



PYTHON

Fundamentals & Core Concepts

Based on My Personal Learning Journey

Rohit Survase

PYTHON

Fundamentals & Core Concepts

Based on My Personal Learning Journey

Written by

Rohit Survase

A book based on my personal Python learning notes, created during my journey of learning Python as my first programming language. This book focuses on Python Fundamentals and Core Concepts.

COPYRIGHT

Copyright © 2026 Rohit Survase

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, transmitted, or distributed in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of the author, except for brief quotations used in reviews or educational references.

This book is based on the author's personal Python learning journey and notes created while studying Python as a complete beginner. It is intended for educational and reference purposes only and does not claim to cover the complete Python programming language or its entire ecosystem.

Every effort has been made to ensure the accuracy of the information presented in this book. However, the author assumes no responsibility for any errors, omissions, or consequences arising from the use of the material contained herein.

First Edition: 2026

Author: Rohit Survase

Language: English + Marathi Mix

Country of Publication: India

DEDICATION

*Dedicated to Python—my first programming language,
which introduced me to the world of programming,
problem-solving, and endless possibilities.*

*This book is also dedicated to every beginner who starts
their coding journey with curiosity, determination, and the
courage to keep learning.*

PREFACE

Learning to program is not just about memorizing syntax—it is about learning how to think logically, solve problems, and turn ideas into meaningful solutions. Programming शिकताना concepts समजून घेणे आणि त्यांचा practical use करणे हे syntax लक्षात ठेवण्याइतकेच important आहे.

This book is a result of my personal Python learning journey. Python ही माझी first programming language होती, आणि learning करताना मी तयार केलेल्या notes आणि experiences वर या book ची foundation आहे. The concepts have been organized in a way that helped me understand Python more clearly, instead of simply collecting or copying information.

The main focus of this book is Python Fundamentals and Core Concepts. हा book complete Python reference नसून, programming ची सुरुवात करणाऱ्या beginners साठी एक strong आणि structured foundation तयार करण्याचा प्रयत्न आहे. प्रत्येक topic simple आणि organized manner मध्ये explain करण्यावर भर दिला आहे.

This book covers the essential concepts required to begin programming with Python. Variables, data types, operators, control flow, functions, collections, and other important concepts are introduced step-by-step, so that learners can gradually build their understanding and confidence.

I hope this book helps learners make their first steps into programming easier and more enjoyable. Concepts फक्त वाचण्याऐवजी ते समजून घेणे, code लिहिणे आणि mistakes मधून शिकणे ही programming शिकण्याची खरी process आहे.

This book marks the beginning of my programming journey. माझ्यासाठी हा फक्त एक book project नसून, मी शिकलेल्या गोष्टी organize करून इतरांसोबत share करण्याचा एक छोटासा प्रयत्न आहे. I look forward to continuing to learn, build, and share more knowledge in the future.

Rohit Survase

ABOUT THIS BOOK

This book is based on my personal Python learning journey and the notes I created while learning Python as my first programming language. Python शिकताना मला समजलेले concepts simple आणि organized manner मध्ये मांडण्याचा या book मध्ये प्रयत्न केला आहे.

The primary goal of this book is to present Python Fundamentals and Core Concepts in a simple, clear, and well-organized manner. Every topic has been arranged to make learning easier for beginners and to serve as a quick reference for study and revision.

This book covers the essential concepts of Core Python, including variables, data types, operators, control flow, functions, collections, object-oriented programming, file handling, exception handling, modules, and other fundamental topics.

It is important to note that this book is not a complete guide to the entire Python ecosystem. Advanced areas such as web development, data science, machine learning, artificial intelligence, and other specialized domains are beyond the scope of this book. म्हणजेच, या book चा मुख्य focus Python Fundamentals and Core Concepts वर आहे.

Whether you are beginning your programming journey or revising the fundamentals, this book is designed to provide a strong foundation in Python programming. Concepts समजून घेऊन त्यांचा practical use करता यावा आणि पुढील advanced learning साठी strong base तयार व्हावा, हा या book चा मुख्य उद्देश आहे.

WHO THIS BOOK IS FOR

This book is designed for anyone who wants to build a strong foundation in Python programming. Python शिकण्याची सुरुवात करणाऱ्या learners साठी हा book simple आणि structured approach देतो.

It is especially suitable for:

- Beginners who are learning Python as their first programming language.
- Students who want clear and well-organized notes for study and revision.
- Self-learners looking for a structured approach to understanding Python fundamentals.
- Anyone who wants a quick reference to the core concepts of Python.

This book assumes no prior programming experience. प्रत्येक topic simple आणि logical manner मध्ये presented केला आहे, making it easier to understand and apply the concepts.

If you are looking for advanced topics such as web development, data science, machine learning, or artificial intelligence, this book should be considered a starting point rather than a complete guide. या book चा मुख्य purpose Python fundamentals समजून घेण्यासाठी strong base तयार करणे हा आहे.

The goal of this book is to help readers develop confidence in Python Fundamentals and prepare them for more advanced areas of programming. Concepts समजून घेऊन ते practically apply करता यावेत आणि पुढे advanced topics शिकण्यासाठी learners तयार व्हावेत, हा या book चा मुख्य उद्देश आहे.

HOW TO USE THIS BOOK

To get the most value from this book, it is recommended that you study each chapter in sequence. प्रत्येक topic previous chapters मध्ये introduced केलेल्या concepts वर build होतो, creating a strong and logical learning path.

As you read, do not simply memorize the syntax. Instead, focus on understanding the purpose of each concept and practice writing the examples on your own. Programming is a practical skill, and consistent practice is essential for mastering it.

This book can be used in two ways:

- * As a step-by-step learning guide for beginners.
- * As a quick reference for revising Python Fundamentals and Core Concepts.

Whenever you complete a chapter, take some time to experiment with the code, modify the examples, and solve small problems on your own. Code स्वतः लिहून, examples मध्ये changes करून आणि problems solve करून शिकणे हा programming skills improve करण्याचा effective way आहे.

Most importantly, enjoy the learning process. Every expert programmer was once a beginner, and every line of code you write brings you one step closer to becoming a better developer. Learning slowly is fine, but keep practicing consistently.

Python Learning Roadmap

This book follows a structured path, taking you from the basics of Python to a solid foundation in object-oriented programming. प्रत्येक stage मागच्या stage मध्ये शिकलेल्या concepts वर build होतो.

- Foundations — Python syntax, variables, data types, आणि basic input/output
- Data Structures — Strings, Lists, Tuples, Sets, आणि Dictionaries
- Control Flow — Conditional statements आणि loops
- Functions — Defining functions, recursion, आणि scope
- Functional Tools — Lambda, map, filter, आणि reduce
- Error Handling — Exceptions आणि safer programs लिहिणे
- Modules & Environment — Code organize करणे आणि packages manage करणे
- Object-Oriented Programming — Classes, objects, आणि core principles
- Best Practices — Clean, readable, आणि Pythonic code लिहिणे

Follow this roadmap stage by stage, आणि प्रत्येक stage सोबत तुमची Python ची understanding आणि confidence improve होत जाईल. By the end, you will have a strong and practical foundation in Core Python.

Table of Contents

Chapter 1 — Getting Started with Python.....	1
Chapter 2 — Variables, Data Types, Type Casting & Operators	2
Chapter 3 — Strings	8
Chapter 4 — Lists	18
Chapter 5 — Tuples.....	23
Chapter 6 — Sets	26
Chapter 7 — Dictionaries.....	34
Chapter 8 — Conditional Statements.....	42
Chapter 9 — Loops	47
Chapter 10 — Functions, Recursion & Variable Scope	54
Chapter 11 — Functional Programming Tools	70
Chapter 12 — Exception Handling.....	77
Chapter 13 — Modules, Packages & Virtual Environments	86
Chapter 14 — File Handling & OS Operations	97
Chapter 15 — Object-Oriented Programming.....	109
Chapter 16 — Python Coding Style & Best Practices.....	137
Chapter 17— Generators	151
Chapter 18— Walrus Operator (:=)	158
Chapter 19— Time Module	162
Chapter 20— Requests Module & HTTP.....	167
Chapter 21— Shutil Module.....	176
Chapter 22— BeautifulSoup / bs4 (Web Scraping)	182
Chapter 23— Operator Overloading	190
Chapter 24— PyPDF	197

Chapter 1 — Getting Started with Python

What is Programming?

Programming म्हणजे Computer ला specific task perform करण्यासाठी instructions देण्याची process. या instructions च्या collection ला Program म्हणतात. Computer ला Marathi किंवा English भाषा समजत नाही, त्यामुळे आपण Programming Language वापरून त्याला instructions देतो.

Why Learn Python?

Python ही आजच्या काळातील सर्वात popular आणि beginner-friendly Programming Language आहे. तिचे syntax simple असल्यामुळे ती शिकणे सोपे जाते. Python चा वापर Web Development, Data Science, Artificial Intelligence (AI), Machine Learning, Automation आणि इतर अनेक fields मध्ये केला जातो.

Features of Python

Easy to Learn - Beginners साठी सोपी भाषा.

Simple Syntax - Code लिहिणे आणि वाचणे सोपे.

Cross Platform - Windows, Linux आणि macOS वर चालते.

Open Source - Free आणि सर्वासाठी उपलब्ध.

Large Community - जगभरातील Developers कडून Support मिळतो.

First Python Program

```
print("Hello, World!") → Hello, World!
```

print() Function चा उपयोग screen वर output display करण्यासाठी केला जातो.

Chapter 2 — Variables, Data Types, Type Casting & Operators

i. Variables

Variable म्हणजे **memory मधला एक container** जो data hold करतो. समजायला easy उदाहरण – घरात Sugar साठी वेगळा डबा, Salt साठी वेगळा डबा. तसंच variable म्हणजे data साठी वेगळी जागा memory मध्ये.

```
a = 1
b = True
c = "Rohit"
```

ii. Data Type

Data type म्हणजे – **variable मध्ये कोणत्या प्रकारचा data stored आहे** ते सांगणारी गोष्ट.

हे का लागतं? कारण Python ला माहीत असणं गरजेचं आहे की तू number store केलंयस की text – नाहीतर operations करताना **error येतो**.

Type बघायला type() हे function वापरतात:

```
type(42)      → <class 'int'>
type("hello") → <class 'str'>
```

iii. Type Casting

1. int – पूर्ण संख्या (decimal नाही)

```
a = 42
a = -8
a = 0
```

2. float – दशांश संख्या (decimal असते)

```
b = 3.14  
b = -8.2
```

3. complex – Real + Imaginary असा number

```
e = complex(6, 2) → (6+2j)
```

4. str (String) – कोणताही text, quotes मध्ये लिहितात

```
c = "Hello World"  
c = 'Python'
```

⚠ Important – "123" हे str आहे आणि 123 हे int आहे. दोन्ही दिसायला सारखे पण आतून वेगळे!

5. bool (Boolean) – फक्त दोनच values: True किंवा False

iv. Operators

Operator म्हणजे असे Symbols जे काही Operation करण्यासाठी वापरले जातात.

```
5 + 3
```

➤ इथे + हा Operator आहे.

Types of Operators

1. Arithmetic Operators

हे Mathematical Calculations करण्यासाठी वापरतात.

```
a = 10
```

```
b = 3
```

```
print(a + b) → 13
```

```
print(a - b) → 7
```

```
print(a * b) → 30
```

```
print(a / b) → 3.333333333333335
```

```
print(a // b) → 3
```

```
print(a % b) → 1
```

```
print(a ** b) → 1000
```

Operator	Meaning	Example
+	Addition	$5 + 2 = 7$
-	Subtraction	$5 - 2 = 3$
*	Multiplication	$5 * 2 = 10$
/	Division	$5 / 2 = 2.5$
//	Floor Division	$5 // 2 = 2$
%	Modulus (Remainder)	$5 \% 2 = 1$
**	Power	$2 ** 3 = 8$

2. Comparison Operators

हे दोन Values Compare करतात आणि Result True किंवा False मध्ये देतात.

Operator	Meaning
==	Equal
!=	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal
<=	Less Than or Equal

```
a = 10  
b = 20  
  
print(a == b) → False  
print(a < b) → True  
print(a != b) → True
```

3. Assignment Operators

हे Variable ला Value Assign किंवा Update करण्यासाठी वापरले जातात.

Operator	Example	Meaning
=	x = 5	Value Assign करतो
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2

4. Logical Operators

हे दोन किंवा अधिक Conditions एकत्र Check करण्यासाठी वापरतात.

Operator	Meaning
and	दोन्ही Conditions True असल्या पाहिजेत
or	एक Condition जरी True असेल तरी Result True
not	Result उलटा करतो (True → False, False → True)

```
a = 10  
  
print(a > 5 and a < 20) → True  
  
print(a > 15 or a < 20) → True  
  
print(not(a > 5)) → False
```

5. Membership Operators

हे एखादी Value Collection (List, Tuple, String इ.) मध्ये आहे का ते Check करतात.

Operator	Meaning
in	Value आहे
not in	Value नाही

```
fruits = ["Apple", "Mango", "Banana"]  
  
print("Mango" in fruits) → True  
  
print("Orange" not in fruits) → True
```

6. Identity Operators

हे दोन Variables Same Object ला Point करतात का ते Check करतात.

Operator	Meaning
is	Same Object
is not	Different Object

```
a = [1, 2]
```

```
b = a
```

```
print(a is b) → True
```

```
print(a is not b) → False
```

Interview/Exam Point (एकदम लक्षात ठेव)

- Type Casting = एका Data Type मधून दुसऱ्या Data Type मध्ये Conversion.
- Arithmetic Operators = Mathematical Calculations.
- Comparison Operators = Result नेहमी True किंवा False.
- Assignment Operators = Variable ची Value Update करतात.
- Logical Operators = Multiple Conditions Check करतात.
- Membership Operators = Collection मध्ये Value आहे का ते Check करतात.
- Identity Operators = दोन Variables एकाच Object ला Refer करतात का ते Check करतात.

Chapter 3 — Strings

Python मध्ये single ' ' किंवा double " " quotes मध्ये लिहिलेला कोणताही text म्हणजे String. String म्हणजे characters चा एक sequence.

```
name = "Rohit"
city = 'Solapur'
```

String print करायला (+ म्हणजे concatenation – strings जोडणं):

```
print("Hello, " + name) → Hello, Rohit
```

Quote च्या आत quote वापरायचा असेल तर:

```
x = 'He said, "I want to eat an apple".'
print(x) → He said, "I want to eat an apple".
```

⚠ Strings **immutable** आहेत – एकदा बनवलेला string directly बदलता येत नाही.

String Characters Access

String म्हणजे characters चा sequence . प्रत्येक character ला index number असतो जो **0 पासून** सुरू होतो.

```
name = "Rohit"

print(name[0]) → R    (पहिला character)
print(name[1]) → o    (दुसरा character)
print(name[2]) → h    (तिसरा character)
print(name[3]) → i
print(name[4]) → t
```

Negative Indexing – मागून access करायला negative index वापरतात:

```
name = "ROHIT"
print(name[-1])    → T    (शेवटचा)
print(name[-2])    → I    (शेवटून दुसरा)
print(name[-4:-2]) → OH   (index -4 ते -2 पर्यंत)
print(name[-5:-2]) → ROH  (index -5 ते -2 पर्यंत)
```

Looping Through String

for loop वापरून string मधले सगळे characters एक एक करून print करता येतात.

```
name = "Rohit"

for character in name:
    print(character) → R
                    o
                    h
                    i
                    t
```

String Slicing

String मधला specific भाग काढायला slicing वापरतात. [start:end] असं लिहितात.

```
die = "ApplePie"
print(die[0:5]) → Apple   (index 0,1,2,3,4)
print(die[5:8]) → Pie     (index 5,6,7)
print(die[0:3]) → App
```

💡 end index तो character **include** होत नाही – [0:5] म्हणजे 0,1,2,3,4 – 5 नाही!

len() Function

String मध्ये किती characters आहेत ते जाणून घ्यायला len() function वापरतात.

```
fruit = "mango"
len1 = len(fruit)
print("mango is a", len1, "letter word.")
→ mango is a 5 letter word.

name = "Rohit"
print(len(name)) → 5
```

String Methods

Python मध्ये strings बदलण्यासाठी बरेच built-in methods आहेत. string.method() असं लिहितात.

1) **upper()** – सगळे characters uppercase मध्ये करतो.

```
text2 = "ABcdEfg"
print(text2.upper()) → ABCDEFG
```

2) **lower()** – सगळे characters lowercase मध्ये करतो.

```
text2 = "ABcdEfg"
print(text2.lower()) → abcdefg
```

3) **strip()** – String च्या आधी आणि नंतरचे extra white spaces काढतो.

```
text4 = "  Silver Spoon  "
print(text4.strip()) → Silver Spoon
```

💡 Useful असतो जेव्हा user input मध्ये चुकून spaces येतात.

4) **rstrip()** – String च्या शेवटून specific character काढतो.

```
str3 = "Hello!!!"
print(str3.rstrip("!")) → Hello
```

r म्हणजे right – शेवटून काढतो.

5) **replace()** – String मधला word किंवा character दुसऱ्याने replace करतो.

```
text = "Silver Spoon"
print(text.replace("Spoon", "Fork")) → Silver Fork
```

6) **split()** – String ला specific जागी split करतो आणि list बनवतो.

```
text2 = "Silver Spoon"
print(text2.split(" ")) → ['Silver', 'Spoon']
```

इथे space " " वर split केलं – list बनली.

7) **capitalize()** – फक्त पहिला character uppercase करतो, बाकी lowercase.

```
text1 = "hello"
capstr1 = text1.capitalize()
print(capstr1) → Hello
```

⚠ पहिला character uppercase असला तरी capitalize() बाकीचे characters lowercase करते.

8) **center()** – String ला given width च्या center मध्ये align करतो.

```
text = "Welcome to the Console!!!"  
print(text.center(50))  
→ (50 characters च्या width मध्ये center होऊन print होतो)
```

9) **count()** – string मध्ये एखादी value किती वेळा आली आहे ते सांगतो.

```
text = "12232456 2"  
counttext = text.count("2")  
print(counttext) → 4
```

10) **endswith()** – string एखाद्या specific value ने end होतो का ते check करतो. True किंवा False येतो.

```
text = "welcome to the Console!!!"  
print(text.endswith("!!!")) → True  
print(text.endswith("hi")) → False
```

11) **find()** – string मध्ये एखादी value पहिल्यांदा कुठे आहे ते index देतो. Value नसेल तर -1 देतो. Start आणि End index देऊन range मध्ये पण check करता येतो.

```
text = "He's name is Dao. He is an honest man."  
print(text.find("is")) → 10  
print(text.find("Rohit")) → -1
```

💡 find() – value नसेल तर -1 देतो. index() – value नसेल तर ERROR येतो. हा फरक लक्षात ठेव!

12) index() – find() सारखाच, पण value string मध्ये नसेल तर error येतो.

```
text = "He's name is Dao."
print(text.find("Rohit")) → -1      # error नाही
print(text.index("Rohit")) → ERROR! # exception येतो
```

13) isalnum() – string मध्ये फक्त A-Z, a-z, 0-9 आहेत का ते check करतो. space किंवा ! जसे characters असतील तर False येतो.

```
text = "welcometotheConsole"
print(text.isalnum()) → True

text2 = "hello world" # space आहे म्हणून False
print(text2.isalnum()) → False
```

14) isalpha() – string मध्ये फक्त A-Z, a-z letters आहेत का ते check करतो. Numbers किंवा इतर characters असतील तर False.

```
text = "12345678"
print(text.isalpha()) → False

text2 = "HelloWorld"
print(text2.isalpha()) → True
```

15) islower() – string मधले सगळे characters lowercase आहेत का ते check करतो.

```
text = "hello world"
print(text.islower()) → True

text2 = "Hello World"
print(text2.islower()) → False
```

16) isupper() – string मधले सगळे characters uppercase आहेत का ते check करतो.

```
text1 = "ROHIT RAJESH SURVASE"  
print(text1.isupper()) → True
```

17) isspace() – string मध्ये फक्त white spaces आहेत का ते check करतो.

```
text3 = " "  
print(text3.isspace()) → True # फक्त space आहे म्हणून True
```

18) istitle() – प्रत्येक word चा पहिला letter uppercase आहे का ते check करतो. सगळे words चा पहिला letter capital असेल तरच True येतो.

```
text2 = "World Health Organisation"  
print(text2.istitle()) → True  
text3 = "world health organisation"  
print(text3.istitle()) → False
```

19) startswith() – string एखाद्या specific value ने start होतो का ते check करतो.

```
text1 = "Python is an Interpreted language"  
print(text1.startswith("Python")) → True  
print(text1.startswith("Java")) → False
```

20) swapcase() – uppercase characters lowercase करतो आणि lowercase characters uppercase करतो – swap होतात!

```
text2 = "Python is an interpreted language"
print(text2.swapcase())
→ pYTHON IS AN INTERPRETED LANGUAGE
```

String Formatting

String मध्ये variables किंवा values घालण्यासाठी Python मध्ये formatting वापरतात.

i. format() method

```
txt = "For only {price:.2f} dollars!"
print(txt.format(price=49))
→ For only 49.00 dollars!

letter = "Hey my name is {} and I am from {}"
country = "India"
name = "Rohit"
print(letter.format(name, country))
→ Hey my name is Rohit and I am from India
```

ii. f-strings

f-strings म्हणजे string च्या आधी f लिहितात आणि variables { } मध्ये directly लिहितात. हे format() पेक्षा simple आणि readable आहे.

Example 1:

```
price = 14.335
```

```
print(f"For only {price:.2f} dollars")
```

→ For only 14.34 dollars

Example 2:

```
country = "India"
```

```
name = "ROHIT"
```

```
print(f"Hey my name is {name} and I am from {country}")
```

→ Hey my name is ROHIT and I am from India

💡 `:.2f` म्हणजे 2 decimal places पर्यंत round करा!

Docstrings in Python

Python docstrings म्हणजे string literals जे function, method, class किंवा module च्या definition नंतर लगेच लिहितात. हे function काय करतो ते explain करतात.

```
def square(n):
```

```
    """ Takes in a number n, returns the square of n """
```

```
    print(n**2)
```

```
    return n**2
```

```
a = int(input("Enter your number: "))
```

```
square(a)
```

```
print(square.__doc__)
```

→ Takes in a number n, returns the square of n

💡 `__doc__` वापरून function चा docstring print करता येतो.

Comments vs Docstrings

Comments

Comment म्हणजे **code मधल्या notes**. Python हे run करत नाही, ignore करतो.

symbol वापरतात single line साठी:

```
# हा comment आहे, Python ignore करेल  
print("Hello") # हे मात्र run होईल
```

Multiple lines साठी ''' ''' वापरतात:

```
'''  
हे multi-line comment आहे  
अनेक ओळी लिहिता येतात  
Python या सगळ्या ओळी ignore करेल  
'''
```

Comments	Docstrings
Comments → # वापरतात	Docstrings → """ """ वापरतात
Comments → Python ignore करतो	Docstrings → Python store करतो
Comments → Code explain करायला	Docstrings → Function/class explain करायला

Chapter 4 — Lists

- Lists म्हणजे ordered collection of data items.
- एकाच variable मध्ये multiple items store करता येतात.
- Items comma ने separate असतात आणि square brackets [] मध्ये असतात.
- Lists changeable असतात – बनवल्यानंतर बदलता येतात.

```
l = [3, 4, 5]
print(l[0]) → 3
```

Negative Indexing

Positive indexing सारखंच असतं पण list च्या शेवटून access करतो. Last item → [-1], Second last → [-2], Third last → [-3]

```
l = ["Rohit", "Rushikesh", "Sakshi"]

print(l[-1]) → Sakshi
print(l[-2]) → Rushikesh
print(l[-3]) → Rohit
```

💡 Positive index पुढून मोजतो, Negative index शेवटून मोजतो!

Check Item Present in List

in keyword वापरून list मध्ये एखादा item present आहे का ते check करता येतो.

```
l = ["Rohit", "Rushikesh", "Sakshi"]

if "Rohit" in l:
    print("Rohit is Present")
else:
    print("Rohit is absent.")
→ Rohit is Present
```

Range of Index (Slicing)

List मधला specific range of items print करायला slicing वापरतात.

Syntax:

```
listName[start : end : JumpIndex]
```

💡 JumpIndex optional आहे – नाही दिला तरी चालतं.

Example:

```
animals = ["cat","dog","bat","mouse","pig",  
           "horse","donkey","goat","cow"]  
  
print(animals[3:7])    → ['mouse', 'pig', 'horse', 'donkey']  
print(animals[-7:-2]) → ['bat' , 'mouse', 'pig', 'horse', 'donkey']  
print(animals[1:8:2])  → ['dog', 'mouse', 'horse', 'goat']
```

💡 animals[1:8:2] म्हणजे index 1 पासून सुरू, index 8 पर्यंत, प्रत्येक 2 skip करत जा!

List Comprehension

नवीन list तयार करण्यासाठी एकाच line मध्ये loop लिहिण्याचा shortcut method.

Lists, tuples, dictionaries, sets, strings यांच्यावर वापरता येतो.

Example 1 – Simple:

```
numbers = [i for i in range(10)]  
print(numbers)  
→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example 2 – With Condition:

```
list1 = [i for i in range(10) if i % 2 == 0]  
print(list1)  
→ [0, 2, 4, 6, 8]
```

💡 range(10) मधून फक्त even numbers घे – एकाच line मध्ये! Normal loop च्या ऐवजी एक shortcut!

List Methods

1) sort()

sort() method list ला ascending order मध्ये sort करतो. Original list update होतो. reverse=True दिलं तर descending order मध्ये sort होतो.

```
l = [11, 45, 1, 2, 4, 6]
print(l)           → [11, 45, 1, 2, 4, 6]

l.append(7)
print(l)           → [11, 45, 1, 2, 4, 6, 7]

l.sort(reverse=True)
print(l)           → [45, 11, 7, 6, 4, 2, 1]
```

2) reverse()

reverse() method list च्या elements चा order उलट करतो.

```
num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
num.reverse()
print(num)         → [0, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

3) index()

index() method list मध्ये दिलेल्या item चा पहिल्यांदा येणारा index देतो.

```
num = [4, 2, 5, 3, 6, 1, 2, 1, 3, 2, 8, 9, 7]
print(num.index(3)) → 3
# म्हणजे 3 पहिल्यांदा index 3 वर आहे!
```

4) count()

count() method list मध्ये एखादी value किती वेळा आली आहे ते सांगतो.

```
num = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(num.count(2)) → 1
```

5) copy()

copy() method list ची एक copy बनवतो. Original list modify न करता नवीन copy वर operations करता येतात.

```
num = [0, 1, 2, 3, 4, 5]
newlist = num.copy()
print(num) → [0, 1, 2, 3, 4, 5]
print(newlist) → [0, 1, 2, 3, 4, 5]
```

💡 newlist मध्ये बदल केला तरी num बदलत नाही – हेच copy() चा फायदा!

6) append()

append() method existing list च्या शेवटी नवीन item add करतो.

```
colour = ["violet", "indigo", "blue"]
colour.append("green")
print(colour) → ['violet', 'indigo', 'blue', 'green']
```

7) insert()

insert() method दिलेल्या specific index वर item insert करतो. User ला index आणि item दोन्ही specify करावे लागतात.

```
num = [1, 2, 3, 4]
num.insert(1, 0)
print(num) → [1, 0, 2, 3, 4]
# म्हणजे index 1 वर 0 insert केला – बाकी elements पुढे सरकले!
```

8) extend()

extend() method एक पूर्ण list किंवा दुसरा कोणताही collection (set, tuple, dictionary) existing list मध्ये add करतो.

```
lst1 = [2, 4, 6, 8, 10]
lst2 = [1, 3, 5, 7, 9]
lst2.extend(lst1)
print(lst2) → [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
```

9) pop()

pop() method दिलेल्या index वरचा item list मधून काढतो.

```
num = [2, 4, 6, 8, 10]
num.pop(0)
print(num) → [4, 6, 8, 10]
# म्हणजे index 0 वरचा item (2) काढला गेला!
```

Concatenating Two Lists

दोन lists एकत्र जोडण्यासाठी + operator वापरतात.

```
lst1 = [2, 4, 6, 8, 10]
lst2 = [1, 3, 5, 7, 9]
lst3 = lst1 + lst2
print(lst3) → [2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
```

💡 extend() → original list मध्ये directly add होतो. | + → नवीन list बनतो, originals बदलत नाहीत.

Chapter 5 — Tuples

Python Tuple

Tuple म्हणजे ordered collection जो parentheses () मध्ये असतो. Tuples immutable असतात – एकदा बनवले की बदलता येत नाहीत.

```
tuple1 = (1, 2, 2, 3, 5, 6)
print(tuple1) → (1, 2, 2, 3, 5, 6)
```

Check Item in Tuple

in keyword वापरून tuple मध्ये एखादा item present आहे का ते check करता येतो.

```
l = (1, 2, 3, 4, 5)
if 2 in l:
    print("2 is present")
else:
    print("2 is absent")
→ 2 is present
```

Range of Index (Tuple Slicing)

List slicing सारखंच tuple मध्ये पण slicing करता येतो. JumpIndex optional आहे.

```
Syntax: Tuple[start : end : JumpIndex]

l = (2, 4, 6, 8, 10)
print(l[3:5]) → (8, 10)
print(l[-3:-1]) → (6, 8)
```

4. Manipulating Tuples

Tuples immutable आहेत म्हणून directly बदलता येत नाहीत. Changes करायचे असतील तर:

- Tuple ला list मध्ये convert कर
- List वर operations कर
- परत tuple मध्ये convert कर

```
countries = ("Spain", "Italy", "India", "England", "Germany")

temp = list(countries)    # tuple → list
temp.append("Russia")     # Russia add केला
temp.pop(3)               # index 3 (England) काढला
temp[2] = "Finland"       # India → Finland replace केलं
countries = tuple(temp)   # list → tuple परत

print(countries) → ('Spain', 'Italy', 'Finland', 'Germany', 'Russia')
```

Tuple Methods

1) count()

Tuple मध्ये दिलेला element किती वेळा आला आहे ते सांगतो.

```
tuple1 = (1, 2, 2, 3, 5, 6, 2)
print(tuple1.count(2)) → 3
```

2) index()

Tuple मध्ये दिलेल्या element चा पहिला occurrence कुठे आहे ते index देतो.

```
Syntax: tuple.index(element, start, end)
```

⚠ Element tuple मध्ये नसेल तर ValueError येतो

```
Tuple = (0, 1, 2, 3, 3, 3, 1, 3, 2)
res = Tuple.index(3)
print("First Occurrence of 3 is", res)
→ First Occurrence of 3 is 3
```

Example 2 – Specific range मध्ये search:

```
Tuple2 = (0, 1, 2, 3, 2, 3, 1, 3, 2, 3)
res = Tuple2.index(3, 4, 8)
print(res) → 5
```

म्हणजे index 4 ते 8 मध्ये 3 शोधला – index 5 वर मिळाला!

💡 थोडक्यात:

List [] → बदलता येते ✓

Tuple () → बदलता येत नाही ✗

Chapter 6 — Sets

Python Sets

Set म्हणजे unordered collection of data items. एकाच variable मध्ये multiple items store करता येतात. Items comma ने separate असतात आणि curly brackets { } मध्ये असतात.

Sets च्या 3 खास गोष्टी:

- Unordered – Elements चा कोणताही fixed order नसतो. Print केल्यावर कोणत्याही order मध्ये येऊ शकतात. Indexing होत नाही.
- Unchangeable – Set बनवल्यावर existing items directly बदलता येत नाहीत. नवीन items add/remove करता येतात.
- No Duplicates – Duplicate values automatically remove होतात. एकच value दोनदा लिहिली तरी एकदाच store होते.

```
info = {"Carla", 19, False, 5.9, 19}
print(info)
→ {False, 19, 5.9, 'Carla'}
# 19 दोनदा लिहिलं होतं – output मध्ये एकदाच आलं!
```

💡 सूचना: Set मध्ये string असेल तर प्रत्येक वेळी order वेगळा येऊ शकतो – ही Python ची सुरक्षा feature आहे. Elements मात्र नेहमी हेच राहतील.

Empty Set – Important Trick!

Empty set बनवताना एक common mistake होते जी सगळ्यांना होते:

```
# ✗ चुकीचं:
name = {}
print(type(name)) → <class 'dict'>
# हे set नाही – Python ला dictionary वाटतं!
# ✔ बरोबर:
name1 = set()
print(type(name1)) → <class 'set'>
# set() function वापरून empty set बनवतात
```

💡 {} हे नेहमी empty dictionary बनवतो. Empty set साठी नेहमी set() वापर!

Set Items Access – for loop

Set unordered असल्यामुळे index वापरून access करता येत नाही. for loop वापरून एक एक item access करतात.

```
info = {True, 'Rohit', 18, 5.9}
for item in info:
    print(item)
→ Elements कोणत्याही order मध्ये print होतात
```

Joining Sets

दोन sets एकत्र करण्याचे अनेक methods आहेत. प्रत्येकाचं काम वेगळं आहे – समजून घे.

1) union() आणि update()

union() – दोन sets चे सर्व unique elements घेऊन नवीन set बनवतो. Original sets बदलत नाहीत.

```
s1 = {1, 2, 3, 4, 5}
s2 = {2, 4, 6, 8, 10}
s3 = s1.union(s2)
print(s3) → {1, 2, 3, 4, 5, 6, 8, 10}
print(s1) → {1, 2, 3, 4, 5} # original बदलला नाही
```

update() – union() सारखंच पण s1 ला directly update करतो. नवीन set बनत नाही.

```
s1 = {1, 2, 4, 5, 7, 8}
s2 = {3, 6, 9}
s1.update(s2)
print(s1) → {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

💡 union() → नवीन set बनतो, originals safe. | update() → original set च बदलतो.

2) intersection() आणि intersection_update()

intersection() – दोन्ही sets मध्ये common असलेले elements घेऊन नवीन set बनवतो.

```
name1 = {'Rohit', 'Rajesh', 'Survase'}
name2 = {'Rushikesh', 'Rajesh', 'Survase'}
name3 = name2.intersection(name1)
print(name3) → {'Rajesh', 'Survase'}
# फक्त Rajesh आणि Survase दोन्हीमध्ये common होते
```

intersection_update() – common elements original set मध्येच ठेवतो, बाकी सर्व काढतो.

```
name1.intersection_update(name2)
print(name1) → {'Rajesh', 'Survase'}
```

3) symmetric_difference() आणि symmetric_difference_update()

symmetric_difference() – दोन sets मध्ये common नसलेले elements नवीन set मध्ये देतो. सोप्या भाषेत: union मधून intersection वजा केलं!

```
n1 = {1, 2, 3, 4, 5}
n2 = {1, 3, 5, 7}
s03 = n1.symmetric_difference(n2)
print(s03) → {2, 4, 7}
# 1,3,5 दोन्हीमध्ये common – गेले
# 2,4 फक्त n1 मध्ये + 7 फक्त n2 मध्ये – राहिले
```

symmetric_difference_update() – same result पण original set update होतो.

```
n2.symmetric_difference_update(n1)
print(n2) → {2, 4, 7}
```

4) difference() आणि difference_update()

difference() – पहिल्या set मध्ये आहेत पण दुसऱ्या set मध्ये नाहीत असे elements नवीन set मध्ये देतो.

```
s1 = {1, 2, 3, 4, 5}
s2 = {2, 4, 6, 8, 10}
s3 = s1.difference(s2)
print(s3) → {1, 3, 5}
# 1,3,5 फक्त s1 मध्ये – आले
# 2,4 दोन्हीमध्ये – गेले
# ⚠ Order matters!
s1.difference(s2) ≠ s2.difference(s1)
```

difference_update() – same result पण original set update होतो.

```
s1.difference_update(s2)
print(s1) → {1, 3, 5}
```

Complete Summary – Joining Methods

Method	म्हणजे	नवीन set?	Original बदलतो?
union()	सर्व elements एकत्र	✓	✗
update()	सर्व elements एकत्र	✗	✓
intersection()	फक्त common elements	✓	✗
intersection_update()	फक्त common elements	✗	✓

Method	म्हणजे	नवीन set?	Original बदलतो?
<code>symmetric_difference()</code>	Non-common elements	✓	✗
<code>symmetric_difference_update()</code>	Non-common elements	✗	✓
<code>difference()</code>	s1 मध्ये, s2 मध्ये नाही	✓	✗
<code>difference_update()</code>	s1 मध्ये, s2 मध्ये नाही	✗	✓

Set Methods

Python मध्ये sets manipulate करण्यासाठी अनेक built-in methods आहेत.

1) `isdisjoint()`

`isdisjoint()` check करतो की दोन sets मध्ये कोणताही common element आहे का. Common असेल → False. Common नसेल → True.

```
name1 = {'Rohit', 'Samanth', 'Sushant', 'Rushikesh'}
name2 = {'Rohit', 'Pappu', 'Shambhu', 'Rushikesh'}
print(name1.isdisjoint(name2)) → False
# 'Rohit' आणि 'Rushikesh' common आहेत → False

name3 = {'Pappu', 'Shambhu'}
print(name1.isdisjoint(name3)) → True
# name1 आणि name3 मध्ये काहीही common नाही → True
```

2) issuperset()

issuperset() check करतो की दुसऱ्या set चे सर्व items original set मध्ये आहेत का. म्हणजे 'मी मोठा आहे का?' हे check करतो.

```
name1 = {'Rohit', 'Samanth', 'Sushant', 'Rushikesh'}
name2 = {'Rohit', 'Samanth'}
print(name1.issuperset(name2)) → True
# name2 चे सर्व items name1 मध्ये आहेत → True

name3 = {'Rohit', 'XYZ'}
print(name1.issuperset(name3)) → False
# 'XYZ' name1 मध्ये नाही → False
```

3) issubset()

issubset() check करतो की original set चे सर्व items दुसऱ्या set मध्ये आहेत का. म्हणजे 'मी छोटा आहे का?' हे check करतो.

```
name1 = {'Rohit', 'Samanth'}
name2 = {'Rohit', 'Samanth', 'Sushant', 'Rushikesh'}
print(name1.issubset(name2)) → True
# name1 चे सर्व items name2 मध्ये आहेत → True
```

💡 A.issuperset(B) → A मध्ये B चं सगळं आहे? | A.issubset(B) → B मध्ये A चं सगळं आहे?

4) add()

एकच item set मध्ये add करायला add() वापरतात. Duplicate असेल तर add होत नाही.

```
name = {'Rohit', 'Samanth', 'Sushant', 'Rushikesh'}
name.add('Pappu')
print(name) → {'Rohit', 'Samanth', 'Sushant', 'Rushikesh', 'Pappu'}

name.add('Rohit') # already आहे → duplicate add नाही होत
```

5) update()

एकाहून जास्त items add करायला update() वापरतात. दुसरा set, list, tuple, dictionary कोणताही iterable देता येतो.

```
num1 = {1, 2, 3, 4}
num2 = {5, 6}
num1.update(num2)
print(num1)    → {1, 2, 3, 4, 5, 6}

# List पण देता येतो:
num1.update([7, 8, 9])
print(num1)    → {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

6) remove() आणि discard()

दोन्ही methods item remove करतात – पण error handling मध्ये फरक आहे.

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

# remove() – item नसेल तर KeyError!
num.remove(3)
print(num)      → {1, 2, 4, 5}  ✓
num.remove(10) → KeyError: 10  ✗
```

💡 remove() → item नक्की आहे याची खात्री असेल तर. | discard() → safe way, error नाही येत.

```
num = {1, 2, 3, 4, 5}
num.discard(3)
print(num)      # {1, 2, 4, 5}
num.discard(10) # No error
```

7) del

del हा method नाही – Python keyword आहे. Set पूर्णपणे memory मधून delete होतो. नंतर access केलं तर NameError येतो.

```
Cities = {"Tokyo", "Madrid", "Berlin", "Delhi"}
del Cities
print(Cities) → NameError: name 'Cities' is not defined
```

8) clear()

clear() method set मध्ये सर्व items काढतो पण variable memory मध्ये राहतो – empty set म्हणून.

```
num = {1, 2, 3, 4, 5}
num.clear()
print(num) → set() # empty set, NameError नाही
```

💡 del → variable पूर्णपणे नष्ट होतो → NameError. | clear() → variable राहतो पण रिकामा → set()

9) Check if Item Exists

in keyword वापरून set मध्ये item आहे का ते check करता येतो.

```
num = {1, 2, 3, 4, 5, 6, 7}
if 2 in num:
    print("2 is present")
else:
    print("2 is absent")
→ 2 is present
```

Chapter 7 – Dictionaries

Python Dictionaries

Dictionary म्हणजे key:value pairs चा ordered collection. प्रत्येक key unique असतो आणि त्याला एक value असते. Curly brackets { } मध्ये लिहितात, key आणि value मध्ये : असतो.

Real life example – Dictionary म्हणजे contact list सारखं. Name (key) → Phone number (value). तू नाव सांगतोस, त्याचा number मिळतो.

```
dic = {  
    "Rohit": "Human being",  
    "spoon": "object"  
}  
print(dic["Rohit"]) → Human being
```

Dictionary च्या खास गोष्टी:

- Keys unique असतात – duplicate keys नाही चालत
- Values duplicate असू शकतात
- Keys immutable असाव्या लागतात (string, number, tuple)
- Values कोणत्याही type च्या असू शकतात
- Python 3.7+ मध्ये Dictionary ordered आहे

Creating a Dictionary

Dictionary तयार करण्यासाठी Curly Brackets {} वापरतात. प्रत्येक key : value pair comma (,) ने वेगळा लिहितात.

```
student = {  
    "name": "Rohit",  
    "age": 19,  
    "course": "Python"  
}  
print(student) → {'name': 'Rohit', 'age': 19, 'course': 'Python'}
```

Empty Dictionary

Empty Dictionary तयार करण्यासाठी दोन ways आहेत.

```
d1 = {}  
  
d2 = dict()  
  
print(d1) → {}  
print(d2) → {}
```

Accessing Dictionary

1) Single Value Access

Dictionary मधून single value access करायला दोन ways आहेत:

```
info = {'name': 'Rohit', 'age': 19, 'eligible': True}  
  
# Way 1 – Square brackets  
print(info['name'])      → Rohit  
print(info['age'])       → 19  
  
# Way 2 – get() method  
print(info.get('eligible')) → True  
print(info.get('salary'))  → None   # key नाही पण error नाही!
```

💡 info['salary'] → KeyError ❌ | info.get('salary') → None ✔️ |
info.get('salary', 0) → 0 (default value) ✔️

2) Accessing Multiple Values – values()

Dictionary मधल्या सर्व values एकत्र print करायला values() method वापरतात.

```
info = {'name': 'Rohit', 'age': 18, 'eligible': True}
print(info.values())
→ dict_values(['Rohit', 18, True])
# Loop वापरून:
for val in info.values():
    print(val)
→ Rohit → 18 → True
```

3) Accessing All Keys – keys()

Dictionary मधल्या सर्व keys print करायला keys() method वापरतात.

```
info = {'name': 'Rohit', 'age': 18, 'eligible': True}
print(info.keys())
→ dict_keys(['name', 'age', 'eligible'])
# Loop वापरून:
for key in info.keys():
    print(key)
→ name → age → eligible
```

4) Accessing Key-Value Pairs – items()

Dictionary मध्ये सर्व key-value pairs एकत्र print करायला items() method वापरतात. प्रत्येक pair tuple म्हणून येतो.

```
info = {'name': 'Rohit', 'age': 18, 'eligible': True}
print(info.items())
→ dict_items([('name', 'Rohit'), ('age', 18), ('eligible', True)])
# Loop वापरून:
for key, value in info.items():
    print(key, "→", value)
→ name → Rohit | age → 18 | eligible → True
```

Adding New Items

Dictionary मध्ये नवीन key:value pair add करण्यासाठी नवीन key assign करतात. जर key आधीपासून नसेल तर ती add होते.

```
info = {  
    "name":"Rohit",  
    "age":19  
}  
  
info["city"] = "Pune"  
  
print(info) → {'name': 'Rohit', 'age': 19, 'city': 'Pune'}
```

💡 जर key आधीपासून असेल तर तिची value update होते.

Dictionary Methods

1) update()

update() method dictionary मध्ये existing key असेल तर value update करतो. Key नसेल तर नवीन key-value pair add करतो.

```
info = {'name': 'Rohit', 'age': 18, 'eligible': True}  
info.update({'age': 20})    # 'age' already आहे → value 18→20 update  
info.update({'DOB': 2006}) # 'DOB' नवीन key → add होतो  
print(info)  
→ {'name': 'Rohit', 'age': 20, 'eligible': True, 'DOB': 2006}
```

2) copy()

copy() method dictionary ची copy तयार करतो. नवीन dictionary original dictionary पासून वेगळी असते.

```
info = {  
    "name":"Rohit",  
    "age":19  
}  
  
new_info = info.copy()  
print(new_info) → {'name': 'Rohit', 'age': 19}
```

Removing Items from Dictionary

Dictionary मधून items काढण्याचे 4 ways आहेत. प्रत्येकाचं वेगळं काम आहे.

i) clear()

clear() method dictionary मधले सर्व items काढतो – empty dictionary राहतो.
Variable नष्ट होत नाही.

```
ep1 = {122:45, 123:89, 567:69, 670:69}
ep1.clear()
print(ep1)    → {}    # empty dictionary
```

ii) pop()

pop() method specified key चा key-value pair dictionary मधून काढतो आणि removed value return करतो.

```
ep1 = {122:45, 123:89, 567:69, 670:69}
removed = ep1.pop(123)
print(ep1)    → {122:45, 567:69, 670:69}
print(removed) → 89    # removed value return झाली
```

⚠ Key नसेल तर KeyError येतो!

iii) popitem()

popitem() method dictionary मधला शेवटचा key-value pair काढतो आणि tuple म्हणून return करतो.

```
ep1 = {122:45, 123:89, 567:69, 670:69}
removed = ep1.popitem()
print(ep1)    → {122:45, 123:89, 567:69}
print(removed) → (670, 69)    # शेवटचा pair tuple म्हणून आला
```

iv) del

del keyword दोन प्रकारे वापरतात – specific item delete किंवा entire dictionary delete.

```
# Specific item delete:
ep2 = {100:100, 101:100, 102:99}
del ep2[100]
print(ep2) → {101:100, 102:99}
# Entire dictionary delete:
ep1 = {122:45, 123:89, 567:69, 670:69}
del ep1
print(ep1) → NameError: name 'ep1' is not defined
# variable नष्ट झाला
```

Complete Summary – Dictionary Remove Methods

Method	काय करतो	Result
clear()	सर्व items काढतो	Empty dict {} राहतो
pop(key)	Specific key चा pair काढतो	Removed value return होते
popitem()	शेवटचा pair काढतो	Removed pair tuple म्हणून येतो
del dict[key]	Specific key delete	Key जातो
del dict	पूर्ण dictionary delete	Variable नष्ट होतो

Membership Operators

Dictionary मध्ये specific key आहे का नाही हे check करण्यासाठी in आणि not in operators वापरतात.

```
info = {  
    "name":"Rohit",  
    "age":19  
}  
  
print("name" in info) → True  
  
print("salary" not in info) → True
```

len() Function

len() function dictionary मधील एकूण key:value pairs ची संख्या return करते.

```
info = {  
    "name":"Rohit",  
    "age":19,  
    "city":"Pune"  
}  
  
print(len(info)) → 3
```

Looping Through Dictionary

Dictionary मधील प्रत्येक key किंवा value एक-एक करून access करण्यासाठी for loop वापरतात.

```
info = {  
    "name":"Rohit",  
    "age":19,  
    "city":"Pune"  
}  
  
for key in info:  
    print(key) →      name  
                     age  
                     city
```

Nested Dictionary

एका Dictionary च्या आत दुसरी Dictionary असू शकते. याला Nested Dictionary म्हणतात.

```
students = {  
    "student1":{  
        "name":"Rohit",  
        "age":19  
    },  
}  
  
print(students["student1"]["name"]) → Rohit
```

Chapter 8 – Conditional Statements

What is Conditional Statement?

Conditional Statement म्हणजे एखादी condition check करून त्यानुसार वेगवेगळे code execute करण्याची process.

जर condition True असेल तर एक block execute होतो आणि False असेल तर दुसरा block execute होतो.

Real-life Example:

जर पाऊस पडत असेल तर Umbrella घे.

नाहीतर Umbrella घेण्याची गरज नाही.

If-Else Statement

कधी कधी programmer ला एखाद्या expression चं evaluation check करायचं असतं – True आहे की False. True असेल तर एक path, False असेल तर दुसरा path follow होतो.

Conditional statements चे पुढील प्रकार आहेत:

- if
- if-else
- if-elif-else
- nested if-elif-else

► Condition Operators

>	म्हणजे greater than
<	म्हणजे less than
>=	म्हणजे greater than or equal to
<=	म्हणजे less than or equal to
==	म्हणजे equal to
!=	म्हणजे not equal to

If-Else कसं काम करतं?

जर expression True असेल → if block मधला code run होतो. जर expression False असेल → else block मधला code run होतो.

Simple if-else:

```
a = int(input("Enter your age: "))
print("Your age is", a)

if (a >= 18):
    print("You can drive")
else:
    print("You can't drive")
```

elif Statement

elif म्हणजे 'else if' – एकाहून जास्त conditions check करायच्या असतील तर elif वापरतात.

if-elif-else:

```
num = int(input("Enter the value of num: "))

if (num < 0):
    print("The Number is Negative")
elif (num == 0):
    print("The Number is Zero")
else:
    print("The Number is Positive")
```

💡 if → elif → elif → else – जास्त conditions असतील तर elif वापरतात. else एकच असतो आणि शेवटी असतो!

Match-Case Statement

Match-case म्हणजे Python मधला switch-case. एका variable च्या value नुसार वेगळे cases handle करतो. case _ म्हणजे default case – कोणताही case match नाही झाला तर हा run होतो.

```
x = 4
match x:
    case 0:
        print("x is zero")
    case 4 if x % 2 == 0:
        print("x%2==0 and case is 4")
    case _ if x <= 10:
        print(x)
    case _:
        print(x)
        → x%2==0 and case is 4
```

💡 case _ म्हणजे default – if-else मधल्या else सारखंच काम करतो!

Nested if Statement

Nested if म्हणजे एका if statement च्या आत दुसरा if statement असणे. जेव्हा एकापेक्षा जास्त conditions step by step check करायच्या असतात तेव्हा Nested if वापरतात.

Syntax

```
if condition1:
    if condition2:
        statement
```

```
age = 20

citizen = True

if age >= 18:

    if citizen:

        print("Eligible to Vote") → Eligible to Vote
```

Explanation:

सुरुवातीला `age >= 18` check होते. Condition True असल्यामुळे दुसरा if execute होतो. citizen पण True असल्यामुळे "Eligible to Vote" print होते.

Short Hand if Statement

जर फक्त एकच statement execute करायचं असेल, तर if statement एका line मध्ये लिहू शकतो. याला Short Hand if म्हणतात.

Syntax

```
if condition: statement
```

```
age = 20

if age >= 18 :

    print("Eligible to Vote")

→ Eligible to Vote
```

Short Hand if...else Statement

जर if आणि else दोन्ही एका line मध्ये लिहायचे असतील, तर Short Hand if...else वापरतात. याला Ternary Operator देखील म्हणतात.

Syntax

```
statement_if_true if condition else statement_if_false
```

```
age = 16  
print("Eligible") if age >= 18 else print("Not Eligible")  
→ Not Eligible
```

```
num = 25  
result = "Even" if num % 2 == 0 else "Odd"  
print(result) → Odd
```

💡 Note: Short Hand if आणि if...else फक्त simple conditions साठी वापरणे चांगले. Complex conditions साठी normal if...else वापरल्यास code जास्त readable आणि easy to understand राहतो.

Chapter 9 – Loops

Loops in Python

कधी कधी programmer ला एक group of statements certain number of times execute करायचे असतात – यासाठी loops वापरतात. Loops चे पुढील प्रकार आहेत: for loop, while loop, nested loop.

for loop

for loop म्हणजे – strings, lists, tuples, sets, dictionaries यांच्यावर iterate करतो. म्हणजे एक एक element घेऊन process करतो.

```
name = 'ROHIT'
for i in name:
    print(i, end=" ") → R O H I T
```

range() Function

range() वापरून specific number of times loop run करता येतो.

```
for k in range(5):
    print(k)
→ 0 1 2 3 4
```

💡 range(5) म्हणजे 0 ते 4 – 5 include होत नाही! Counting नेहमी 0 पासून सुरू होते.

while loop

while loop म्हणजे – दिलेली condition True असेपर्यंत loop चालत राहतो.

```
count = 5
while (count > 0):
    print(count)
    count = count - 1
```

Output: 5 4 3 2 1

```
i = 0
while (i <= 3):
    print(i)
    i = i + 1
print("Done with the loop")
```

Output: 0 1 2 3 Done with the loop

⚠ while loop मध्ये counter update ($i = i + 1$) करायला विसरू नकोस – नाहीतर infinite loop येतो आणि program बंद होत नाही!

Else with While Loop

While loop सोबत else पण वापरता येतो. जेव्हा while loop ची condition False होते, तेव्हा else block run होतो.

```
i = 0
while (i < 5):
    print(i)
    i = i + 1
else:
    print("Loop finished!")
→ 0, 1, 2, 3, 4
→ Loop finished!
```

do-while Loop

Python मध्ये directly do-while loop नाही, पण while loop वापरून same behavior मिळवता येतो.

while vs do-while फरक:

- while → आधी condition check होते, मग code run होतो.
- do-while → आधी code एकदा run होतो, मग condition check होते.
- म्हणजे do-while मध्ये code कमीत कमी एकदा नक्की run होतो.

Python मध्ये do-while असं लिहितात:

```
i = 1
while True:
    print(i)
    i = i + 1
    if i > 5:
        break
→ 1, 2, 3, 4, 5
```

💡 while True म्हणजे loop नेहमी चालत राहतो – break आल्यावरच थांबतो. यामुळे code कमीत कमी एकदा run होतो.

break Statement

break statement म्हणजे – loop च्या मध्येच बाहेर पडणं. जेव्हा break येतो, तेव्हा तो loop त्याच क्षणी terminate होतो.

```
for i in range(1, 201, 2):
    print(i, end=" ")
    if (i == 49):
        break
    else:
        print("Mississippi")
print("Thank You")
```

```
for i in range(12):
    if (i == 10):
        break
    print("5 x", i+1, "=", 5*(i+1))
```

continue Statement

continue statement म्हणजे – current iteration skip करणं आणि पुढच्या iteration ला जाणं. Loop बंद होत नाही, फक्त ती एक iteration skip होते.

```
for i in [2, 3, 4, 6, 8, 0]:
```

```
    if (i % 2 != 0):
```

```
        continue
```

```
    print(i)
```

→ Odd numbers skip, फक्त even print होतात

💡 break → loop पूर्णपणे बंद होतो. | continue → फक्त ती iteration skip होते, loop चालत राहतो.

Nested Loop

Nested Loop म्हणजे एका loop च्या आत दुसरा loop असणे. जेव्हा rows आणि columns वर repeated operation करायचे असतात तेव्हा Nested Loop वापरतात.

Syntax

```
for outer in range():
```

```
    for inner in range():
```

```
        statement
```

```
for i in range(3):
```

```
    for j in range(2):
```

```
        print(i, j)
```

Output → 0 0

0 1

1 0

1 1

2 0

2 1

💡 Outer loop प्रत्येक वेळी run होतो आणि प्रत्येक outer iteration साठी inner loop पूर्ण execute होतो.

pass Statement

pass statement म्हणजे काहीही operation न करता program पुढे continue करणे. जेव्हा syntax पूर्ण ठेवायचा असतो पण आता code लिहायचा नसतो तेव्हा pass वापरतात.

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

```
    if i == 3:
```

```
        pass
```

```
    print(i) →      0
```

```
                  1
```

```
                  2
```

```
                  3
```

```
                  4
```

💡 pass काहीच करत नाही. तो फक्त placeholder म्हणून वापरला जातो.

else in Loop

आपण if statement सोबत else वापरतो हे माहीत आहे. पण Python मध्ये एक unique feature आहे – for loop आणि while loop सोबत पण else वापरता येतो. हे इतर programming languages मध्ये नसतं – Python specific आहे!

else in loop म्हणजे काय?

Loop successfully complete झाला – म्हणजे कोणताही break statement आला नाही – तर else block run होतो. जर loop मध्ये break आला तर else block skip होतो.

Syntax:

```
for counter in sequence:
    # for loop चा code
else:
    # loop break शिवाय संपल्यावर run होतो
```

```
for i in range(5):
    print(i)
else:
    print('Sorry no i')
→ 0  1  2  3  4
→ Sorry no i
# Loop successfully संपला → else run झाला!
```

```
for i in range(5):
    if i == 3:
        break      # i == 3 आल्यावर loop break
    print(i)
else:
    print('loop संपला')  # break आला → else skip!
→ 0  1  2
# Break आला म्हणून else run झाला नाही!
```

Real-world use – Search करणे:

```
numbers = [1, 3, 5, 7, 9]
search = 4

for num in numbers:
    if num == search:
        print(f"{search} found!")
        break
else:
    print(f"{search} not found in list!")

→ 4 not found in list!
```

💡 else in loop – break आला नाही याची confirmation देतो. Searching, validation यासाठी खूप useful!

Chapter 10 – Functions, Recursion & Variable Scope

What is Function

Function म्हणजे एक block of code जो एखादं specific task perform करतो. जेव्हा त्याला call करतो तेव्हाच तो execute होतो. मोठा program छोट्या छोट्या parts मध्ये divide करण्यासाठी functions वापरतात, त्यामुळे code वाचायला आणि maintain करायला सोपा जातो.

Real-life Example – जसं mixer ला वेगवेगळी buttons असतात (grind, juice, blend) आणि प्रत्येक button दाबल्यावर ठराविक काम होतं, तसंच function ला call केल्यावर तो त्याचं ठराविक task करतो.

Syntax

```
def function_name(parameters):
```

```
    # code
```

```
    return value
```

```
def greet():
```

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

```
greet() → Hello, Welcome to Python!
```

💡 Tip: Function नेहमी def keyword ने सुरू होतो आणि code मध्ये reusability आणतो

Built-in Functions

Built-in Functions म्हणजे Python मध्ये आधीच तयार असलेली functions. ही function आपल्याला स्वतः लिहावी लागत नाहीत, ती थेट वापरता येतात. उदा. `print()`, `input()`, `len()`, `type()` वगैरे.

```
print(len("Python")) → 6
```

```
print(type(10)) → <class 'int'>
```

💡 Tip: Built-in functions वापरण्यासाठी वेगळं काहीही तयार करावं लागत नाही, ते Python मध्ये आधीच उपलब्ध असतात.

input() Function

`input()` function वापरून user कडून keyboard द्वारे data घेता येतो. हे function जे काही टाइप केलं जातं ते नेहमी String स्वरूपात return करतं, मग user ने number टाइप केला तरी.

Syntax

```
variable = input("message")
```

```
name = input("Enter your name: ")
```

```
print("Hello,", name) → Hello, Rohit
```

```
# (user ने "Rohit" टाइप केलं तर)
```

💡 Tip: `input()` मधून येणारी value नेहमी str असते. Number हवा असेल तर `int()` किंवा `float()` वापरून convert करावं लागतं.

int() Function

int() function एखाद्या value ला integer मध्ये convert करतो. String मधला number किंवा decimal number, int मध्ये बदलण्यासाठी हे function वापरतात.

Syntax

```
int(value)
```

```
age = int(input("Enter your age: "))  
  
print(age + 1) → 19  
  
# (user ने 18 टाइप केलं तर)
```

💡 Tip: int() ला decimal point असलेली text (उदा. "18.5") दिली तर error येतो.

float() Function

float() function value ला decimal number मध्ये convert करतो.

Syntax

```
float(value)
```

```
price = float(input("Enter price: "))  
  
print(price) → 99.0  
  
# (user ने 99 टाइप केलं तर)
```

💡 Tip: float() ला valid number दिला नाही तर ValueError येतो.

print() Function

print() function screen वर output दाखवण्यासाठी वापरतात. यामध्ये multiple values comma ने वेगळे देता येतात, ते आपोआप space ने वेगळे print होतात.

```
print("Name:", "Rohit", "Age:", 19) → Name: Rohit Age: 19
```

print() मध्ये sep आणि end नावाचे parameters पण असतात, ज्यामुळे output चा format बदलता येतो. sep values मधलं separator ठरवतो आणि end शेवटी काय येईल ते ठरवतो.

```
print("Python", "is", "fun", sep="-") → Python-is-fun  
  
print("Hello", end=" ")  
  
print("World") → Hello World
```

💡 **Tip:** end चं default value "\n" असतं, म्हणूनच प्रत्येक print() नंतर आपोआप नवीन ओळ येते.

User-defined Functions

User-defined Function म्हणजे programmer स्वतः तयार करतो ते function. Built-in functions च्या उलट, हे function आपण आपल्या गरजेनुसार स्वतः लिहितो.

Syntax

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

```
def wish():  
    print("Good Morning!")  
  
wish() → Good Morning!
```

💡 **Tip:** Function चं नाव नेहमी meaningful ठेवावं, म्हणजे code वाचायला सोपा जातो.

Function Calling

Function तयार केल्यानंतर तो प्रत्यक्ष run करण्यासाठी call करावा लागतो. Function name नंतर parentheses () लिहून तो call करतात. Function फक्त define केल्याने काहीच होत नाही – call केल्यावरच तो execute होतो.

Syntax

```
function_name()
```

```
def add():  
    print(5 + 3)  
  
add() # function calling → 8
```

💡 **Tip:** एकच function हवं तेवढ्या वेळा call करता येतो.

Function Arguments

Function ला call करताना आपण त्याला काही values पाठवतो, त्यांना Arguments म्हणतात. Function define करताना parentheses मध्ये

लिहिलेल्या नावांना Parameters म्हणतात. Python मध्ये Arguments चे मुख्यतः चार प्रकार आहेत – Required, Default, Keyword आणि Variable-length Arguments.

```
def student(name, age):  
    print(name, "is", age, "years old")  
  
student("Rohit", 19) → Rohit is 19 years old
```

💡 **Tip:** Parameter म्हणजे function define करतानाच नाव, आणि Argument म्हणजे function call करताना दिलेली actual value.

Required Arguments

Required Arguments म्हणजे function call करताना द्यावेच लागणारे arguments. जितके parameters असतील तितकेच arguments call करताना द्यावे लागतात, नाहीतर error येतो.

Real-life Example – जसं ATM मधून पैसे काढताना PIN टाकणं required असतं, नाहीतर transaction होत नाही, तसंच Required Argument दिल्याशिवाय function काम करत नाही.

```
def add(a, b):  
    print(a + b)  
  
add(5, 3) → 8
```

💡 **Tip:** Arguments ची संख्या आणि order दोन्ही parameters सोबत match व्हावी लागतात.

Default Arguments

Default Argument म्हणजे function define करतानाच parameter ला एक default value दिलेली असते. Call करताना त्यासाठी value दिली नाही तर हीच default value वापरली जाते.

Real-life Example – जसं Swiggy वर order करताना "Extra Spicy" सांगितलं नाही तर normal spice level येतो, तसंच value न दिल्यास function ची default value वापरली जाते.

Syntax

```
def function_name(parameter=default_value):  
    # code
```

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

💡 **Tip:** Default Argument नेहमी parameter list मध्ये शेवटी ठेवावा.

Keyword Arguments

Keyword Arguments मध्ये आपण argument देताना parameter चं नाव लिहून value देतो. यामुळे arguments कोणत्याही order मध्ये दिले तरी चालतात.

Real-life Example – जसं एखादा Form भरताना Name, Age, City कोणत्याही order मध्ये भरता येतं कारण प्रत्येक field ला label असतो, तसंच Keyword Argument मध्ये नाव दिल्यामुळे order मॅटर करत नाही.

Syntax

```
function_name(parameter1=value1, parameter2=value2)
```

```
def student(name, age):  
    print(name, "is", age, "years old")  
  
student(age=19, name="Rohit")  
→ Rohit is 19 years old
```

💡 **Tip:** Keyword Arguments वापरल्यास Python ला नावावरूनच कळतं कोणती value कुठे जायची, म्हणून order बदलली तरी काही फरक पडत नाही.

Variable-length Arguments (*args)

कधी कधी function ला किती arguments येणार आहेत हे आधीच माहीत नसतं. अशा वेळी *args वापरतात. यामुळे function एकाहून जास्त arguments एका tuple च्या स्वरूपात स्वीकारतो.

Real-life Example – जसं Grocery bag मध्ये कितीही वस्तू टाकता येतात, तसंच *args मध्ये कितीही arguments पाठवता येतात.

Syntax

```
def function_name(*args):  
    # code
```

```
def add_all(*numbers):  
    print(sum(numbers))
```

```
add_all(1, 2, 3) → 6
```

```
add_all(10, 20) → 30
```

💡 **Tip:** *args मधलं नाव "args" काहीही ठेवता येतं, पण * हे symbol महत्वाचं आहे.

Keyword Variable-length Arguments (**kwargs)

**kwargs वापरून function मध्ये कितीही Keyword Arguments एका dictionary च्या स्वरूपात पाठवता येतात. यामध्ये key आणि value दोन्ही मिळतात.

Syntax

```
def function_name(**kwargs):  
    # code
```

```
def info(**details):  
    for key, value in details.items():  
        print(key, "-", value)  
  
info(name="Rohit", age=19)  
  
→ name - Rohit  
    age - 19
```

💡 **Tip:** `**kwargs` मुळे function ला fix arguments न देता कितीही named values पाठवता येतात.

Return Statement

Return statement वापरून function मधून एखादी value बाहेर पाठवली जाते. Return झाल्यावर function चा execution तिथेच थांबतो, त्यानंतरचा code कधीच run होत नाही.

Syntax

```
return value
```

```
def square(n):  
    return n * n  
  
result = square(5)  
print(result) → 25
```

💡 **Tip:** Return statement नंतर लिहिलेला code कधीच execute होत नाही.

Difference between print() and return()

`print()` फक्त value screen वर दाखवतो, तो value कुठेही store करत नाही. `return` मात्र value function च्या बाहेर पाठवतो, जी पुढे variable मध्ये store करून वापरता येते.

```
def add_print(a, b):  
    print(a + b)
```

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

add_print(2, 3) → 5

result = add_return(2, 3)

print(result) → 5
```

print()	return
Value फक्त screen वर दाखवतो	Value function च्या बाहेर पाठवतो
Value कुठेही store होत नाही	Value variable मध्ये store करता येते
पुढे वापरता येत नाही	पुढे calculation साठी वापरता येते

💡 **Tip:** फक्त बघण्यासाठी print() वापर, पुढे value वापरायची असेल तर return वापर.

Enumerate Function

enumerate() function loop चालवताना प्रत्येक item सोबत त्याचा index पण देतो. यामुळे item आणि त्याचं position दोन्ही एकत्र मिळतं.

Syntax

```
enumerate(sequence)
```

```
fruits = ["Apple", "Mango", "Banana"]
```

```
for index, fruit in enumerate(fruits):
```

```
    print(index, fruit) →      0 Apple  
                                1 Mango  
                                2 Banana
```

💡 **Tip:** `enumerate()` मध्ये `start value` पण देता येते, उदा. `enumerate(fruits, start=1)`.

Lambda Function

Lambda Function म्हणजे नाव नसलेलं (anonymous) छोटं function. हे एका ओळीत लिहितात आणि साधी, छोटी कामं करण्यासाठी वापरतात.

Syntax

```
lambda arguments: expression
```

```
square = lambda n: n * n
```

```
print(square(5)) → 25
```

💡 **Tip:** Lambda function मध्ये `return keyword` लिहावं लागत नाही, `expression` चा `result` आपोआप `return` होतो.

Lambda Limitations

Lambda function मध्ये फक्त एकच expression लिहिता येतो. यामध्ये एका ओळीत if-else सारखी छोटी condition वापरता येते, पण multiple statements किंवा loops टाकता येत नाहीत. मोठं आणि complex logic असेल तर normal function वापरणेच योग्य असतं.

```
check = lambda n: "Even" if n % 2 == 0 else "Odd"

print(check(4)) → Even
```

💡 **Tip:** फक्त छोट्या, one-line कामांसाठीच Lambda वापरावा.

Local Variables

Local Variable म्हणजे function च्या आत define केलेला variable. हा variable फक्त त्या function च्या आतच वापरता येतो, बाहेर त्याचा access नसतो.

Real-life Example – जसं तुमच्या स्वतःच्या खोलीतली वस्तू फक्त तुमच्याच खोलीत वापरता येते, तसंच Local Variable फक्त त्याच function च्या आत वापरता येतो.

```
def show():

    message = "Hello from function"

    print(message)

show()

print(message)

→ Hello from function

NameError: name 'message' is not defined
```

💡 **Tip:** Local variable फक्त त्याच function पुरता मर्यादित असतो – function संपला की तो नष्ट होतो.

Global Variables

Global Variable म्हणजे function च्या बाहेर define केलेला variable. हा variable संपूर्ण program मध्ये कुठूनही access करता येतो – function च्या आतून सुद्धा.

Real-life Example – जसं घरातील एखादी वस्तू संपूर्ण घरात कुठूनही वापरता येते, तसंच Global Variable संपूर्ण program मध्ये access करता येतो.

```
message = "Hello Global"

def show():
    print(message)

show() → Hello Global
```

💡 **Tip:** Global variable function च्या आतून फक्त read करता येतो, त्याची value बदलायची असेल तर global keyword वापरावं लागतं.

global Keyword

Function च्या आतून Global Variable ची value बदलायची असेल तर global keyword वापरावं लागतं. हे keyword न वापरता value बदलण्याचा प्रयत्न केला तर Python एक नवीन Local Variable तयार करतो, आणि original global variable बदलत नाही.

Syntax

```
global variable_name
```

```
count = 0

def increase():
    global count
    count = count + 1

increase()

print(count) → 1
```

💡 **Tip:** global keyword वापरल्याशिवाय function च्या आत original global variable बदलता येत नाही.

Recursion

Recursion म्हणजे function स्वतःलाच call करतो. मोठी problem छोट्या छोट्या भागांमध्ये तोडून सोडवण्यासाठी Recursion वापरतात. प्रत्येक recursive function ला एक Base Condition असावी लागते, नाहीतर तो infinite वेळा call होत राहतो.

Real-life Example –

आरशासमोर आरसा ठेवला की एकाच image ची प्रतिमा पुन्हा पुन्हा दिसत राहते, तसंच Recursion मध्ये function स्वतःला पुन्हा पुन्हा call करतो.

Syntax

```
def function_name(parameters):  
    if base_condition:  
        return value  
  
    else:  
        return function_name(smaller_parameters)
```

```
def factorial(n):  
    if n <= 1:  
        return 1  
  
    else:  
        return n * factorial(n - 1)  
  
print(factorial(5)) → 120
```

💡 **Tip:** Base Condition नसेल तर RecursionError येतो – function कधीच थांबत नाही.

Chapter 11 – Functional Programming Tools

Functional Programming म्हणजे functions चा वापर करून data वर operations perform करण्याची programming style. यात code छोटा, clean आणि reusable बनतो. Python मध्ये map(), filter() आणि reduce() हे Functional Programming चे मुख्य tools आहेत.

Real-life Example:

समजा एका class मधल्या सर्व विद्यार्थ्यांच्या marks मध्ये 5 marks add करायचे आहेत. प्रत्येक mark manually बदलण्यापेक्षा एक function तयार करून तो सर्व marks वर apply करणे सोपं आणि जलद असतं. Functional Programming मध्ये हेच काम map() सारख्या tools ने केलं जातं.

💡 **Note** : Functional Programming मध्ये original data बदलण्याऐवजी नवीन result तयार केला जातो. त्यामुळे code जास्त readable आणि maintain करायला सोपा होतो.

map() – प्रत्येक Element वर Operation

map() म्हणजे एक factory worker सारखं आहे. Assembly line वर प्रत्येक item येतो, worker त्यावर same operation करतो. map() पण तसंच – list च्या प्रत्येक element वर same function apply करतो.

Syntax:

```
map(function, iterable)
```

Without map – Problem:

```
numbers = [1, 2, 3, 4, 5]

squared = []
```

```
for num in numbers:

    squared.append(num ** 2)    # manually loop – boring!

print(squared)    → [1, 4, 9, 16, 25]
```

With map – Solution:

```
numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x ** 2, numbers))

print(squared)    → [1, 4, 9, 16, 25]

# एकाच line मध्ये! ✓
```

More Examples:

```
# Cube of each element:

l = [1, 2, 4, 6, 4, 3]

cubes = list(map(lambda x: x ** 3, l))

print(cubes)    → [1, 8, 64, 216, 64, 27]


# String ला uppercase:

names = ["rohit", "rushikesh", "sakshi"]

upper = list(map(lambda s: s.upper(), names))

print(upper)    → ['ROHIT', 'RUSHIKESH', 'SAKSHI']


# Celsius ते Fahrenheit:
```

```
celsius = [0, 20, 37, 100]

fahrenheit = list(map(lambda c: (c * 9/5) + 32, celsius))

print(fahrenheit) → [32.0, 68.0, 98.6, 212.0]

# Numbers double करणे:

nums = [5, 10, 15, 20]

doubled = list(map(lambda x: x * 2, nums))

print(doubled) → [10, 20, 30, 40]
```

⚠ Important – map() एक map object return करतो. List बघायला list() मध्ये convert करावं लागतो! list(map(...))

filter() – Condition पूर्ण करणारे Elements काढणे

filter() म्हणजे चाळणी सारखं आहे. तू मैदा चाळतोस – मोठे कण बाहेर राहतात, बारीक आत जातात. filter() पण तसंच – condition True असेल तेच elements राहतात, बाकी जातात.

Syntax:

```
filter(predicate, iterable)
```

Without filter – Problem:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even = []

for num in numbers:
```

```
if num % 2 == 0:

    even.append(num)

print(even)    → [2, 4, 6, 8, 10]
```

With filter – Solution:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even = list(filter(lambda x: x % 2 == 0, numbers))

print(even)    → [2, 4, 6, 8, 10]

# एकाच line मध्ये! ✓
```

More Examples:

```
# 3 पेक्षा मोठे numbers:

l = [1, 2, 4, 6, 8, 13]

result = list(filter(lambda a: a > 3, l))

print(result)    → [4, 6, 8, 13]

# Positive numbers only:

nums = [-5, -3, 0, 2, 4, 7]

positive = list(filter(lambda x: x > 0, nums))

print(positive)    → [2, 4, 7]

# 5 पेक्षा जास्त length असलेले words:
```

```
words = ["hi", "python", "code", "programming", "AI"]
long_words = list(filter(lambda w: len(w) > 5, words))
print(long_words) → ['python', 'programming']

# Odd numbers:

nums = [1, 2, 3, 4, 5, 6, 7, 8, 9]
odd = list(filter(lambda x: x % 2 != 0, nums))
print(odd) → [1, 3, 5, 7, 9]
```

💡 map vs filter फरक: map() → प्रत्येक element transform करतो (सगळे येतात). filter() → condition पूर्ण करणारेच येतात (कमी येतात).

4. reduce() – List ला एका Value मध्ये Reduce करणे

reduce() म्हणजे snowball सारखं आहे. डोंगरावरून खाली येताना प्रत्येक वेळी snow घेऊन मोठा होतो. reduce() पण तसंच – list मधले elements एकत्र combine करत जातो जोपर्यंत एकच value राहत नाही.

⚠️ reduce() built-in नाही – functools module मधून import करावं लागतं!

Syntax: reduce(function, iterable)

```
from functools import reduce
```

► **Step by Step कसं काम करतो:**

```
from functools import reduce
```

```
numbers = [1, 2, 3, 4, 5]
```

```
sum = reduce(lambda x, y: x + y, numbers)
```

```
print(sum) → 15
```

Step by step:

→ Step 1: $1 + 2 = 3$

→ Step 2: $3 + 3 = 6$

→ Step 3: $6 + 4 = 10$

→ Step 4: $10 + 5 = 15$ # Final answer!

More Examples:

```
from functools import reduce
```

Maximum number find करने:

```
nums = [3, 7, 2, 9, 4, 1]
```

```
maximum = reduce(lambda x, y: x if x > y else y, nums)
```

```
print(maximum) → 9
```

सर्व numbers multiply:

```
nums = [1, 2, 3, 4, 5]
```

```
product = reduce(lambda x, y: x * y, nums)
```

```
print(product) → 120 (1x2x3x4x5)
```

Strings join करने:

```
words = ["Hello", " ", "World", "!"]

sentence = reduce(lambda x, y: x + y, words)

print(sentence) → Hello World!
```

map vs filter vs reduce

Function	મ્હણજે	Input → Output	Example
map()	Transform each element	[1,2,3] → [1,4,9]	Square of each
filter()	Select elements only	[1,2,3,4] → [2,4]	Even only
reduce()	Combine to one value	[1,2,3,4] → 10	Sum of all

Chapter 12 – Exception Handling

Exception Handling

Program run होताना unexpected errors येतात – यांना Exceptions म्हणतात. Exception handling म्हणजे या errors ला gracefully handle करणं – program crash होऊ न देणं आणि user ला meaningful message देणं.

Common Exception Types:

Exception	कधी येतो
ValueError	चुकीचा value – जसं <code>int("abc")</code>
ZeroDivisionError	0 ने divide केलं
TypeError	चुकीचा data type
KeyError	Dictionary मध्ये key नाही
IndexError	List चा invalid index
FileNotFoundError	File सापडली नाही
NameError	Variable define केलेला नाही
AttributeError	Object चा attribute नाही

► try-except Block

- try block – तो code लिहितात जो error येऊ शकतो.
- except block – error आल्यावर हा code run होतो.

Structure:

```
try:

    # error येऊ शकणारा code

except:

    # error आल्यावर काय करायचं
```

Example – Multiplication Table:

```
a = input("Enter the number: ")
print(f"Multiplication table of {a} is:")

try:
    for i in range(1, 10):
        print(f"{a} x {i} = {int(a) * i}")
except Exception as e:
    print("Sorry some error occurred")

print("Some imp lines of code")
print("End of Program")
```

Case 1 – User ने '5' टाकलं:

```
→ 5 x 1 = 5
→ 5 x 2 = 10 ... and so on
→ Some imp lines of code
→ End of Program
```

Case 2 – User ने 'abc' टाकलं:

```
→ Sorry some error occurred # except block run झाला
→ Some imp lines of code    # program crash नाही झाला!
→ End of Program
```

💡 try-except शिवाय 'abc' टाकलं असतं तर program crash झाला असता!
try-except मुळे program gracefully continue होतो.

Specific Exception catch करणे:

```
try:
    num = int(input("Enter number: "))
    result = 10 / num
    print(result)
except ValueError:
    print("Please enter a valid number!")
```

```
except ZeroDivisionError:

    print("Cannot divide by zero!")

except Exception as e:

    print(f"Unexpected error: {e}")
```

💡 वेगळ्या errors साठी वेगळे except blocks वापरून specific messages देता येतात!

else Block

else block हा try...except सोबत वापरला जातो. जर try block मध्ये कोणतीही exception आली नाही, तर else block execute होतो. जर exception आली, तर except block execute होतो आणि else block run होत नाही.

Syntax:

```
try:

    # risky code

except ExceptionType:

    # handle exception

else:

    # execute if no exception occurs
```

```
try:

    num = int(input("Enter a number: "))

    print("You entered:", num)

except ValueError:

    print("Invalid Input!")

else:

    print("Program executed successfully.")
```

Output 1 (Valid Input):

```
Enter a number: 25

You entered: 25

Program executed successfully.
```

Output 2 (Invalid Input):

```
Enter a number: abc

Invalid Input!
```

💡 else block फक्त तेव्हाच execute होतो जेव्हा try block मध्ये कोणतीही exception येत नाही.

finally Block

finally block हा try...except सोबत वापरला जातो. Exception आली किंवा नाही आली, तरी finally block नेहमी execute होतो. Cleanup कामांसाठी, जसे file बंद करणे किंवा database connection close करणे, finally block वापरतात.

Syntax:

```
try:

    # risky code

except ExceptionType:

    # handle exception

finally:

    # always executes
```

```
try:

    num = 10 / 0

except ZeroDivisionError:

    print("Cannot divide by zero.")

finally:

    print("Program Finished.")
```

Output:

```
Cannot divide by zero.

Program Finished.
```

Another Example:

```
try:  
    print("Hello Rohit")  
  
finally:  
    print("Execution Completed.")
```

Output:

```
Hello Rohit  
  
Execution Completed.
```

💡 finally block नेहमी execute होतो, मग exception आली असो किंवा नसो.

raise – Custom Error Raise करणे

कधी कधी आपल्याला स्वतः error raise करायचा असतो – specific invalid condition मध्ये program ला सांगायचं असतं की 'हे चुकीचं आहे, पुढे जाऊ नकोस.'

```
salary = int(input("Enter salary amount: "))  
  
if not 2000 < salary < 5000:  
    raise ValueError("Not a Valid salary")  
  
# salary = 1500 → ValueError: Not a Valid salary  
  
# salary = 3000 → code पुढे चालू राहतो
```

💡 raise keyword वापरून कोणताही built-in exception manually trigger करता येतो. Input validation साठी खूप useful!

Defining Custom Exceptions

Built-in exceptions नाही पुरत तेव्हा स्वतःचे custom exception classes बनवता येतात. हे Exception class inherit करून बनवतात.

Syntax:

```
class CustomError(Exception):  
    # code  
  
    pass  
  
try:  
    # code  
  
except CustomError:  
    # code
```

Example:

```
class InvalidAgeError(Exception):  
    pass  
  
age = int(input("Enter age: "))  
  
try:  
    if age < 0:  
        raise InvalidAgeError("Age cannot be negative!")  
  
    if age > 150:
```

```
        raise InvalidAgeError("Age cannot be more than 150!")

    print(f"Valid age: {age}")

except InvalidAgeError as e:

    print(f"Error: {e}")
```

💡 Custom Exceptions – Code more readable, specific errors ला specific names, Large projects मध्ये debugging easy होतो!

Chapter 13 – Modules, Packages & Virtual Environments

What is Module?

Module म्हणजे एक Python file (.py) जी related functions, variables आणि classes एकत्र ठेवते – जेणेकरून त्या code ला दुसऱ्या program मध्ये reuse करता येईल.

Real-life Example:

math module मध्ये mathematical functions आहेत, random module मध्ये random number generation आहे – तुम्हाला scratch पासून बनवायला नको!

Module वापरण्याचे फायदे:

- Code reuse – एकदा लिहा, अनेक ठिकाणी वापरा
- Organized code – related code एकाच file मध्ये
- Easy maintenance – एका ठिकाणी बदल की सगळीकडे update

Types of Modules

Python मध्ये modules तीन प्रकारचे असतात:

Built-in Modules – Python सोबतच येतात, वेगळं install करायला नको.

```
import math  
  
import random  
  
import os  
  
import datetime
```

External Modules – Third-party लोकांनी बनवलेले. वापरायचे असल्यास आधी install करावं लागतं.

```
pip install pandas  
  
pip install numpy  
  
pip install requests
```

User-defined Modules – तुम्ही स्वतः बनवलेले modules.

```
# तुमची स्वतःची greet.py file म्हणजेच module!
```

Creating Your Own Module

स्वतःचा module बनवणं खूप सोपं आहे – एक नवीन .py file बनवा आणि त्यात functions लिहा!

Step 1 – mymodule.py नावाची file बनवा:

```
# mymodule.py  
  
def greet(name):  
    print(f"Hello, {name}!")  
  
def add(a, b):  
    return a + b  
  
fav_language = "Python"
```

Step 2 – दुसऱ्या file मध्ये import करा:

```
import mymodule

mymodule.greet("Rohit")    → Hello, Rohit!

print(mymodule.add(3, 4))  → 7

print(mymodule.fav_language) → Python
```

💡 दोन्ही files एकाच folder मध्ये असाव्यात!

Importing Modules

Python मध्ये modules import करण्याचे अनेक ways आहेत. प्रत्येकाचे फायदे वेगळे आहेत.

1) Basic import

सर्वात simple way – पूर्ण module import होतो. Use करताना module_name.function() असं लिहावं लागतं.

```
import math

result = math.sqrt(9)

print(result)    → 3.0

print(math.pi)  → 3.141592653589793
```

2) from keyword – Specific Import

Specific function किंवा variable import करायला from keyword वापरतात. पूर्ण module load होत नाही – फक्त हवं ते import होतं. Memory efficient आहे.

```
from math import sqrt, pi

result = sqrt(9)

print(result)    → 3.0

print(pi)        → 3.141592653589793

# math.sqrt() ऐवजी directly sqrt() – simpler!
```

3) Importing Everything – *(Wildcard)

* वापरून module मधलं सगळं import करता येतं. पण हे generally not recommended!

```
from math import *

result = sqrt(9) * pi

print(result)    → 9.42477796076938
```

⚠ * का टाळायचं? कोणती function कुठून आली ते कळत नाही. Two modules मध्ये same name असेल तर conflict होतो. Code समजायला कठीण होतो!

4) as keyword – Alias

Module ला short nickname द्यायला as वापरतात. Long module names साठी खूप उपयुक्त आहे.

```
import math as m

result = m.sqrt(9)

print(result)

print(m.pi)

# Data Science मध्ये popular aliases:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

💡 as वापरल्यामुळे code shorter आणि readable होतो!

5. dir() Function

dir() function module मधल्या सगळ्या available functions आणि variables ची list दाखवतो.

```
import math

print(dir(math))

→ ['__doc__', '__loader__', 'acos', 'acosh', 'asin',

    'atan', 'ceil', 'cos', 'degrees', 'e', 'exp',

    'floor', 'log', 'pi', 'pow', 'radians', 'sin',

    'sqrt', 'tan', ...]

print(math.nan)    → nan
```

💡 नवीन module explore करताना `dir()` खूप उपयुक्त आहे – module मध्ये नक्की काय काय available आहे ते एका नजरेत दिसतं!

6. `if name == "__main__"`

हे Python मधला एक खूप important pattern आहे जो professional code मध्ये नेहमी दिसतो.

Problem काय आहे?

तू एक Python file `helper.py` बनवतोस ज्यात काही functions आहेत. तू ती directly run करतोस – सगळं code run होतं. पण दुसऱ्या file ने हीच file import केली (`import helper`) तर तिचा code पण automatically run होतो! हे नको असतं.

`__name__` variable म्हणजे काय?

Python मध्ये **`__name__`** हा built-in variable automatically set होतो:

- File directly run केली → `__name__ = "__main__"`
- File import केली → `__name__ = file चं नाव (module name)`

```
def main():  
    print("Running script directly")  
  
if __name__ == "__main__":  
    main()
```

Case 1 – File directly run केली:

```
__name__ = "__main__"
```

→ if condition True → main() call होतो ✓

→ "Running script directly" print होतो

Case 2 – File import केली (import myfile):

```
__name__ = "myfile"
```

→ if condition False → main() call होत नाही ✓

→ काहीही print होत नाही

💡 हे pattern वापरल्यामुळे तुमचा module safely import करता येतो – unwanted code run होत नाही!

7. What is Package?

Package म्हणजे modules चा एक organized collection – एक folder ज्यात related modules एकत्र ठेवलेले असतात.

Package Structure:

```
mypackage/
```

```
    __init__.py    ← हे file असणं जरूरी आहे!
```

```
    greet.py
```

```
    math_utils.py
```

```
    file_utils.py
```

⚠ `__init__.py` – ही file folder ला Python package म्हणून ओळखवते. ती रिकामी असली तरी चालते – पण ती असणं आवश्यक आहे. Python 3.3 पूर्वी ती mandatory होती. आता technically mandatory नाही (namespace packages), पण best practice म्हणून ठेवा – यामुळे code clear आणि organized राहतो!

Package import करणे:

```
from mypackage import greet

from mypackage.math_utils import add

greet.greet("Rohit")

print(add(3, 4))
```

💡 Package म्हणजे modules ची library – बड्या projects मध्ये code organized ठेवायला packages खूप उपयुक्त असतात!

8. pip – Python Package Installer

pip म्हणजे Python चा official package manager – इंटरनेटवरून external packages download आणि install करायला वापरतात.

Common pip Commands:

```
pip install package_name      # Install

pip uninstall package_name    # Uninstall

pip install package==1.0.0    # Specific version install
```

```
pip install --upgrade package # Update to latest version

pip list                        # Installed packages list

pip show package_name         # Package info

pip freeze                    # requirements list (for sharing)

pip freeze > requirements.txt # Save to file

pip install -r requirements.txt # Install from file
```

Examples:

```
pip install pandas
pip install numpy
pip install requests
pip uninstall pandas
pip list
→ Package    Version
-----
numpy        1.24.0
pandas       2.0.0
```

💡 requirements.txt म्हणजे project च्या सगळ्या dependencies ची list –
दुसऱ्याला project share केल्यावर ते pip install -r requirements.txt run
करतात आणि सगळं automatically install होतं!

9. Virtual Environment

Virtual Environment म्हणजे एका Python project साठी isolated, independent environment. प्रत्येक project चे स्वतःचे packages आणि versions असतात – दुसऱ्या projects वर effect होत नाही.

का लागतो?

Project A ला Django 3.0 हवा आणि Project B ला Django 4.0 हवा – दोन्ही एकाच system वर कसे install करायचे? Virtual Environment!

Virtual Environment – Commands:

```
# Create:

python -m venv myenv

# Activate (Windows):

myenv\Scripts\activate

# Activate (Mac/Linux):

source myenv/bin/activate

# Install inside env:

pip install package_name

# Check installed packages:

pip list

# Deactivate:

deactivate
```

Delete (simply delete folder):

```
rm -rf myenv
```

Visual Representation:

System Python

```
├── Project A (venv_a)
│   ├── Django 3.0
│   └── numpy 1.20
└── Project B (venv_b)
    ├── Django 4.0
    └── numpy 1.24
```

💡 Best Practice – प्रत्येक नवीन project साठी स्वतःचा virtual environment बनवा. यामुळे package conflicts कधीच होणार नाहीत!

⚠️ Virtual environment activate केल्यावर terminal मध्ये (myenv) असं दिसेल – म्हणजे तुम्ही virtual environment च्या आत आहात.

Chapter 14 – File Handling & OS Operations

File Handling – Opening a File

File handling म्हणजे – Python program मधून files create करणं, read करणं, write करणं आणि close करणं. Real world applications मध्ये data persistent ठेवायला files वापरतात.

open() Function – Syntax:

```
f = open('filename.txt', 'mode')
```

File Modes

Mode	Symbol	काय करतो	File नसेल तर
Read	r	File read करतो	Error येतो
Write	w	File write करतो (overwrite!)	नवीन बनवतो
Append	a	शेवटी content add करतो	नवीन बनवतो
Create	x	नवीन file बनवतो	–
Text	t	Text mode (default)	–

⚠ w mode मध्ये file open केली तर existing content permanently delete होतो! Append a mode safe आहे existing content साठी.

read() – पूर्ण File Read करणे

```
# Method 1 – Manual close:  
f = open('myfile.txt', 'r')  
text = f.read()  
print(text)  
f.close() # नेहमी close करणं important – memory leak टाळायला!
```

read() method पूर्ण file एकदाच read करतो आणि string म्हणून return करतो.

```
# Method 2 – with statement (Recommended!):
with open('myfile.txt', 'r') as f:
    text = f.read()
    print(text)
# with block संपल्यावर file automatically close होतो!

# Partial read:
with open('myfile.txt', 'r') as f:
    first_10 = f.read(10)    # पहिले 10 characters read
    print(first_10)
```

💡 with statement नेहमी prefer कर – file automatically close होते, exception आली तरी!

readline() – एक Line Read करणे

readline() method एक वेळी एक line read करतो. Multiple lines साठी loop वापरतात.

```
f = open('myfile.txt', 'r')
while True:
    line = f.readline()
    if not line:        # empty string म्हणजे file संपली
        break
    print(line, end="") # end="" – extra blank line टाळण्यासाठी
f.close()

# readlines() – सर्व lines list मध्ये:
with open('myfile.txt', 'r') as f:
    lines = f.readlines()
    for line in lines:
        print(line.strip()) # strip() – \n remove करतो
```

writelines() – Multiple Lines Write करणे

writelines() method strings च्या list मधल्या सर्व strings file मध्ये write करतो. write() एकच string लिहितो, writelines() list लिहितो.

```
with open('myfile.txt', 'w') as f:
    lines = ['Hello\n', 'World\n', 'Python\n']
    f.writelines(lines)
# ⚠️ \n नाही दिलं तर सगळं एकाच line मध्ये येतं!
# चुकीचं:
lines = ['line1', 'line2', 'line3']
→ File मध्ये: "line1line2line3"
# बरोबर:
lines = ['line1\n', 'line2\n', 'line3\n']
→ File मध्ये: प्रत्येक वेगळ्या line वर!
```

seek() – File मध्ये Position Move करणे

seek() function म्हणजे file मध्ये cursor specific byte position वर move करतो. File एक tape सारखी आहे – seek() ने tape च्या कोणत्याही position वर jump करता येतो.

Syntax:

```
f.seek(offset, whence=0)
```

- offset – कितव्या byte वर जायचं
- whence – 0 = beginning (default), 1 = current, 2 = end

```
with open('myfile.txt', 'r') as f:
    f.seek(10)          # 10th byte वर जा
    data = f.read(5)    # त्यानंतरचे 5 bytes read
    print(data)
    f.seek(0)          # Beginning ला परत जा
    full = f.read()
    print(full)
# File दोनदा read करणे:
```

```
with open('myfile.txt', 'r') as f:
    first_read = f.read()
    f.seek(0)    # Beginning ला परत जा
    second_read = f.read()    # परत read होतं!
```

💡 seek(0) म्हणजे file च्या beginning ला जा – file परत सुरुवातीपासून read करायला खूप useful!

tell() – Current Position जाणून घेणे

tell() function म्हणजे file मध्ये सध्या cursor कुठे आहे ते byte number मध्ये सांगतो. seek() सोबत वापरून specific positions bookmark करता येतात.

```
with open('file.txt', 'r') as f:
    print(f.tell())    → 0    (beginning)

    data = f.read(10)    # पहिले 10 bytes read
    print(f.tell())    → 10    (10 bytes पुढे)

    # Position save करा
    current_position = f.tell()

    # आणखी read करा
    more_data = f.read(5)
    print(f.tell())    → 15

    # Saved position वर परत जा
    f.seek(current_position)
    print(f.tell())    → 10    (परत 10 वर)
```

💡 seek() आणि tell() एकत्र वापरून file च्या specific positions वर precisely navigate करता येतो – large files process करताना खूप useful!

truncate() – File Cut करणे

truncate() म्हणजे file ला specified size पर्यंत cut करणं. जास्तीचा content permanently delete होतो.

```

with open('sample.txt', 'w') as f:
    f.write('Hello World!')    # 12 characters लिहिलं
    f.truncate(5)              # फक्त पहिले 5 bytes ठेव
with open('sample.txt', 'r') as f:
    print(f.read())
→ Hello    # "Hello World!" मधून फक्त "Hello" राहिलं!
# File empty करणे:
with open('myfile.txt', 'w') as f:
    f.truncate(0)    # पूर्ण empty

```

💡 truncate(0) म्हणजे file पूर्णपणे empty करणं – content delete पण file exist राहते!

File Handling Methods Summary

Method	काय करतो
open(file, mode)	File open करतो
f.read()	पूर्ण file read
f.read(n)	n characters read
f.readline()	एक line read
f.readlines()	सर्व lines list मध्ये
f.write(str)	String write
f.writelines(list)	List of strings write
f.seek(n)	n position वर जा
f.tell()	Current position
f.truncate(n)	n bytes पर्यंत cut
f.close()	File close

OS Module in Python

OS module म्हणजे Python चा एक powerful built-in library जो operating system सोबत directly interact करण्यासाठी functions provide करतो. Python code वापरून folders बनवणे, files delete करणे, system information मिळवणे – हे सगळं os module ने करता येतं.

OS Module का महत्वाचं आहे?

- Real world projects मध्ये data files मध्ये save करावी लागते
- Project structure बनवणे, logs manage करणे
- Automation scripts, web scrapers, data pipelines – सगळीकडे वापरतात

```
import os    # नेहमी आधी import करावं लागतो
```

1) os.mkdir() – Directory बनवणे

mkdir() म्हणजे 'make directory'. एक नवीन folder बनवतो. Folder already असेल तर FileExistsError येतो.

```
import os
os.mkdir("data")
# "data" नावाचा folder current directory मध्ये बनेल
```

Practical Example – 100 sub-folders बनवणे:

```
import os
# आधी "data" folder आहे का check करा
if not os.path.exists("data"):
    os.mkdir("data")
    print("data folder created!")
# 100 sub-folders बनवा
for i in range(0, 100):
    os.mkdir(f"data/day{i}")
    # data/day0, data/day1, ... data/day99 बनतील
```

💡 os.path.exists() आधी check करणं best practice आहे – FileExistsError टाळण्यासाठी!

2) `os.path.exists()` – File/Folder Check करणे

`os.path.exists()` म्हणजे – दिलेल्या path वर file किंवा folder exist करतो का ते check करतो. True (exist) किंवा False (not exist) येतो.

```
import os

# File check:
print(os.path.exists("myfile.txt")) → True or False

# Practical use:
if not os.path.exists("output"):
    os.mkdir("output")
    print("output folder created!")
else:
    print("output folder already exists!")
```

💡 File operations करण्याआधी नेहमी `os.path.exists()` check करायला हवं – errors टाळण्यासाठी!

3) `os.rename()` – File/Folder Rename करणे

`rename()` म्हणजे file किंवा folder चं नाव बदलणं. Different path दिली तर file move पण होते.

```
import os

# Simple Rename:
os.rename("Rohit", "Shambheu")
# "Rohit" नावाचा file/folder → "Shambheu" rename झाला

# Move + Rename:
os.rename("old_folder/file.txt", "new_folder/file.txt")
# file एका folder मधून दुसऱ्या folder मध्ये move झाली

# Safe way:
if os.path.exists("Rohit"):
    os.rename("Rohit", "Shambheu")
    print("Renamed successfully!")
```

4) `os.getcwd()` – Current Directory बघणे

`getcwd()` म्हणजे 'Get Current Working Directory'. Python आता कोणत्या folder मध्ये काम करतोय ते सांगतो.

```
import os
current = os.getcwd()
print(current)
→ C:\Users\Rohit\Desktop\Python Projects (Windows)
→ /home/rohit/python_projects (Linux/Mac)
```

💡 File open करताना relative path वापरली तर ती current working directory च्या relative असते. `getcwd()` कुठे आहेत ते जाणून घ्यायला important!

5) `os.chdir()` – Directory बदलणे

`chdir()` म्हणजे 'Change Directory'. Current working directory बदलतो.

```
import os
print(os.getcwd())          → /home/rohit
os.chdir("/home/rohit/documents")
print(os.getcwd())          → /home/rohit/documents
# Relative path ने पण:
os.chdir("projects")      # /home/rohit/documents/projects ला जाईल
```

6) `os.listdir()` – Directory Contents बघणे

`listdir()` म्हणजे एखाद्या directory मधल्या सर्व files आणि folders ची list देतो.

```
import os
# Current directory:
files = os.listdir(".")
print(files)
→ ['file1.txt', 'file2.py', 'folder1', ...]
# फक्त .py files filter करणे:
py_files = [f for f in os.listdir(".") if f.endswith(".py")]
print(py_files)
→ ['main.py', 'helper.py', 'test.py']
```

7) os.remove() – File Delete करणे

remove() म्हणजे file permanently delete करतो. Recycle Bin मध्ये जात नाही – direct delete!

```
import os
# Safe way – नेहमी हेच वापर:
if os.path.exists("myfile.txt"):
    os.remove("myfile.txt")
    print("File deleted successfully!")
else:
    print("File not found!")
```

⚠ os.remove() फक्त files साठी आहे – folders साठी नाही. Folder delete करायला os.rmdir() किंवा shutil.rmtree() वापरतात!

8) os.rmdir() – Empty Directory Delete करणे

rmdir() म्हणजे empty folder delete करतो. Folder मध्ये काहीही असेल तर OSError येतो.

```
import os
# Safe way:
if os.path.exists("empty_folder"):
    try:
        os.rmdir("empty_folder")
        print("Empty folder deleted!")
    except OSError:
        print("Folder is not empty!")
```

9) shutil.rmtree() – Files सकट Directory Delete करणे

shutil हा separate module आहे. rmtree() म्हणजे folder आणि त्यातल्या सर्व files/sub-folders सकट permanently delete करतो.

```
import shutil, os
# Safe way:
if os.path.exists("data_folder"):
    confirm = input("Are you sure? (yes/no): ")
    if confirm == "yes":
        shutil.rmtree("data_folder")
        print("Deleted!")
```

⚠ Extremely dangerous – permanent delete, recovery नाही! Production code मध्ये खूप carefully वापर.

10) `os.environ` / `os.getenv()` – Environment Variables

Environment variables म्हणजे operating system मध्ये stored key-value pairs. Database passwords, API keys secure ठेवायला environment variables वापरतात.

```
import os
# Specific variable:
print(os.environ['PATH'])
print(os.environ['HOME'])
# Safe way – key नसेल तर None येतो, KeyError नाही:
print(os.getenv('DATABASE_URL'))          → None
print(os.getenv('PORT', '8080'))          → 8080 (default value)
```

💡 API keys, database passwords code मध्ये hardcode करू नये – environment variables मध्ये ठेव आणि `os.getenv()` ने access कर. हे security best practice आहे!

11) System Information

```
import os
print(os.getpid())      # Current process ID → 12345
print(os.getlogin())    # Current logged in user → 'rohit'
print(os.getppid())     # Parent process ID
print(os.getuid())      # User ID (Unix/Linux)
print(os.getgid())      # Group ID (Unix/Linux)
```

★ OS Module Quick Reference

Function	काय करतो	Error येतो कधी
os.mkdir(path)	नवीन directory बनवतो	Already exists
os.path.exists(path)	Exist check करतो	कधीच नाही
os.rename(old, new)	Rename/Move करतो	Old नसेल तर
os.getcwd()	Current directory	कधीच नाही
os.chdir(path)	Directory बदलतो	Path नसेल तर
os.listdir(path)	Contents list	Path नसेल तर
os.remove(file)	File delete	File नसेल तर
os.rmdir(dir)	Empty dir delete	Empty नसेल तर
shutil.rmtree(dir)	सगळं delete	Path नसेल तर
os.getenv(key)	Specific env var	कधीच नाही
os.getpid()	Process ID	कधीच नाही

OS Module Other Functions (Reference)

<code>os.ctermid()</code>	# Terminal file name
<code>os.fchdir()</code>	# File descriptor ने directory बदल
<code>os.fsencode()</code> / <code>os.fsdecode()</code>	# Filename encode/decode
<code>os.fspath()</code>	# File system path
<code>os.getenvb()</code>	# Bytes env variable
<code>os.get_exec_path()</code>	# Executable search path
<code>os.getegid()</code> / <code>os.geteuid()</code>	# Effective group/user ID
<code>os.getgrouplist()</code> / <code>os.getgroups()</code>	# User groups
<code>os.getpgid(pid)</code> / <code>os.getppid()</code>	# Process group/parent ID
<code>os.getpriority(which, who)</code>	# Process priority
<code>os.getresuid()</code> / <code>os.getresgid()</code>	# Real, effective, saved IDs
<code>os.putenv(key, value)</code>	# Environment variable set
<code>os.setegid()</code> / <code>os.setgid()</code>	# Group ID set
<code>os.setpriority(which, who, p)</code>	# Priority set

Chapter 15 – Object-Oriented Programming

Introduction to OOP

आतापर्यंत आपण procedural programming केली – म्हणजे एक एक instruction लिहून program बनवला. पण real world मध्ये गोष्टी अशा work करत नाहीत.

Object-Oriented Programming (OOP) म्हणजे real world objects प्रमाणे code organize करणं. Data आणि functions एकाच object मध्ये असतात. Code reusable, organized आणि maintainable होतो.

OOP चे Core Pillars:

Pillar	म्हणजे काय
Encapsulation	Data आणि methods एकत्र wrap करणं
Inheritance	Parent class कडून properties घेणं
Polymorphism	Same method, वेगळं behavior
Abstraction	फक्त जरूरी details दाखवणं

💡 OOP का शिकायचं? – Code reuse होतो, मोठे projects manage करणं सोपं होतं, real world problems सहज model करता येतात!

Note: Abstraction is one of the four pillars of OOP. It is not covered in detail in this chapter.

Class and Object

Class म्हणजे काय? – Blueprint!

Class म्हणजे एक blueprint (नकाशा). जसं घराचा नकाशा म्हणजे instructions आहेत, actual घर नाही. त्या नकाशावरून अनेक घरं बांधता येतात. तसंच class एक template आहे ज्यावरून अनेक objects बनवता येतात.

```
class ClassName:

    # attributes (data)

    # methods (functions)
```

Object म्हणजे काय? – Instance!

Object म्हणजे class चा instance – blueprint वापरून बनवलेलं actual 'घर'. एका class वरून अनेक objects बनवता येतात – प्रत्येकाचे values वेगळे असू शकतात.

```
# Class बनवणे:

class Dog:

    name = "Tommy"

    breed = "Labrador"

    def bark(self):

        print(f"{self.name} says: Woof!")

# Object बनवणे:

dog1 = Dog()

print(dog1.name)    → Tommy

dog1.bark()         → Tommy says: Woof!
```

Real World Example:

Class (Blueprint)	Object (Instance)
Car	माझी Red Toyota
Student	Rohit, Sakshi
BankAccount	Account No. 12345
Phone	iPhone 16, Samsung S24

💡 एकाच class वरून अनेक objects बनवता येतात – प्रत्येक object स्वतंत्र असतो!

Creating Objects

Object बनवणं अत्यंत सोपं आहे – class चं नाव लिहून () लावायचं.

```
# Simple class:
class Details:
    name = "Rohit"
    age = 18
    def dec(self):
        print(f"my name is {self.name} and I'm {self.age} years old.")
# Object बनवणे:
obj = Details()
# Attributes access:
print(obj.name)    → Rohit
```

```
print(obj.age)    → 18
```

```
# Method call:
```

```
obj.dec()         → my name is Rohit and I'm 18 years old.
```

Multiple Objects एकाच Class मधून:

```
class Car:
```

```
    def __init__(self, brand, color):
```

```
        self.brand = brand
```

```
        self.color = color
```

```
    def describe(self):
```

```
        print(f'I have a {self.color} {self.brand}')
```

```
car1 = Car("Toyota", "Red")
```

```
car2 = Car("Honda", "Blue")
```

```
car3 = Car("BMW", "Black")
```

```
car1.describe()    → I have a Red Toyota
```

```
car2.describe()    → I have a Blue Honda
```

```
car3.describe()    → I have a Black BMW
```

💡 एकाच blueprint (class) वरून 3 वेगळ्या cars (objects) बनवल्या –
प्रत्येकाचे values वेगळे!

Constructors (init)

Constructor म्हणजे एक special method जो object बनताना automatically call होतो.

तू hotel मध्ये check-in करतोस – receptionist automatically room key देतो, forms fill करतो. `__init__` पण तसंच – object बनताना automatically initial setup करतो!

Simple Constructor:

```
class ABC:

    def __init__(self):

        print("Object created!")

obj = ABC()    → Object created!

# कुठेही call नाही केला पण automatically run झाला!
```

Constructor with Parameters:

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

        print(f"Person {name} created!")

p1 = Person("Rohit", 18)    → Person Rohit created!

p2 = Person("Sakshi", 20)   → Person Sakshi created!

print(p1.name)    → Rohit

print(p2.name)    → Sakshi
```

Practical Example – BankAccount:

```
class BankAccount:

    def __init__(self, owner, balance=0):

        self.owner = owner

        self.balance = balance

        print(f"Account created for {owner}")

    def deposit(self, amount):

        self.balance += amount

        print(f"Deposited {amount}. Balance: {self.balance}")

    def withdraw(self, amount):

        if amount > self.balance:

            print("Insufficient balance!")

        else:

            self.balance -= amount

            print(f"Withdrawn {amount}. Balance: {self.balance}")

acc1 = BankAccount("Rohit", 1000)    → Account created for Rohit
acc2 = BankAccount("Sakshi")        → Account created for Sakshi
acc1.deposit(500)                    → Deposited 500. Balance: 1500
acc1.withdraw(200)                   → Withdrawn 200. Balance: 1300
acc2.withdraw(100)                   → Insufficient balance!
```

💡 balance=0 म्हणजे default value – Sakshi ने balance न दिल्यामुळे automatically 0 set झाला!

Instance Variables

Instance variables म्हणजे प्रत्येक object साठी वेगळ्या variables – एका object मधला बदल दुसऱ्यावर affect होत नाही.

__init__ मध्ये self. वापरून define करतात.

```
class Student:

    def __init__(self, name, marks):

        self.name = name          # instance variable

        self.marks = marks       # instance variable

s1 = Student("Rohit", 95)

s2 = Student("Sakshi", 88)

s3 = Student("Raj", 72)

print(s1.name, s1.marks)    → Rohit 95

print(s2.name, s2.marks)    → Sakshi 88

print(s3.name, s3.marks)    → Raj 72

# s1 चं marks बदलवलं:

s1.marks = 98

print(s1.marks)    → 98

print(s2.marks)    → 88    # s2 वर काहीच effect नाही!
```

⚠ Instance variables प्रत्येक object साठी स्वतंत्र असतात – एक बदलला तरी दुसरा तसाच राहतो!

Class Variables

Class variables म्हणजे सगळ्या objects साठी common असलेले variables – एका ठिकाणी बदललं की सगळीकडे बदलतं.

Class च्या आत, `__init__` च्या बाहेर define करतात.

```
class Student:

    school = "Solapur High School"    # class variable

    def __init__(self, name):

        self.name = name                # instance variable

s1 = Student("Rohit")
s2 = Student("Sakshi")

print(s1.school)    → Solapur High School
print(s2.school)    → Solapur High School
print(Student.school) → Solapur High School

# Class variable बदवला – सगळ्यांना effect:

Student.school = "Pune High School"

print(s1.school)    → Pune High School
print(s2.school)    → Pune High School
```

#	Instance Variable	Class Variable
Define	__init__ मध्ये self. वापरून	Class मध्ये directly
Scope	फक्त त्या object साठी	सगळ्या objects साठी
Access	obj.variable	Class.variable

💡 Counter, school name, company name – जे सगळ्यांसाठी same असतं ते class variable मध्ये ठेवा!

Instance Methods

Instance methods म्हणजे class मध्ये define केलेले functions जे object वर काम करतात. पहिला parameter नेहमी self असतो.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
    def perimeter(self):
        return 2 * (self.length + self.width)
    def is_square(self):
        return self.length == self.width
    def describe(self):
```

```
print(f"Rectangle: {self.length} x {self.width}")
print(f"Area: {self.area()}")
print(f"Perimeter: {self.perimeter()}")
print(f"Square: {self.is_square()}")

r1 = Rectangle(5, 3)
r2 = Rectangle(4, 4)
r1.describe()
→ Rectangle: 5 x 3
→ Area: 15
→ Perimeter: 16
→ Square: False
r2.describe()
→ Rectangle: 4 x 4
→ Area: 16
→ Perimeter: 16
→ Square: True
```

💡 Instance methods `self` वापरून object च्या attributes access करतात. Method च्या आत दुसरा method पण call करता येतो – `self.area()` सारखं!

self Keyword

`self` म्हणजे 'मी स्वतः' – म्हणजे ज्या object ने method call केला तो object स्वतः. Class च्या आत method define करताना पहिला parameter नेहमी `self` असतो. Python automatically current object ला `self` म्हणून pass करतो!

```

class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def greet(self):

        print(f"Hello! I'm {self.name}")

    def birthday(self):

        self.age += 1

        print(f"Happy Birthday {self.name}! Now {self.age} years
old!")

p1 = Person("Rohit", 18)

p2 = Person("Sakshi", 20)

p1.greet()      → Hello! I'm Rohit

p2.greet()      → Hello! I'm Sakshi

p1.birthday()   → Happy Birthday Rohit! Now 19 years old!

p2.birthday()   → Happy Birthday Sakshi! Now 21 years old!

```

⚠ self हे keyword नाही – technically कोणतंही नाव चालतं. पण convention म्हणून नेहमी self वापरतात – हे Python community चं standard आहे!

Encapsulation

Encapsulation म्हणजे data आणि methods एकत्र wrap करणं आणि बाहेरून direct access पासून protect करणं.

ATM सारखं! तू directly bank vault मध्ये जाऊन पैसे घेत नाहीस – ATM (getter) वापरतोस. Deposit साठी counter (setter) वापरतोस!

Private Variables – `_` आणि `__`

```
class BankAccount:

    def __init__(self, balance):

        self._balance = balance      # _ = convention (soft private)

        self.__pin = 1234           # __ = name mangling (strong
private)

    def get_balance(self):

        return self._balance

    def deposit(self, amount):

        if amount > 0:

            self._balance += amount

acc = BankAccount(1000)

print(acc.get_balance())    → 1000

acc.deposit(500)

print(acc.get_balance())    → 1500
```

Getter आणि Setter – @property

```
class MyClass:

    def __init__(self, value):

        self._value = value

    @property

    def value(self):                # Getter

        return self._value

    @value.setter

    def value(self, new_value):    # Setter

        self._value = new_value

obj = MyClass(10)

print(obj.value)    → 10

obj.value = 20

print(obj.value)    → 20
```

Setter मध्ये Validation:

```
class Student:

    def __init__(self, name, marks):

        self._name = name

        self._marks = marks
```

```

@property

def marks(self):

    return self._marks

@marks.setter

def marks(self, new_marks):

    if new_marks < 0:

        raise ValueError("Marks cannot be negative!")

    if new_marks > 100:

        raise ValueError("Marks cannot exceed 100!")

    self._marks = new_marks

obj = Student("Rohit", 85)

print(obj.marks)    → 85

obj.marks = 95      → OK

obj.marks = -10     → ValueError: Marks cannot be negative!

obj.marks = 150     → ValueError: Marks cannot exceed 100!

```

💡 Encapsulation के फायदे: Data protection, Validation एकाच ठिकाणी,
Internal changes ने external code break होत नाही!

Inheritance

Inheritance म्हणजे एका class च्या properties आणि methods दुसऱ्या class ला मिळणं.

आई-बाबांचे गुण मुलांमध्ये येतात – तसंच parent class चे methods आणि attributes child class ला automatically मिळतात!

Basic Inheritance:

```
# Parent Class:

class Animal:

    def __init__(self, name, sound):

        self.name = name

        self.sound = sound

    def speak(self):

        print(f"{self.name} says {self.sound}")

    def eat(self):

        print(f"{self.name} is eating")

# Child Class:

class Dog(Animal):

    def fetch(self):

        print(f"{self.name} is fetching the ball!")

class Cat(Animal):

    def purr(self):
```

```
        print(f"{self.name} is purring...")

dog = Dog("Tommy", "Woof")

cat = Cat("Kitty", "Meow")

dog.speak()    → Tommy says Woof

dog.eat()      → Tommy is eating

dog.fetch()    → Tommy is fetching the ball!

cat.speak()    → Kitty says Meow

cat.purr()     → Kitty is purring...
```

Practical Example – School System:

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

    def introduce(self):

        print(f"Hi! I'm {self.name}, {self.age} years old.")

class Student(Person):

    def __init__(self, name, age, marks):

        super().__init__(name, age)

        self.marks = marks
```

```

def study(self):

    print(f"{self.name} is studying.")

def show_marks(self):

    print(f"{self.name}'s marks: {self.marks}")

class Teacher(Person):

    def __init__(self, name, age, subject):

        super().__init__(name, age)

        self.subject = subject

    def teach(self):

        print(f"{self.name} is teaching {self.subject}.")

s1 = Student("Rohit", 18, 95)

t1 = Teacher("Mr. Patil", 35, "Python")

s1.introduce()    → Hi! I'm Rohit, 18 years old.

s1.study()        → Rohit is studying.

s1.show_marks()   → Rohit's marks: 95

t1.introduce()    → Hi! I'm Mr. Patil, 35 years old.

t1.teach()        → Mr. Patil is teaching Python.

isinstance():

```

```
print(isinstance(s1, Student))    → True
print(isinstance(s1, Person))     → True
print(isinstance(t1, Student))    → False
print(isinstance(t1, Person))     → True
```

💡 Child class चा object parent class चा पण instance असतो!

Types of Inheritance

1) Single Inheritance – $A \rightarrow B$

```
class Parent:
    def parent_method(self):
        print("Parent method")

class Child(Parent):
    def child_method(self):
        print("Child method")

obj = Child()

obj.parent_method()    → Parent method
obj.child_method()     → Child method
```

2) Multilevel Inheritance – $A \rightarrow B \rightarrow C$

```
class Grandparent:
    def grandparent_method(self):
        print("Grandparent")

class Parent(Grandparent):
    def parent_method(self):
        print("Parent")

class Child(Parent):
    def child_method(self):
        print("Child")

obj = Child()

obj.grandparent_method()    → Grandparent
obj.parent_method()        → Parent
obj.child_method()          → Child
```

3) Multiple Inheritance – $A + B \rightarrow C$

```
class Father:
    def coding(self):
        print("Father knows coding")

class Mother:
    def cooking(self):
        print("Mother knows cooking")
```

```
class Child(Father, Mother):  
    def playing(self):  
        print("Child loves playing")  
  
obj = Child()  
  
obj.coding()    → Father knows coding  
obj.cooking()   → Mother knows cooking  
obj.playing()   → Child loves playing
```

4) Hierarchical Inheritance – $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$

```
class Animal:  
    def breathe(self):  
        print("Breathing...")  
  
class Dog(Animal):  
    def bark(self):  
        print("Woof!")  
  
class Cat(Animal):  
    def meow(self):  
        print("Meow!")  
  
class Bird(Animal):  
    def fly(self):  
        print("Flying!")  
  
d = Dog(); c = Cat()  
  
d.speak()    → Dog barks  
c.speak()    → Cat meows
```

5) Hybrid Inheritance – Multiple + Single combined*

Hybrid inheritance म्हणजे Multiple Inheritance आणि Single Inheritance यांचं combination. Multiple inheritance वापरून अनेक base classes च्या properties एका derived class मध्ये येतात, आणि single inheritance वापरून त्या derived class च्या properties पुढे एका sub-derived class मध्ये जातात.

```
class A:

    def method_a(self):

        print("Class A")

class B(A):

    def method_b(self):

        print("Class B")

class C(A):

    def method_c(self):

        print("Class C")

class D(B, C):      # Multiple + Hierarchical combined

    def method_d(self):

        print("Class D")

obj = D()

obj.method_a()     → Class A
```

obj.method_b() → Class B

obj.method_c() → Class C

obj.method_d() → Class D

💡 Hybrid inheritance complex projects मध्ये वापरतात जिथे एका class ला अनेक sources कडून properties हव्या असतात!

Type	Structure
Single	$A \rightarrow B$
Multilevel	$A \rightarrow B \rightarrow C$
Multiple	$A + B \rightarrow C$
Hierarchical	$A \rightarrow B, A \rightarrow C, A \rightarrow D$
Hybrid	$A + B \rightarrow C \rightarrow D$ (combination)

Method Overriding

Method Overriding म्हणजे parent class च्या method ला child class मध्ये redefine करणं – same नाव, वेगळं behavior!

```
class Animal:
    def speak(self):
        print("Animal makes a sound")

class Dog(Animal):
```

```
def speak(self):  
    print("Dog says: Woof!")  
  
class Cat(Animal):  
    def speak(self):  
        print("Cat says: Meow!")  
  
class Duck(Animal):  
    def speak(self):  
        print("Duck says: Quack!")  
  
animals = [Dog(), Cat(), Duck(), Animal()]  
  
for animal in animals:  
    animal.speak()  
  
→ Dog says: Woof!  
→ Cat says: Meow!  
→ Duck says: Quack!  
→ Animal makes a sound
```

⚠ Method name exactly same असावं लागतं – तेव्हाच override होतो!

super() Function

super() म्हणजे parent class ला refer करण्याचा एक simple way – parent class चा `__init__` किंवा method call करायला वापरतात.

```
class Person:

    def __init__(self, name, age):

        self.name = name

        self.age = age

        print(f"Person created: {name}")

class Student(Person):

    def __init__(self, name, age, marks):

        super().__init__(name, age)

        self.marks = marks

        print(f"Student marks: {marks}")
```

```
s = Student("Rohit", 18, 95)
```

```
→ Person created: Rohit
```

```
→ Student marks: 95
```

```
print(s.name)    → Rohit
```

```
print(s.age)     → 18
```

```
print(s.marks)   → 95
```

super() – Overridden Method call करणे:

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

class Dog(Animal):
    def speak(self):
        super().speak()
        print("Dog says: Woof!")

d = Dog()
d.speak()
→ Animal sound
→ Dog says: Woof!
```

💡 **super()** वापरल्यामुळे parent चा code reuse होतो – duplicate code लिहायला नको!

Polymorphism

Polymorphism म्हणजे "many forms" – same method name, वेगळ्या objects वर वेगळं behavior.

```
class Shape:

    def area(self):

        pass

class Circle(Shape):

    def __init__(self, radius):

        self.radius = radius

    def area(self):

        return 3.14 * self.radius ** 2

class Rectangle(Shape):

    def __init__(self, length, width):

        self.length = length

        self.width = width

    def area(self):

        return self.length * self.width

class Triangle(Shape):

    def __init__(self, base, height):

        self.base = base

        self.height = height
```

```
def area(self):  
    return 0.5 * self.base * self.height  
  
shapes = [Circle(5), Rectangle(4, 6), Triangle(3, 8)]  
  
for shape in shapes:  
    print(f"Area: {shape.area()}")  
  
→ Area: 78.5  
→ Area: 24  
→ Area: 12.0
```

Built-in Polymorphism:

```
print(5 + 3)          → 8  
print("Hello" + "!") → Hello!  
  
print(len("Rohit"))   → 5  
print(len([1, 2, 3, 4])) → 4  
print(len({"a": 1, "b": 2})) → 2
```

💡 Polymorphism चा सर्वात मोठा फायदा – एकाच loop मध्ये different objects वर same method call करता येतो!

Summary

Concept	एका ओळीत
Class	Blueprint/Template
Object	Class चा instance
__init__	Object बनताना automatically run होतो
self	Current object चा reference
Instance Variable	प्रत्येक object साठी वेगळं
Class Variable	सगळ्या objects साठी common
Instance Method	Object वर काम करणारे functions
Encapsulation	Data wrap करणं, direct access बंद
Inheritance	Parent कडून properties घेणं
Method Overriding	Parent चा method child मध्ये redefine
super()	Parent class access करणं
Polymorphism	Same method, वेगळं behavior

💡 OOP हे real world problems solve करायचा सर्वात powerful मार्ग आहे!

Chapter 16 – Python Coding Style & Best Practices

Introduction to Coding Style

Coding Style म्हणजे काय?

Coding style म्हणजे code लिहिण्याची पद्धत – variable names कसे द्यायचे, spaces कसे द्यायचे, comments कसे लिहायचे. Code फक्त computer साठी नाही, माणसांसाठी पण असतो!

Clean Code का महत्वाचा आहे?

Bad code काम करतो, पण good code समजतो. 6 महिन्यांनी स्वतःचाच code वाचताना "हे मी लिहिलं होतं का?" असं वाटायला नको!

💡 Code लिहिताना नेहमी विचार करा – "हे दुसऱ्याला वाचता येईल का?"

Naming Conventions

Variable Names – snake_case

✔ बरोबर:

```
student_name = "Rohit"
```

```
total_marks = 95
```

```
is_present = True
```

✗ चुकीचं:

```
StudentName = "Rohit"
```

```
totalmarks = 95
```

```
x = 95
```

Function Names – snake_case

```
# ✔ बरोबर:  
  
def calculate_area():  
  
def get_student_marks():  
  
def is_valid_age():  
  
# ✗ चुकीचं:  
  
def CalculateArea():  
  
def ca():
```

Class Names – PascalCase

```
# ✔ बरोबर:  
  
class StudentDetails:  
  
class BankAccount:  
  
class CarEngine:  
  
# ✗ चुकीचं:  
  
class student_details:  
  
class bankaccount:
```

Constants – UPPER_SNAKE_CASE

✓ बरोबर:

MAX_SIZE = 100

PI = 3.14159

DATABASE_URL = "localhost"

✗ चुकीचं:

max_size = 100

maxsize = 100

Type	Convention	Example
Variable	snake_case	student_name
Function	snake_case	get_marks()
Class	PascalCase	StudentDetails
Constant	UPPER_SNAKE_CASE	MAX_SIZE

Indentation

Python मध्ये Indentation का महत्वाचं आहे?

Python मध्ये indentation फक्त style नाही – हे syntax आहे!
Indentation चुकली तर IndentationError येतो.

Tabs vs Spaces

```
# ✔ नेहमी Spaces वापरा (4 spaces):  
  
def greet():  
    print("Hello")  
  
    if True:  
        print("Yes")  
  
# ✗ Tabs आणि Spaces mix करू नका!
```

Spaces Rule:

```
# ✔ बरोबर:  
  
for i in range(5):  
    print(i)  
  
# ✗ चुकीचं:  
  
for i in range(5):  
    print(i)
```

⚠ VS Code मध्ये Tab key दाबल्यावर automatically 4 spaces येतात –
हे check करा!

Comments

Single-line Comments (#)

```
# हे single-line comment आहे

name = "Rohit"    # variable मध्ये name store केलं

# ✔ बरोबर:

# User चं वय validate करणे

if age >= 18:

    print("Adult")

# ✗ चुकीचं:

x = x + 1    # x ला 1 ने वाढवतो
```

Multi-line Comments

```
"""

हे multi-line comment आहे.

अनेक ओळी लिहिता येतात.

"""

"""

हे पण multi-line comment आहे.

"""
```

Docstrings:

```
def calculate_area(length, width):  
    """  
    Rectangle चं area calculate करतो.  
    Parameters: length (int/float), width (int/float)  
    Returns: area (float)  
    """  
    return length * width  
  
print(calculate_area.__doc__)
```

	Comments	Docstrings
Symbol	#	""" "" किंवा """ """
Python	Ignore करतो	Store करतो
वापर	Code explain	Function/Class explain

Code Readability

Meaningful Variable Names

✗ चुकीचं:

```
d = 86400
```

```
for i in range(n):
```

```
    s += i
```

✔ बरोबर:

```
seconds_per_day = 86400
```

```
for student in students:
```

```
    total_marks += student
```

Proper Spacing

✗ चुकीचं:

```
x=10
```

```
y=20
```

```
result=x+y
```

✔ बरोबर:

```
x = 10
```

```
y = 20
```

```
result = x + y
```

Small Functions:

✗ चुकीचं:

```
def process():
```

```
    # सगळं एकाच function मध्ये – खूप लांब!
```

```
    pass
```

✔ बरोबर:

```
def get_user_input():
```

```
    pass
```

```
def validate_input(data):
```

```
    pass
```

```
def calculate(data):
```

```
    pass
```

Unnecessary Code टाळणे:

✗ चुकीचं:

```
def is_even(n):  
    if n % 2 == 0:  
        return True  
    else:  
        return False
```

✔ बरोबर:

```
def is_even(n):  
    return n % 2 == 0
```

PEP 8 Introduction

PEP 8 म्हणजे काय?

PEP 8 म्हणजे Python Enhancement Proposal 8. हे Python साठी code लिहिण्याचे official style guidelines आहेत.

Beginner साठी Important PEP 8 Rules:

1. Indentation – 4 spaces

```
def function():
```

```
    pass
```

2. Line Length – maximum 79 characters

3. Blank Lines – functions/classes मध्ये 2 blank lines

```
class MyClass:
```

```
    pass
```

```
def my_function():
```

```
    pass
```

4. Imports – सगळ्यात वर, एक एक line वर:

```
import os
```

```
import sys
```

```
import math
```

✗ नाही:

```
import os, sys, math
```

5. Spaces around operators:

```
x = 10 + 5      # ✓
```

```
x = 10+5        # ✗
```

6. Comparison:

```
if x is None:    # ✓
```

```
if x == None:    # ✗
```

Common Beginner Mistakes

Mistake 1 – Poor Variable Names

```
# ✗ चुकीचं:  
a = 18  
b = "Rohit"  
c = 95.5  
  
# ✔ बरोबर:  
age = 18  
student_name = "Rohit"  
average_marks = 95.5
```

Mistake 2 – Wrong Indentation

```
# ✗ चुकीचं:  
def greet():  
print("Hello")  
  
# ✔ बरोबर:  
def greet():  
    print("Hello")
```

Mistake 3 – Infinite Loops

```
# ✗ चुकीचं:  
i = 0  
while i < 10:  
    print(i)  
    # i += 1 लिहायला विसरलो!  
  
# ✓ बरोबर:  
i = 0  
while i < 10:  
    print(i)  
    i += 1
```

Mistake 4 – Too Many Global Variables

```
# ✗ चुकीचं:  
name = "Rohit"  
age = 18  
def show():  
    print(name, age)  
  
# ✓ बरोबर:  
def show(name, age):  
    print(name, age)  
show("Rohit", 18)
```

Mistake 5 – Duplicate Code

```
# ✗ चुकीचं:  
print("Rohit".upper())  
print("Sakshi".upper())  
print("Raj".upper())  
  
# ✔ बरोबर:  
def print_upper(name):  
    print(name.upper())  
  
for name in ["Rohit", "Sakshi", "Raj"]:  
    print_upper(name)
```

Mistake 6 – Unused Variables

```
# ✗ चुकीचं:  
def calculate(a, b):  
    result = a + b  
    temp = a * b    # temp कुठेही use नाही!  
    return result  
  
# ✔ बरोबर:  
def calculate(a, b):  
    result = a + b  
    return result
```

Best Practices Summary

Best Practice	Benefit
Meaningful Names	Code समजायला सोपा
Comments	Code explain होतो
Indentation (4 spaces)	Errors टाळता येतात
Small Functions	Reusable code बनतो
Follow PEP 8	Professional coding style
Avoid Global Variables	Code predictable राहतो
No Duplicate Code	Maintenance सोपं होतं
Remove Unused Variables	Clean, efficient code

💡 **Final Thought** – Good code म्हणजे फक्त काम करणारा code नाही.
Good code म्हणजे दुसऱ्याला (किंवा 6 महिन्यांनी स्वतःला) सहज समजणारा code!

Chapter 17– Generators

1. Generators म्हणजे काय?

Generators म्हणजे Python मध्ये special type of functions जे iterable sequence of values create करतात. एक generator function एक generator object return करतो जो values one-by-one generate करतो.

साध्या भाषेत – जसं एक factory एकाच वेळी एकच product बनवते (सगळे products एकत्र बनवत नाही), तसंच generator एकाच वेळी एकच value produce करतो!

Regular Function vs Generator:

Regular Function	Generator
सगळ्या values एकत्र return करतो	एका वेळी एक value yield करतो
सगळ्या values memory मध्ये store होतात	Values on-the-fly generate होतात
return keyword वापरतो	yield keyword वापरतो

2. yield Keyword

yield म्हणजे generator चा सर्वात महत्वाचा भाग. return सारखंच काम करतो, पण function संपत नाही – पुढच्या next() call पर्यंत pause होतो!

```
def my_generator():  
    yield 1  
    yield 2  
  
gen = my_generator()
```

```
print(next(gen))    → 1
```

```
print(next(gen))    → 2
```

💡 return वापरल्यावर function पूर्णपणे संपतो. yield वापरल्यावर function pause होतो आणि पुढच्या next() ला resume होतो!

3. Generator बनवणे

```
def my_generator():  
    for i in range(5):  
        yield i  
  
gen = my_generator()  
print(next(gen))    → 0  
print(next(gen))    → 1  
print(next(gen))    → 2  
print(next(gen))    → 3  
print(next(gen))    → 4
```

for loop वापरून:

```
def my_generator():  
    for i in range(5):  
        yield i  
  
for value in my_generator():
```

```
print(value)
```

```
→ 0
```

```
→ 1
```

```
→ 2
```

```
→ 3
```

```
→ 4
```

4. next() Function

next() म्हणजे generator कडून पुढची value मागवणं. जेव्हा generator exhausted होतो (सगळ्या values संपतात) तेव्हा StopIteration error येतो.

```
def countdown():
```

```
    yield 3
```

```
    yield 2
```

```
    yield 1
```

```
    yield "Go!"
```

```
gen = countdown()
```

```
print(next(gen))    → 3
```

```
print(next(gen))    → 2
```

```
print(next(gen))    → 1
```

```
print(next(gen))    → Go!
```

```
print(next(gen))    → StopIteration Error!
```

StopIteration handle करने:

```
gen = countdown()

while True:

    try:

        value = next(gen)

        print(value)

    except StopIteration:

        print("Generator exhausted!")

        break
```

5. Practical Examples

Fibonacci Generator:

```
def fibonacci():

    a, b = 0, 1

    while True:

        yield a

        a, b = b, a + b

fib = fibonacci()

for _ in range(10):

    print(next(fib))

→ 0 1 1 2 3 5 8 13 21 34
```

Range Generator (Custom range):

```
def my_range(start, end, step=1):  
    current = start  
    while current < end:  
        yield current  
        current += step  
  
for num in my_range(0, 10, 2):  
    print(num)  
  
→ 0 2 4 6 8
```

File Reader Generator:

```
def read_large_file(file_path):  
    with open(file_path) as f:  
        for line in f:  
            yield line.strip()  
  
for line in read_large_file("data.txt"):  
    print(line)  
  
# प्रत्येक line एकाच वेळी read होतो – memory efficient!
```

💡 Generators large datasets, infinite sequences आणि memory-efficient programming साठी perfect आहेत!

6. Benefits of Generators

1. Memory Efficient:

```
# Regular List – संगठ्या values memory मध्ये:  
numbers_list = [x ** 2 for x in range(1000000)]    # 8 MB memory!  
  
# Generator – एकाच वेळी एक value:  
numbers_gen = (x ** 2 for x in range(1000000))    # जवळपास 0 MB!
```

2. Lazy Evaluation:

Generator values तेव्हाच calculate करतो जेव्हा actually हव्या असतात.
Unnecessary computation टाळतो.

3. Infinite Sequences:

```
def infinite_counter():  
    n = 0  
    while True:  
        yield n  
        n += 1  
  
counter = infinite_counter()  
print(next(counter))    → 0  
print(next(counter))    → 1  
  
# हे कधीच संपत नाही – पण memory वापरत नाही!
```

7. Generator Expression

List comprehension सारखंच, पण [] ऐवजी () वापरतात – generator बनतो.

List comprehension:

```
squares_list = [x ** 2 for x in range(10)]    # List – सगळं memory मध्ये
```

Generator expression:

```
squares_gen = (x ** 2 for x in range(10))    # Generator – lazy evaluation
```

```
for sq in squares_gen:
```

```
    print(sq)
```

→ 0 1 4 9 16 25 36 49 64 81

8. Summary

Concept	एका ओळीत
Generator	Special function जो values one-by-one yield करतो
yield	Function pause करतो, value return करतो
next()	Generator कडून पुढची value मागवणं
StopIteration	Generator exhausted झाल्यावर येतो
Lazy Evaluation	Values तेव्हाच calculate होतात जेव्हा हव्या
Memory Efficient	सगळ्या values एकत्र store होत नाहीत

Chapter 18– Walrus Operator (:=)

1. Walrus Operator म्हणजे काय?

Walrus Operator (:=) हा Python 3.8 मध्ये introduce झालेला assignment expression operator आहे. := हे symbol walrus (एक प्राणी) च्या डोळ्यांसारखं दिसतं – म्हणून हे नाव!

Walrus Operator एकाच वेळी दोन गोष्टी करतो:

1. Variable ला value assign करतो
2. त्याच value चा expression मध्ये वापर करतो

```
# Without Walrus:
```

```
n = 10
```

```
if n > 5:
```

```
    print(n)
```

```
# With Walrus:
```

```
if (n := 10) > 5:
```

```
    print(n)
```

```
→ 10
```

2.Traditional vs Walrus Operator

```
# Traditional way – 2 steps:
```

```
data = input("Enter something: ")
```

```
if len(data) > 5:
```

```
print(f"Long input: {data}")

# Walrus way – 1 step:

if len(data := input("Enter something: ")) > 5:

    print(f"Long input: {data}")
```

3. Walrus Operator – while Loop मध्ये

Walrus Operator while loop मध्ये खूप useful आहे – condition check करताना value assign करता येते.

```
# Traditional way:

user_input = input("Enter: ")

while user_input != "quit":

    print(f"You entered: {user_input}")

    user_input = input("Enter: ")

# Walrus way – cleaner!

while (user_input := input("Enter: ")) != "quit":

    print(f"You entered: {user_input}")
```

4. Walrus Operator – List मध्ये

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Traditional:

even = []
```

```
for n in numbers:

    result = n * 2

    if result > 10:

        even.append(result)

# Walrus way:

even = [result for n in numbers if (result := n * 2) > 10]

print(even)    → [12, 14, 16, 18, 20]
```

5. Practical Examples

Example 1 – File Reading:

```
# Traditional:

with open("data.txt") as f:

    line = f.readline()

    while line:

        print(line.strip())

        line = f.readline()

# Walrus way:

with open("data.txt") as f:

    while line := f.readline():

        print(line.strip())
```

Example 2 – Repeated Calculation:

```
import math

numbers = [4, 9, 16, 25, 36]

# Traditional – calculate twice:
results = [math.sqrt(n) for n in numbers if math.sqrt(n) > 3]

# Walrus – calculate once:
results = [root for n in numbers if (root := math.sqrt(n)) > 3]

print(results)    → [4.0, 5.0, 6.0]
```

6. कधी वापरावा, कधी नाही

✓ Walrus वापरावा:

Loop मध्ये repeated calculation टाळायला

while loop condition मध्ये assignment हवी असेल

Code shorter आणि efficient करायला

✗ Walrus वापरू नये:

Simple assignments साठी (x := 10 – unnecessary!), just use x= 10!)

Code कमी readable होत असेल तेव्हा

Beginners साठी confusing वाटत असेल तेव्हा

⚠ Walrus Operator Python 3.8+ मध्येच काम करतो. जुन्या Python versions मध्ये SyntaxError येईल!

Chapter 19– Time Module

1. Time Module म्हणजे काय?

Python चा time module time-related operations साठी functions provide करतो – timekeeping, formatting, आणि time conversions. हे module Python standard library मध्ये आहे – वेगळं install करायला नको!

```
import time
```

2. time.time()

time.time() म्हणजे 1 January 1970 पासून ते आतापर्यंत किती seconds गेले ते सांगतो (Epoch time).

```
import time

print(time.time())

→ 1736000000.123456    # आताचा exact time seconds मध्ये
```

Program Speed मोजणे:

```
import time

def using_for():
    for i in range(50000):
        print(i)

init = time.time()

using_for()

print(time.time() - init)    → किती seconds लागले ते दाखवतो
```

3. time.sleep()

time.sleep(seconds) म्हणजे program ला specified seconds साठी pause करणं.

```
import time

print("Hello")

time.sleep(3)          # 3 seconds थांब

print("How are you")

→ Hello
→ (3 seconds pause)
→ How are you
```

Countdown Timer:

```
import time

def countdown(seconds):

    for i in range(seconds, 0, -1):

        print(f"{i}...")

        time.sleep(1)

    print("Time's up!")

countdown(5)

→ 5... 4... 3... 2... 1... Time's up!
```

4. time.localtime()

time.localtime() મ્હળજે current local time struct_time format મધ્યે દેણં.

```
import time

t = time.localtime()

print(t)

→ time.struct_time(tm_year=2026, tm_mon=7, tm_mday=15,
                    tm_hour=10, tm_min=30, tm_sec=45, ...)
```

5. time.strftime()

time.strftime() મ્હળજે time લા human-readable string format મધ્યે convert કરણં.

```
import time

t = time.localtime()

formatted_time = time.strftime("%d-%m-%Y %H-%M-%S", t)

print(formatted_time)

→ 15-07-2026 10-30-45
```

Common Format Codes:

Code	મ્હળજે	Example
%Y	Year (4 digits)	2026
%m	Month (01-12)	07

Code	म्हणजे	Example
%d	Day (01-31)	15
%H	Hour (00-23)	10
%M	Minutes (00-59)	30
%S	Seconds (00-59)	45
%A	Weekday name	Wednesday
%B	Month name	July

Examples:

```
import time

t = time.localtime()

print(time.strftime("%d/%m/%Y", t))      → 15/07/2026

print(time.strftime("%A, %B %d, %Y", t)) → Wednesday, July 15,
2026

print(time.strftime("%I:%M %p", t))      → 10:30 AM
```

6. Practical Examples

Simple Timer:

```
import time

start = time.time()

for i in range(1000000):    # काहीतरी काम:
```

```

    pass

end = time.time()

print(f"Time taken: {end - start:.4f} seconds")

→ Time taken: 0.0234 seconds

```

Digital Clock:

```

import time

while True:

    current = time.strftime("%H:%M:%S")

    print(f"\rTime: {current}", end="")

    time.sleep(1)

```

7. Summary

Function	म्हणजे काय
time.time()	Epoch seconds (float)
time.sleep(n)	n seconds pause
time.localtime()	Local time struct
time.strftime(format, t)	Time ला string मध्ये format करणं

💡 time module performance testing, delays, आणि time formatting साठी खूप उपयुक्त आहे!

Chapter 20 – Requests Module & HTTP

1. Requests Module म्हणजे काय?

Python requests module म्हणजे एक HTTP library जी developers ला Python code वापरून HTTP requests send करायला enable करते. यामुळे APIs सोबत interact करणं आणि web services वापरणं खूप सोपं होतं.

Installation:

```
pip install requests
```

```
import requests
```

2. HTTP म्हणजे काय?

HTTP (HyperText Transfer Protocol) म्हणजे web वर data transfer करण्याचा protocol. Browser आणि server मध्ये communication HTTP वापरून होतं.

HTTP Methods:

Method	काय करतो
GET	Data fetch करणं (सर्वात common)
POST	नवीन data send करणं
PUT	Existing data update करणं
DELETE	Data delete करणं

3. GET Request

GET request म्हणजे server कडून data मागवणं.

```
import requests

response = requests.get("https://api.github.com")

print(response.status_code)    → 200

print(response.url)            → https://api.github.com

print(type(response))          → <class 'requests.models.Response'>
```

4. Status Codes

Status code म्हणजे server कडून येणारा response code – request यशस्वी झाली का ते सांगतो.

Code	म्हणजे
200	OK – यशस्वी
201	Created – नवीन resource बनलं
400	Bad Request – चुकीची request
401	Unauthorized – login हवं
403	Forbidden – access नाही
404	Not Found – page नाही
500	Internal Server Error

```

import requests

response = requests.get("https://api.github.com")

if response.status_code == 200:

    print("Success!")

elif response.status_code == 404:

    print("Not found!")

else:

    print(f"Error: {response.status_code}")

```

5. Response Content

```

import requests

response = requests.get("https://api.github.com")

# Text content:

print(response.text)          # String format मध्ये

# JSON content:

data = response.json()        # Dictionary format मध्ये

print(data)

# Bytes content:

print(response.content)       # Raw bytes

```

6. GET Request with Parameters

```
import requests

# URL parameters:

params = {

    "q": "Python",

    "sort": "stars",

    "order": "desc"

}

response = requests.get(

    "https://api.github.com/search/repositories",

    params=params

)

data = response.json()

print(f"Total results: {data['total_count']}")
```

7. POST Request

POST request म्हणजे server ला नवीन data send करणं.

```
import requests

# JSON data send करणं:

data = {
    "name": "Rohit",
    "age": 18,
    "language": "Python"
}

response = requests.post(
    "https://httpbin.org/post",
    json=data
)

print(response.status_code)    → 200

print(response.json())
```

8. Headers

```
import requests

headers = {
    "Authorization": "Bearer your_token_here",
    "Content-Type": "application/json",
    "User-Agent": "MyApp/1.0"
}

response = requests.get(
    "https://api.example.com/data",
    headers=headers
)

print(response.json())
```

9. Error Handling

```
import requests

try:
    response = requests.get("https://api.github.com", timeout=5)
    response.raise_for_status()    # 4xx, 5xx errors साठी exception
    raise
```

```
data = response.json()

print("Success:", data)

except requests.exceptions.ConnectionError:

    print("Internet connection नाही!")

except requests.exceptions.Timeout:

    print("Request timeout झाली!")

except requests.exceptions.HTTPError as e:

    print(f"HTTP Error: {e}")

except requests.exceptions.RequestException as e:

    print(f"Error: {e}")
```

Practical Example – Weather API

```
import requests

API_KEY = "your_api_key"

city = "Solapur"

url = "https://api.openweathermap.org/data/2.5/weather"

params = {
    "q": city,
    "appid": API_KEY,
    "units": "metric"
}

response = requests.get(url, params=params)

if response.status_code == 200:
    data = response.json()
    temp = data["main"]["temp"]
    description = data["weather"][0]["description"]
    print(f"City: {city}")
    print(f"Temperature: {temp}°C")
    print(f"Weather: {description}")
else:
    print(f"Error: {response.status_code}")
```

Summary

Concept	म्हणजे काय
requests.get()	Data fetch करणं
requests.post()	Data send करणं
response.status_code	Server response code
response.json()	JSON data dictionary मध्ये
response.text	Response string मध्ये
params=	URL parameters
headers=	Request headers
timeout=	Request timeout set करणं
raise_for_status()	HTTP errors साठी exception

💡 Requests module वापरून APIs, weather data, news, GitHub data – कुठलंही web data Python मध्ये easily fetch करता येतं!

Chapter 21 – Shutil Module

1. Shutil Module म्हणजे काय?

shutil (Shell Utilities) म्हणजे Python चा module जो files आणि directories वर high-level operations provide करतो. OS module पेक्षा जास्त convenient असतो – file copy, move, delete यांसाठी!

```
import shutil
```

⚠️ shutil Python standard library मध्ये आहे – install करायला नको!

2. shutil.copy() – File Copy करणे

```
import shutil
```

1) shutil.copy(src, dst) – file copy (shutil.copy(src, dst) – file copy (tmetadata नाही, permissions copied)):

```
shutil.copy("source.txt", "destination.txt")
```

2) shutil.copy2(src, dst) – file copy (shutil.copy2(src, dst) – file copy (timestamps सहित सर्व metadata copied)):

```
shutil.copy2("source.txt", "destination.txt")
```

Example:

```
import os
```

```
open("original.txt", "w").write("hello")
```

```
shutil.copy("original.txt", "backup.txt")
```

```
print(os.path.exists("backup.txt"))    → True
```

```
import shutil

import os

# File बनवणे:

with open("original.txt", "w") as f:

    f.write("Hello, Rohit!")
```

3. `shutil.move()` – File/Folder Move करणे

```
import shutil

# File move करणे:

shutil.move("source.txt", "folder/source.txt")

# File rename करणे (same folder):

shutil.move("old_name.txt", "new_name.txt")

# Folder move करणे:

shutil.move("old_folder", "new_location/old_folder")
```

4. `shutil.copytree()` – Entire Folder Copy करणे

```
import shutil

# Entire directory tree copy:

shutil.copytree("source_folder", "destination_folder")
```

Example:

```
import shutil

import os

# Folder structure बनवणे:

os.makedirs("project/src", exist_ok=True)

os.makedirs("project/tests", exist_ok=True)

with open("project/src/main.py", "w") as f:

    f.write("print('Hello')")

# Entire project copy:

shutil.copytree("project", "project_backup")

print("Backup created!")
```

5. shutil.rmtree() – Entire Folder Delete करणे

```
import shutil

# Entire directory tree delete (⚠ Permanent!):

shutil.rmtree("folder_to_delete")
```

Safe Delete with Check:

```
import shutil

import os

folder = "temp_folder"

if os.path.exists(folder):

    shutil.rmtree(folder)

    print(f'{folder} deleted!')

else:

    print(f'{folder} doesn't exist!')
```

6. Archive बनवणे आणि Extract करणे

```
import shutil

# Archive बनवणे (zip/tar):

shutil.make_archive("my_archive", "zip", "folder_to_compress")

# → my_archive.zip बनेल

shutil.make_archive("my_archive", "tar", "folder_to_compress")

# → my_archive.tar बनेल

# Archive extract करणे:

shutil.unpack_archive("my_archive.zip", "extracted_folder")
```

7. Practical Example – Backup System

```
import shutil

import os

import time

def create_backup(source_folder):

    timestamp = time.strftime("%Y%m%d_%H%M%S")

    backup_name = f"backup_{timestamp}"

    # Uncompressed backup folder delete (keep only the .zip):

    shutil.copytree(source_folder, backup_name)

    # Archive बनवणे:

    shutil.make_archive(backup_name, "zip", backup_name)

    # Original folder delete:

    shutil.rmtree(backup_name)

    print(f"Backup created: {backup_name}.zip")

create_backup("my_project")

→ Backup created: backup_20260715_103045.zip
```

8. Summary

Function	काय करतो
<code>shutil.copy(src, dst)</code>	File copy (metadata नाही)
<code>shutil.copy2(src, dst)</code>	File copy (metadata सहित)
<code>shutil.move(src, dst)</code>	File/Folder move किंवा rename
<code>shutil.copytree(src, dst)</code>	Entire directory copy
<code>shutil.rmtree(path)</code>	Entire directory delete
<code>shutil.make_archive(name, format, folder)</code>	Archive बनवणे
<code>shutil.unpack_archive(archive, folder)</code>	Archive extract करणे

💡 Shutil module backup systems, file organizers, आणि deployment scripts साठी खूप उपयुक्त आहे!

Chapter 22 – BeautifulSoup / bs4 (Web Scraping)

1. Web Scraping म्हणजे काय?

Web Scraping म्हणजे website वरून automatically data extract करणं. जसं तू manually website उघडतोस आणि information copy करतोस, तसंच Python आपोआप करतो – पण खूप जलद!

Real-world use cases:

- Price comparison websites
- News aggregators
- Job listings
- Research data collection

2. BeautifulSoup म्हणजे काय?

BeautifulSoup (bs4) म्हणजे HTML आणि XML parse करण्यासाठीचा Python library. Website चा HTML code घेतो आणि त्यातून specific data easily extract करायला मदत करतो.

Installation:

```
pip install beautifulsoup4
```

```
pip install requests
```

```
from bs4 import BeautifulSoup
```

```
import requests
```

3. Basic BeautifulSoup Setup

```
from bs4 import BeautifulSoup

html = """

<html>

    <head><title>My Page</title></head>

    <body>

        <h1>Hello, World!</h1>

        <p class="intro">This is Python.</p>

        <p class="content">BeautifulSoup is great!</p>

    </body>

</html>

"""

soup = BeautifulSoup(html, "html.parser")

print(soup.title.text)      → My Page
print(soup.h1.text)        → Hello, World!
```

4. Finding Elements

```
from bs4 import BeautifulSoup

html = """
```

```

<html>

    <body>

        <h1>Python Tutorial</h1>

        <p class="intro">Learn Python</p>

        <p class="content">Step by step</p>

        <a href="https://python.org">Python Website</a>

    </body>

</html>

"""

soup = BeautifulSoup(html, "html.parser")

# Single element:

print(soup.find("h1").text)           → Python Tutorial

print(soup.find("p").text)           → Learn Python (पहिला p)

# Class वापरून:

print(soup.find("p", class_="content").text) → Step by step

# All elements:

paragraphs = soup.find_all("p")

```

```
for p in paragraphs:
```

```
    print(p.text)
```

```
→ Learn Python
```

```
→ Step by step
```

```
# Attribute access:
```

```
link = soup.find("a")
```

```
print(link["href"]) → https://python.org
```

```
print(link.text) → Python Website
```

5.CSS Selectors

```
from bs4 import BeautifulSoup
```

```
html = """
```

```
<div id="main">
```

```
    <ul class="list">
```

```
        <li>Python</li>
```

```
        <li>JavaScript</li>
```

```
        <li>Java</li>
```

```
    </ul>
```

```
</div>
```

```
"""
```

```
soup = BeautifulSoup(html, "html.parser")
```

```
# ID वापरून:
```

```
main = soup.select("#main")
```

```
# Class वापरून:
```

```
items = soup.select(".list li")
```

```
for item in items:
```

```
    print(item.text)
```

```
→ Python
```

```
→ JavaScript
```

```
→ Java
```

6.Real Website Scraping

```
import requests
```

```
from bs4 import BeautifulSoup
```

```
# Website fetch करणे:
```

```
url = "https://books.toscrape.com"
```

```
response = requests.get(url)
```

```
# Parse करणे:
```

```
soup = BeautifulSoup(response.content, "html.parser")
```

```
# सगळ्या books चे titles:
```

```
books = soup.find_all("h3")

for book in books[:5]:    # पहिले 5

    print(book.find("a")["title"])
```

7.Scraping with Pagination

```
import requests

from bs4 import BeautifulSoup

base_url = "https://books.toscrape.com/catalogue/page-{}.html"

all_books = []

for page in range(1, 4):    # पहिले 3 pages

    url = base_url.format(page)

    response = requests.get(url)

    soup = BeautifulSoup(response.content, "html.parser")

    books = soup.find_all("article", class_="product_pod")

    for book in books:

        title = book.h3.a["title"]

        price = book.find("p", class_="price_color").text

        all_books.append({"title": title, "price": price})

    print(f"Page {page} done!")

print(f"Total books: {len(all_books)}")
```

8. Important Notes

⚠ Web Scraping करताना हे लक्षात ठेवा:

- Website चे robots.txt check करा
- Terms of Service वाचा
- Server वर जास्त load टाकू नका
- `time.sleep()` वापरून requests मध्ये delay द्या
- Public APIs available असतील तर scraping ऐवजी API वापरा

9. Summary

Function	काय करतो
<code>BeautifulSoup(html, parser)</code>	HTML parse करणं
<code>soup.find(tag)</code>	पहिला matching element
<code>soup.find_all(tag)</code>	सगळे matching elements
<code>soup.select(css_selector)</code>	CSS selector वापरून search
<code>element.text</code>	Element चा text content
<code>element["attribute"]</code>	Element चा attribute value

💡 BeautifulSoup + Requests = Python Web Scraping चं सर्वात powerful combination!

Chapter 23 – Operator Overloading

1. Operator Overloading म्हणजे काय?

Operator Overloading म्हणजे Python मध्ये mathematical (+, -, *, /) आणि comparison (>, <, ==) operators चं behavior custom data types साठी redefine करणं. म्हणजे तुमच्या स्वतःच्या classes मध्ये standard operators वापरता येतात!

Built-in types मध्ये + already overloaded आहे:

print(5 + 3) → 8 # Addition

print("Hello" + "!") → Hello! # Concatenation

print([1,2] + [3,4]) → [1,2,3,4] # List joining

2. Magic Methods (Dunder Methods)

Operator overloading साठी special methods वापरतात ज्यांना **Magic Methods** किंवा **Dunder Methods** (double underscore) म्हणतात.

Operator	Method	म्हणजे
+	__add__	Addition
-	__sub__	Subtraction
*	__mul__	Multiplication
/	__truediv__	Division
==	__eq__	Equal

Operator	Method	મ્હણજે
!=	__ne__	Not Equal
>	__gt__	Greater Than
<	__lt__	Less Than
>=	__ge__	Greater or Equal
<=	__le__	Less or Equal
str()	__str__	String representation
len()	__len__	Length

3. add – + Operator Overload

```

class Point:

    def __init__(self, x, y):

        self.x = x

        self.y = y

    def __add__(self, other):

        return Point(self.x + other.x, self.y + other.y)

    def __str__(self):

        return f"Point({self.x}, {self.y})"

p1 = Point(1, 2)

```

```
p2 = Point(3, 4)

p3 = p1 + p2    # __add__ call होतो
print(p3)       → Point(4, 6)
```

4. Comparison Operators Overload

```
class Student:

    def __init__(self, name, marks):

        self.name = name

        self.marks = marks

    def __eq__(self, other):

        return self.marks == other.marks

    def __gt__(self, other):

        return self.marks > other.marks

    def __lt__(self, other):

        return self.marks < other.marks

    def __str__(self):

        return f"{self.name}: {self.marks}"

s1 = Student("Rohit", 95)

s2 = Student("Sakshi", 88)
```

```
s3 = Student("Raj", 95)

print(s1 > s2)    → True
print(s1 < s2)    → False
print(s1 == s3)   → True
print(s1 == s2)   → False
```

5. Practical Example – Vector Class

```
class Vector:

    def __init__(self, x, y, z):

        self.x = x

        self.y = y

        self.z = z

    def __add__(self, other):

        return Vector(self.x + other.x,

                        self.y + other.y,

                        self.z + other.z)

    def __sub__(self, other):

        return Vector(self.x - other.x,

                        self.y - other.y,
```

```

        self.z - other.z)

def __mul__(self, scalar):
    return Vector(self.x * scalar,
                  self.y * scalar,
                  self.z * scalar)

def __str__(self):
    return f"Vector({self.x}, {self.y}, {self.z})"

def __len__(self):
    return 3

v1 = Vector(1, 2, 3)
v2 = Vector(4, 5, 6)

print(v1 + v2)    → Vector(5, 7, 9)
print(v1 - v2)    → Vector(-3, -3, -3)
print(v1 * 3)     → Vector(3, 6, 9)
print(len(v1))    → 3

```

6. str आणि repr

```
class Book:

    def __init__(self, title, author, pages):

        self.title = title

        self.author = author

        self.pages = pages
```

```
    def __str__(self):

        # User-friendly representation

        return f"{self.title}" by {self.author}'

    def __repr__(self):

        # Developer-friendly representation

        return f'Book("{self.title}", "{self.author}", {self.pages})'

    def __len__(self):

        return self.pages
```

```
book = Book("Python Fundamentals", "Rohit Survase", 200)

print(book)           → "Python Fundamentals" by Rohit Survase

print(repr(book))     → Book("Python Fundamentals", "Rohit
Survase", 200)

print(len(book))      → 200
```

7. Summary

Magic Method	Operator	Example
<code>__add__</code>	<code>+</code>	<code>obj1 + obj2</code>
<code>__sub__</code>	<code>-</code>	<code>obj1 - obj2</code>
<code>__mul__</code>	<code>*</code>	<code>obj1 * 3</code>
<code>__eq__</code>	<code>==</code>	<code>obj1 == obj2</code>
<code>__gt__</code>	<code>></code>	<code>obj1 > obj2</code>
<code>__str__</code>	<code>str()</code>	<code>print(obj)</code>
<code>__len__</code>	<code>len()</code>	<code>len(obj)</code>

💡 Operator Overloading वापरल्यामुळे तुमच्या custom classes standard Python syntax वापरून काम करतात – code intuitive आणि readable होतो!

Chapter 24 – PyPDF

1. PyPDF म्हणजे काय?

PyPDF म्हणजे Python मधला library जो PDF files वर operations करायला वापरतात – PDF read करणे, text extract करणे, pages merge/split करणे, rotate करणे, आणि बरंच काही.

Installation:

```
pip install pypdf
```

```
from pypdf import PdfReader, PdfWriter
```

2. PDF Read करणे

```
from pypdf import PdfReader

# PDF उघडणे:

reader = PdfReader("document.pdf")

# Pages count:

print(f"Total pages: {len(reader.pages)}")

# Specific page वाचणे:

page = reader.pages[0]    # पहिली page (index 0)

text = page.extract_text()

print(text)
```

💡 PyPDF वापरून automated PDF processing, report generation, आणि document management systems बनवता येतात!

3. सगळ्या Pages मधून Text Extract करणे

```
from pypdf import PdfReader

reader = PdfReader("document.pdf")

full_text = ""

for page in reader.pages:
    full_text += page.extract_text()

print(full_text)

# File मध्ये save करणे:

with open("extracted_text.txt", "w") as f:
    f.write(full_text)
```

4. PDF Merge करणे

```
from pypdf import PdfReader, PdfWriter

writer = PdfWriter()

# Multiple PDFs add करणे:

pdf_files = ["file1.pdf", "file2.pdf", "file3.pdf"]

for pdf_file in pdf_files:
```

```

reader = PdfReader(pdf_file)

for page in reader.pages:

    writer.add_page(page)

# Merged PDF save करणे:

with open("merged.pdf", "wb") as output:

    writer.write(output)

print("PDFs merged successfully!")

```

5. PDF Split करणे

```

from pypdf import PdfReader, PdfWriter

reader = PdfReader("large_document.pdf")

# प्रत्येक page वेगळ्या PDF मध्ये save करणे:

for i, page in enumerate(reader.pages):

    writer = PdfWriter()

    writer.add_page(page)

    with open(f"page_{i+1}.pdf", "wb") as output:

        writer.write(output)

print(f"Split into {len(reader.pages)} files!")

```

6. Specific Pages Extract करणे

```
from pypdf import PdfReader, PdfWriter

reader = PdfReader("document.pdf")

writer = PdfWriter()

# Pages 2 ते 5 extract करणे (index 1 to 4):

for page_num in range(1, 5):

    writer.add_page(reader.pages[page_num])

with open("extracted_pages.pdf", "wb") as output:

    writer.write(output)
```

7.PDF Rotate करणे

```
from pypdf import PdfReader, PdfWriter

reader = PdfReader("document.pdf")

writer = PdfWriter()

for page in reader.pages:

    page.rotate(90)    # 90 degrees rotate

    writer.add_page(page)

with open("rotated.pdf", "wb") as output:

    writer.write(output)
```

8. PDF Metadata

```
from pypdf import PdfReader

reader = PdfReader("document.pdf")

metadata = reader.metadata

print(f"Title: {metadata.title}")

print(f"Author: {metadata.author}")

print(f"Pages: {len(reader.pages)}")
```

9. Practical Example – PDF Report Generator

```
from pypdf import PdfWriter, PdfReader
import os

def merge_reports(folder_path, output_file):
    writer = PdfWriter()

    pdf_files = sorted([f for f in os.listdir(folder_path)
                        if f.endswith(".pdf")])

    for pdf_file in pdf_files:
        reader = PdfReader(os.path.join(folder_path, pdf_file))

        for page in reader.pages:
            writer.add_page(page)
```

```

print(f"Added: {pdf_file}")

with open(output_file, "wb") as output:

    writer.write(output)

print(f"Merged PDF saved: {output_file}")

print(f"Total files merged: {len(pdf_files)}")

merge_reports("reports_folder", "annual_report.pdf")

```

10. Summary

Function	काय करतो
PdfReader(file)	PDF उघडणे
reader.pages	सगळ्या pages
page.extract_text()	Page मधून text काढणे
PdfWriter()	नवीन PDF बनवणे
writer.add_page(page)	Page add करणे
writer.write(file)	PDF save करणे
page.rotate(degrees)	page.rotate(degrees) (90,180,270)
reader.metadata	reader.metadata

Conclusion

This book marks the completion of my very first programming language — Python. What started as a simple decision to learn programming slowly turned into 200 pages of notes, and eventually, into this book. एक छोटासा learning decision कसा या पूर्ण journey मध्ये बदलला, याचा हा एक भाग आहे.

Every concept written here was first understood, then practiced, and then explained in my own words. This book does not claim to cover the entire Python ecosystem — it focuses on the core fundamentals that form the foundation of any programmer's journey.

This is not just a collection of notes; it is a record of where my programming journey began. माझ्यासाठी हा book माझ्या learning journey ची सुरुवात आणि मी आतापर्यंत केलेल्या efforts चा एक record आहे. As I continue learning and exploring more advanced topics, I hope to build further volumes on this same foundation.

To anyone reading this — may this book serve as a reminder that every expert was once a beginner, and every great journey starts with the first line of code. Keep learning, keep practicing, and most importantly, keep building.

About the Author

Rohit Survase is a self-taught programmer who began his coding journey with Python. This book is based on his personal notes, written while learning Python as his first programming language. Python शिकताना तयार केलेल्या या notes ना एक structured form देऊन हे book तयार झाले आहे.

Python being his first programming language, this book represents not just technical learning, but the beginning of a much larger journey into the world of software development. हा journey पुढे अनेक नवीन concepts, technologies आणि projects explore करण्याचा आहे.

Through consistent practice and curiosity, Rohit continues to explore new concepts in programming, with the goal of building a strong foundation for advanced topics in the future. Learning करताना concepts समजून घेणे, practice करणे आणि continuously improve होणे यावर तो भर देतो.

This book is the first step of that journey — and the first of many more to come.

References & Future Learning Resources

This book was created using personal notes prepared while following the CodeWithHarry 100-video Python programming playlist on YouTube. The explanations, examples, and structure in this book reflect a personal understanding of these concepts, written in my own words.

For readers who wish to continue their Python journey beyond this book, here are some recommended directions:

Official Python Documentation – docs.python.org

Practice Platforms – **HackerRank, LeetCode, Codewars**

Project-Based Learning – Building small real-world projects is the best way to strengthen concepts

Advanced Topics – Object-Oriented Design, Web Development, Automation, Data Structures & Algorithms, and eventually Data Science or AI

Learning never truly ends in programming. This book is just the foundation — the next steps are exploration, practice, and building real projects.

About This Book

*A personal collection of Python notes from my first programming journey.
Covers core Python fundamentals for beginners.*

What You'll Learn

- Variables, Data Types & Operators
- Strings, Lists, Tuples, Dictionaries & Sets
- Conditional Statements, Loops, Functions & Recursion
- Object-Oriented Programming, File & Exception Handling

About the Author

*Rohit Survase is a self-taught programmer who wrote this book from his own
Python notes*

