Python Programming Essentials (Mini eBook)

□ Index

1. Introduction to Python

Python is a high-level, interpreted programming language known for its readability and simplicity. It is widely used in web development, data science, AI, automation, and more.

Download Python

Go to official website

Open:
 https://www.python.org/downloads/

You will see a **Download Python (latest version)** button.

Click it.

Create and Run Python File

Create file inside myproject folder.

Open Notepad or VS Code and write:

```
print("Hello World")
```

Save it as:

`hello.py`

 Make sure extension is `.py`

Open terminal (cmd) in file folder

Go to folder where file is saved.

For Example:

```
cd myproject
```

□ Run Python file

```
python hello.py
```

□ Output:

```
Hello World
```

Recommended Setup (Best for Beginners)

Install VS Code (Highly recommended)

□ https://code.visualstudio.com/

Then:

1. Install VS Code
2. Install Python extension (inside VS Code)
3. Open `.py` file
4. Click ▶□ Run button

---

2. □ Variables

Definition:

Variables are used to store data values.

Example:

```
x = 10
name = "John"
price = 99.5
```

Explanation:

- `x` stores an integer
- `name` stores a string
- `price` stores a float

Output:

```
print(x, name, price)
```

Output:

```
10 John 99.5
```

---

3. □ Data Types

Common Data Types:

- int
- float
- str
- bool

Example:

```
a = 5
b = 3.2
c = "Hello"
d = True

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

Output:

```
<class 'int'> <class 'float'> <class 'str'> <class 'bool'>
```

---

4. Statements

Statements control the flow of a program.

Python, **`if`**, **`elif`**, and **`else`** are used for decision-making. They allow your program to run different blocks of code based on conditions.

`if` Statement (Basic Condition)

Syntax:

```
if condition:
    # code runs if condition is True
```

Example:

```
age = 18

if age >= 18:
    print("You are an adult")
```

Output:

```
You are an adult
```

Explanation:

- Condition `age >= 18` is checked.
- Since it is **True**, the block runs.

`if-else` Statement

Used when you want **two choices**: one if condition is true, another if false.

Syntax:

```
if condition:
    # runs if True
else:
    # runs if False
```

Example:

```
age = 16

if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

Output:

```
You are a minor
```

Explanation:

- Condition `age >= 18` is False.
- So `else` block runs.

`if-elif-else` Statement (Multiple Conditions)

Used when you have **more than 2 conditions**.

Syntax:

```
if condition1:
    # runs if condition1 is True
elif condition2:
    # runs if condition2 is True
else:
    # runs if none are True
```

Example:

```
marks = 75

if marks >= 90:
    print("Grade: A")
elif marks >= 75:
    print("Grade: B")
elif marks >= 50:
    print("Grade: C")
else:
    print("Fail")
```

Output:

```
Grade: B
```

Explanation step-by-step:

- `marks >= 90` → False
- `marks >= 75` → True → so this block runs
- Remaining conditions are ignored

Important Points

1. Only ONE block runs

Even if multiple conditions are true, only the **first matching block executes**.

---

2. Order matters

```
marks = 80

if marks >= 50:
    print("Pass")
elif marks >= 75:
    print("Distinction")
```

Output:

```
Pass
```

Because `marks >= 50` is checked first and is True.

Real-life Example

```
temperature = 30

if temperature > 35:
    print("Very hot")
elif temperature > 25:
    print("Warm")
else:
    print("Cold")
```

Output:

```
Warm
```

Summary Table

Statement	Use
`if`	single condition
`if-else`	two choices
`if-elif-else`	multiple choices

5. Loops

Loops are used to repeat code.

In Python, **loops** are used to repeat a block of code multiple times. One of the most commonly used loops is the **`for` loop**.

□ What is a `for` loop?

A **`for` loop** is used to iterate over a sequence such as:

- list
- string
- tuple
- range()

It runs the block of code **once for each item** in the sequence.

`for` loop using `range()`

Syntax:

```
for variable in range(start, stop):
    # code block
```

Example 1:

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

Output:

```
0
1
2
3
4
```

Explanation:

- `range(5)` generates numbers from **0 to 4** (5 is excluded)
- Loop runs 5 times
- `i` takes each value one by one
- `print(i)` displays each value

`for` loop with start and stop

Example 2:

```
for i in range(1, 6):
    print(i)
```

Output:

```
1
2
3
4
5
```

Explanation:

- Starts from **1**
- Ends at **5**
- 6 is NOT included
- Useful when you don't want starting from 0

`for` loop with step value

Example 3:

```
for i in range(0, 10, 2):
    print(i)
```

Output:

```
0
2
4
6
8
```

Explanation:

- Start = 0
- Stop = 10 (excluded)
- Step = 2 (jump by 2)
- So it prints every second number

`for` loop with a list

Example 4:

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

for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
mango
```

Explanation:

- Loop goes through each item in list
- `fruit` holds one value at a time
- Prints each fruit

`for` loop with string

Example 5:

```
for letter in "Python":
    print(letter)
```

Output:

```
P
y
t
h
o
n
```

Explanation:

- String is treated like a sequence of characters
- Loop prints each character separately

□ Key Points

- `for` loop is used when you know how many times to repeat
- Works with **range, list, string, tuple**
- Stops automatically when sequence ends

Python `while` Loop

A **`while` loop** is used in Python to repeat a block of code **as long as a condition is True**.

Unlike a `for` loop (which runs a fixed number of times), a `while` loop runs **until the condition becomes False**.

□ Syntax

```
while condition:
    # code block
```

`while` loop example

Example:

```
i = 1

while i <= 5:
    print(i)
    i = i + 1
```

Output:

```
1
2
3
4
5
```

□ Explanation step-by-step:

Step	Value of i	Condition `i <= 5`	Action
1	1	True	print 1
2	2	True	print 2
3	3	True	print 3
4	4	True	print 4
5	5	True	print 5
6	6	False	loop stops

✔ Important: `i = i + 1` is necessary, otherwise loop will run forever.

`while` loop with countdown:-

Example:

```
count = 5

while count > 0:
    print(count)
    count = count - 1

print("Done!")
```

Output:

```
5
4
3
2
1
Done!
```

□ Explanation:

- Starts at 5
- Decreases by 1 each time
- Stops when count becomes 0

6. □ Functions

Python Functions (with Syntax, Explanation & Output)

A **function** in Python is a **block of reusable code** that performs a specific task. Instead of writing the same code again and again, you define a function once and use it whenever needed.

□ Why use functions?

- Reusability (write once, use many times)
- Makes code clean and organized
- Easy debugging
- Better readability

□ Function Syntax

```
def function_name(parameters):
    # block of code
    return value   # optional
```

□ Explanation of syntax:

Part	Meaning
`def`	keyword to define a function
`function_name`	name of function
`parameters`	input values (optional)
`:`	starts function block
`return`	sends result back (optional)

Simple Function (No Parameters)

Example:

```
def greet():
    print("Hello, Welcome to Python!")

greet()
```

Output:

```
Hello, Welcome to Python!
```

□ Explanation:

- Function `greet()` is defined
- It has no input parameters
- When called, it prints message

Function with Parameters

Example:

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

greet("Amit")
greet("Sara")
```

Output:

```
Hello Amit
Hello Sara
```

□ Explanation:

- `name` is a parameter

- Function accepts input value
- Each call passes different value

Function with Return Value

Example:

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

result = add(5, 3)
print(result)
```

Output:

```
8
```

□ Explanation:

- Function takes 2 numbers
- Adds them and returns result
- `return` sends value back to variable `result`

Function with Condition

Example:

```
def check_even(number):
    if number % 2 == 0:
        return "Even"
    else:
        return "Odd"

print(check_even(10))
print(check_even(7))
```

Output:

```
Even
Odd
```

□ Explanation:

- Function checks number
- Uses `if-else`
- Returns result based on condition

Function with Default Parameter

Example:

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

greet()
greet("Rahul")
```

Output:

```
Hello Guest
Hello Rahul
```

□ Explanation:

- Default value is `"Guest"`
- If no input is given, default is used

□ Key Points

- Functions are defined using `def`
- Can take inputs (parameters)
- Can return output using `return`
- Can be reused multiple times

7. □ Lists

Python List

A **list** is an ordered, changeable (mutable) collection that allows duplicate values.

□ Syntax:

```
my_list = [item1, item2, item3]
```

□ Example:

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

print(fruits)
print(fruits[0])
fruits[1] = "grapes"
print(fruits)
```

□ Output:

```
['apple', 'banana', 'mango']
apple
['apple', 'grapes', 'mango']
```

□ Explanation:

- Ordered → keeps index
- Mutable → can change values
- Allows duplicates

8. □ Tuples

Python Tuple

A **tuple** is ordered but **unchangeable (immutable)**.

□ Syntax:

```
my_tuple = (item1, item2, item3)
```

□ Example:

```
colors = ("red", "green", "blue")

print(colors)
print(colors[1])
```

□ Output:

```
('red', 'green', 'blue')
green
```

□ Trying to change tuple:

```
colors[0] = "yellow"
```

□ Output:

```
TypeError: 'tuple' object does not support item assignment
```

□ Explanation:

- Ordered
- Immutable (cannot change)
- Faster than list

9. □ Sets

Python Set

A **set** is an unordered collection with **no duplicates**.

□ Syntax:

```
my_set = {item1, item2, item3}
```

□ Example:

```
numbers = {1, 2, 3, 3, 4}

print(numbers)
numbers.add(5)
print(numbers)
```

□ Output:

```
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
```

□ Explanation:

- Removes duplicates automatically
- Unordered (no index)
- Supports add/remove

10. □ Dictionaries

Python Dictionary

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

□ Syntax:

```
my_dict = {
    "key1": "value1",
    "key2": "value2"
}
```

□ Example:

```
student = {
    "name": "Rahul",
    "age": 20,
    "city": "Mumbai"
}

print(student)
print(student["name"])

student["age"] = 21
print(student)
```

□ Output:

```
{'name': 'Rahul', 'age': 20, 'city': 'Mumbai'}
Rahul
{'name': 'Rahul', 'age': 21, 'city': 'Mumbai'}
```

□ Explanation:

- Data stored as **key: value**
- Mutable (can change values)
- Keys must be unique

Comparison Table

Feature	List	Tuple	Set	Dictionary
Ordered	Yes	Yes	No	Yes (Python 3.7+)
Mutable	Yes	No	Yes	Yes
Duplicates	Allowed	Allowed	Not allowed	Keys not allowed duplicate
Indexing	Yes	Yes	No	Key-based

Real-Life Analogy

Type	Real-world example
List	Shopping list
Tuple	Aadhaar number (fixed)
Set	Unique IDs in a class
Dictionary	Phone book (name → number)

11. □ Classes and Objects

Definition:

Class is a blueprint; object is an instance.

Example:

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

    def greet(self):
        print("Hello", self.name)

p1 = Person("Alice")
p1.greet()
```

Output:

```
Hello Alice
```

---

12. Inheritance

Definition:

One class inherits properties from another.

Example:

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

class Dog(Animal):
    def bark(self):
        print("Dog barks")

d = Dog()
d.speak()
d.bark()
```

Output:

```
Animal speaks
Dog barks
```

Explanation:

- `Dog` inherits from `Animal`
- Can use parent + its own methods

NumPy, Pandas, and Matplotlib in Python with examples and outputs.

These three are the **core libraries for Data Science and Data Analysis**.

□ 1. NumPy (Numerical Python)

□ What is NumPy?

NumPy is used for:

- Fast mathematical operations
- Working with arrays (faster than Python lists)
- Linear algebra, statistics, etc.

□ Import NumPy:

```
import numpy as np
```

□ Example 1: Creating an array

```
import numpy as np

arr = np.array([1, 2, 3, 4])
print(arr)
```

□ Output:

```
[1 2 3 4]
```

□ Example 2: Array operations

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)
print(a * b)
```

□ Output:

```
[5 7 9]
[4 10 18]
```

□ Explanation:

- NumPy performs **element-wise operations**
- Much faster than Python lists

□? 2. Pandas

□ What is Pandas?

Pandas is used for:

- Data analysis
- Working with tables (like Excel)
- Data cleaning and manipulation

□ Import Pandas:

```
import pandas as pd
```

□ Example 1: Series (1D data)

```
import pandas as pd

data = pd.Series([10, 20, 30])
print(data)
```

□ Output:

```
0    10
1    20
2    30
dtype: int64
```

□ Example 2: DataFrame (2D table)

```
import pandas as pd

data = {
    "Name": ["Amit", "Sara", "John"],
    "Age": [20, 22, 21]
}

df = pd.DataFrame(data)
print(df)
```

□ Output:

```
   Name  Age
0  Amit   20
1  Sara   22
2  John   21
```

□ Explanation:

- Series → single column
- DataFrame → table (rows + columns)
- Very useful for real datasets

Below is a **complete beginner-friendly Pandas example using a CSV file**, including:

- □ Reading CSV data
- □ Cleaning data
- □ Filtering data
- □ Removing missing/duplicate values
- □ Visualizing data with Matplotlib
- □ Outputs explained step-by-step

□ 1. Create a Sample CSV File

Let's assume we have a file named:

students.csv

```
Name,Age,Marks,City
Amit,20,85,Mumbai
Sara,21,90,Delhi
John,19,,Pune
Raj,,70,Mumbai
Sara,21,90,Delhi
Neha,22,95,Bangalore
```

---

2. Import Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
```

---

3. Read CSV File

```
df = pd.read_csv("students.csv")
print(df)
```

Output:

```
   Name   Age  Marks       City
0  Amit  20.0   85.0     Mumbai
1  Sara  21.0   90.0      Delhi
2  John  19.0    NaN       Pune
3   Raj   NaN   70.0     Mumbai
```

```
4   Sara   21.0    90.0       Delhi
5   Neha   22.0    95.0   Bangalore
```

□ Explanation:

- `read_csv()` loads data into a DataFrame
- Missing values are shown as `NaN`

□ 4. Data Cleaning (Remove missing values)

```
cleaned_df = df.dropna()
print(cleaned_df)
```

□ Output:

```
   Name    Age  Marks        City
0  Amit   20.0   85.0      Mumbai
1  Sara   21.0   90.0       Delhi
5  Neha   22.0   95.0   Bangalore
```

□ Explanation:

- `dropna()` removes rows with missing values
- John and Raj rows are removed

□ 5. Remove Duplicate Rows

```
no_duplicates = cleaned_df.drop_duplicates()
print(no_duplicates)
```

□ Output:

```
   Name   Age  Marks       City
0  Amit  20.0   85.0     Mumbai
1  Sara  21.0   90.0      Delhi
5  Neha  22.0   95.0  Bangalore
```

□ Explanation:

- Duplicate Sara row is removed

□ 6. Filtering Data (Condition-based)

□ Students with Marks > 85

```
filtered = no_duplicates[no_duplicates["Marks"] > 85]
print(filtered)
```

□ Output:

```
   Name   Age  Marks       City
1  Sara  21.0   90.0      Delhi
5  Neha  22.0   95.0  Bangalore
```

□ Explanation:

- Only rows where Marks > 85 are selected

□ 7. Filter by City

□ Only Mumbai students

```
mumbai_students = no_duplicates[no_duplicates["City"] == "Mumbai"]
print(mumbai_students)
```

□ Output:

```
   Name   Age  Marks    City
0  Amit  20.0   85.0  Mumbai
```

□ 8. Data Visualization (Bar Chart)

□ Marks of students

```
plt.bar(no_duplicates["Name"], no_duplicates["Marks"])

plt.title("Student Marks")
plt.xlabel("Name")
plt.ylabel("Marks")

plt.show()
```

□ Output:

A **bar graph** will be displayed:

- Amit → 85
- Sara → 90
- Neha → 95

□ Explanation:

- `bar()` creates bar chart
- X-axis = Names
- Y-axis = Marks

□ 9. Average Marks

```
avg_marks = no_duplicates["Marks"].mean()
print("Average Marks:", avg_marks)
```

□ Output:

```
Average Marks: 90.0
```

□ Explanation:

- `.mean()` calculates average of Marks

□ FULL SUMMARY OF OPERATIONS

Task	Function
Read CSV	`read_csv()`
Remove missing data	`dropna()`
Remove duplicates	`drop_duplicates()`
Filter data	`df[df["col"] condition]`
Average	`mean()`
Visualization	`matplotlib.pyplot`