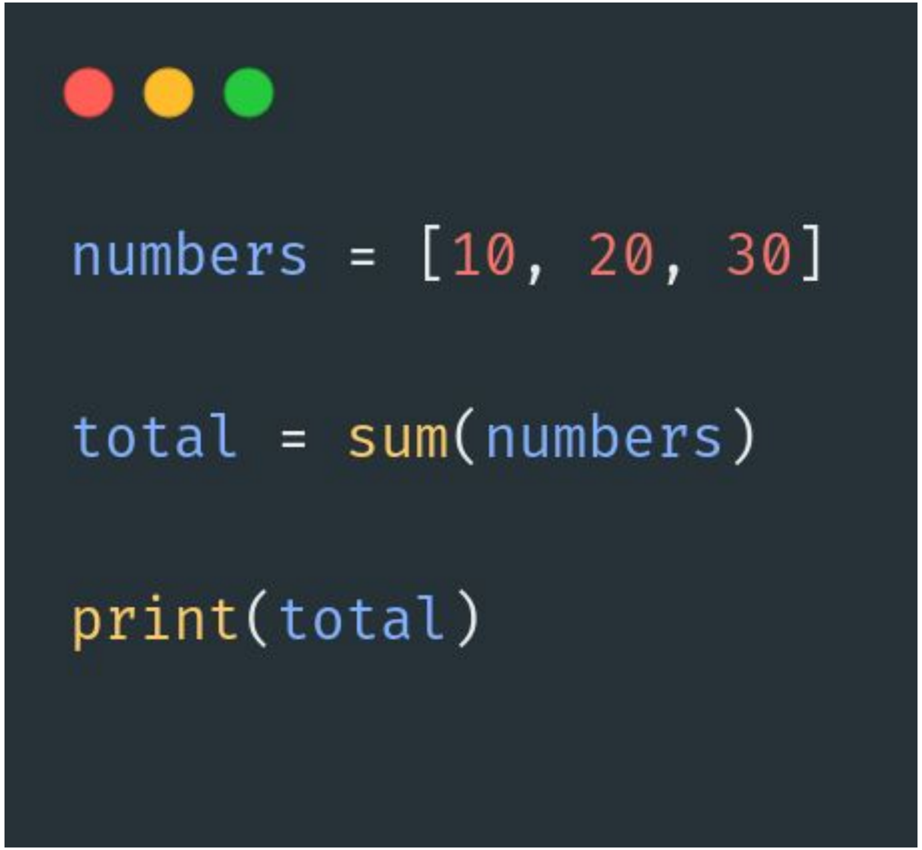
```
numbers = [5, 2, 9, 1, 3]

sorted_numbers = sorted(numbers)

print(sorted_numbers)
```

`enumerate()` – Index + Value



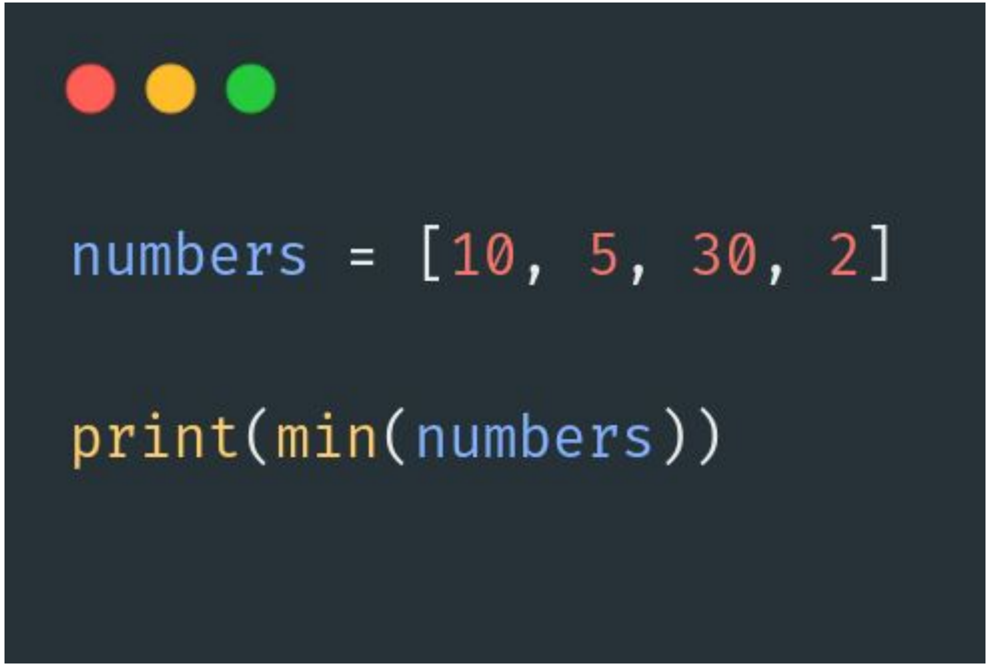
```python
fruits = ["apple", "banana", "mango"]

for index, value in enumerate(fruits):
    print(index, value)
```

`sum()` – Total



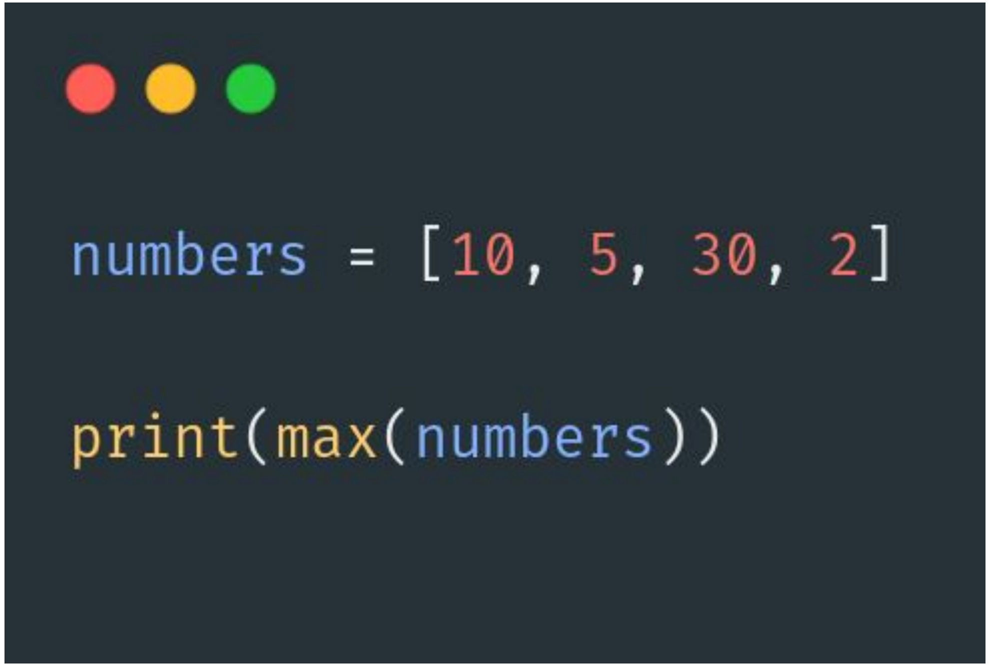
```
numbers = [10, 20, 30]

total = sum(numbers)

print(total)
```

`min()` – Smallest Value from the list



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

print(min(numbers))
```

`max()` – Largest Value from the list



```python
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