PYTHON KNIGHTS

The most Powerful python book

AUTHOR :
Needi Developer

Introduction

Python Knight is not just a book; it’s your ultimate guide to mastering Python, crafted with care and precision by **Needi Developer**, a passionate educator and developer with a vision to simplify programming for everyone. Whether you're taking your first steps in Python or looking to sharpen your existing skills, this book equips you with the tools to become a true knight in the world of coding.

What Makes Python Knight Unique?

- Step-by-Step Journey: This book guides you from Python's fundamentals to advanced concepts with real-world examples and practical exercises.
- Knight's Quest for Excellence: Like a knight embarks on an adventurous quest, you'll explore topics ranging from basic syntax to object-oriented programming, data structures, and more.
- Real-Life Applications: Beyond theory, the book dives into real-life coding challenges, empowering you to solve problems like a pro.
- Interactive Learning: With quizzes, coding challenges, and projects, Python Knight ensures you actively engage with every topic.

About the Author: Needi Developer

Needi Developer is a trailblazer in the tech world, blending a passion for teaching with hands-on experience in software development. With a mission to make programming accessible to everyone, Needi has dedicated their efforts to creating intuitive courses and resources that cater to learners from all walks of life. Their motto? “Code to create, learn to lead.”

Understanding the unique challenges faced by learners in Asia, especially those for whom English is not a primary language, Needi has gone the extra mile to ensure inclusivity. Python Knight is available in two distinct versions:

- Hinglish/Roman Urdu Version – Tailored for learners in South Asia, this version helps those who may struggle with traditional English resources to easily grasp programming concepts.
- English Version – A universal edition designed for readers worldwide.

By offering **Python Knight** for free, **Needi Developer** aims to break barriers to education, ensuring that everyone—regardless of background—has the opportunity to learn and grow in the ever-evolving world of technology. This initiative is a testament to Needi's unwavering commitment to empowering individuals to unlock their potential and build a brighter future.

Join the journey and discover how coding can transform lives, one learner at a time.

CONTENTS

6 ------ Lesson 1: Basics of Python
6 ------ What is Python?
9 ------ Installing Python and VS Code
14 ----- Writing and Running Python Scripts.
17 ----- Python Syntax and Comments.
21 ----- Exercise: Short Quiz.

23 ----- Lesson 2: Variables and Data Types
23 ----- Variables.
24 ----- Constants.
24 ----- Keywords.
26 ----- Data Types.
29 ----- Type Conversion.
32 ----- Input and Output.
35 ----- Exercise: Simple Calculator.

36 ----- Lesson 3: Strings
36 ----- Strings.
40 ----- String Methods.
44 ----- Escape Sequence.
47 ----- f-Strings (Formatted Strings).
49 ----- Docstrings (Documentation Strings).
51 ----- Exercise: String Manipulation Program.

52 ----- Lesson 4: Control Flow
52 ----- if, elif, else statements.
56 ----- Loops.
60 ----- Break, Continue and Pass.
64 ----- Shorthand if-Else.
49 ----- Docstrings (Documentation Strings).
66 ----- Exercise: ATM Simulation.

67 ----- Lesson 5: Functions
67 ----- Defining Functions and Return.
70 ----- Functions Arguments.
73 ----- Exercise: Prime Number Check.

74 ----- Lesson 6: Data Structures
74 ----- List.
77 ----- Tuples.
80 ----- Sets.
85 ----- Dictionaries.
88 ----- Enumerates and Zip Functions.
91 ----- Exercise: Data Structures Puzzle.

92 ----- Lesson 7: Error Handling
92 ----- Introduction to Errors and Exceptions.
94 ----- How to Handle Exceptions?
97 ----- Raising Custom Errors
101 ---- Debugging Basics.

105 ----- Lesson 8: File Handling
105 ----- Reading and Writing Files.
108 ----- Important Methods in File Handling.
111 ----- Understanding seek() and tell() Functions in File Handling.
114 ---- Exercise: File-Based-To-Do List Program.

115 ----- Lesson 9: Object-Oriented Programming
115 ----- Classes and Objects.
118 ----- Constructors and Instance Methods.
122 ----- Inheritance.
128 ---- Method Overriding.
132 ---- Class Methods vs Static Methods.
135 ---- Magic (Dunder) Methods.
138 ---- Operator Overloading.
142 ---- Exercise: Build a Simple Banking System.

143 ----- Lesson 10: Python Modules
143 ----- Modules.
146 ----- if __name__ == "__main__".
148 ----- Installing Third-Party Libraries with pip.
151 ---- Virtual Environments (venv).
154 ---- Exercise: File Management Script.

155 ----- Lesson 11: Advanced Python Concepts
155 ----- Introduction to Generators and Iterators.
158 ----- Map, Filter, and Reduce Function.
161 ----- Lambda Functions.
164 ---- Walrus Operator (:=).
167 ---- Mega Knight Projects.

171 ----- Advanced Tips For Becoming a Python Knight

Lesson 1: Basics of Python

What Is Python

Python ek high-level programming language hai jo aaj duniya ki sabse popular aur powerful languages me se ek hai. Ye language 1991 me Guido van Rossum ne design ki thi. Python ka naam ek comedy show "**Monty Python's Flying Circus**" se inspired hai, iska snakes se koi lena dena nahi hai! 😄

Python Ki Khasiyat

- Simple aur Easy-to-Learn: Python ki syntax (likhne ka tareeqa) bilkul simple aur human-readable hai. Agar aapko basic English samajh aati hai, to aap Python seekhna shuru kar sakte hain.

Example:

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

Is code se screen par "**Hello World!**" print hoga.

- Multipurpose Language: Python ko har tarah ke kaam ke liye use kiya ja sakta hai:
  - Web Development (e.g., Django, Flask frameworks)
  - Data Science aur Machine Learning
  - Game Development
  - Mobile Apps aur Desktop Apps
  - Web Scraping (data collect karna websites se)
  - Automation (repetitive tasks automate karna)
- Free Aur Open Source: Python ek free aur open-source language hai, matlab aap ise free use kar sakte hain aur modify bhi kar sakte hain apne use ke hisaab se.
- Cross-Platform: Python har operating system (Windows, Mac, Linux) pe chalti hai. Aap ek jagah code likhein aur wo har jagah kaam karega.

- Large Community: Python ki ek badi aur active community hai. Matlab agar aapko kahin problem aaye, to bohot saare log aapki madad karne ke liye ready hote hain.

Python Kyu Seekhein?

- Beginner-Friendly: Agar aap programming me naye hain, to Python seekhna sabse best hai. Ye easy aur logical hai.
- Job Opportunities: Python developers ki demand har field me barh rahi hai, aur iski salaries bhi achhi hoti hain.
- Automation: Aap boring aur repetitive kaam automate kar sakte hain Python se.
- Projects Banane Ke Liye Best: Chhoti apps se lekar bade AI projects tak, Python har tarah ke projects ke liye perfect hai.

Python Ki Features

- Dynamic Typing: Variables ko declare karne ki zarurat nahi hoti. **Example:**

```
x = 10 # Python automatically samajhta hai k x ek integer hai.
```

- **Libraries Ka Treasure**: Python ke paas 140,000+ libraries hain jo alag-alag kaam ke liye use hoti hain, jaise:
  - **NumPy** aur **Pandas**: Data analysis ke liye.
  - **Matplotlib** aur **Seaborn**: Data visualization ke liye.
  - **TensorFlow** aur P**yTorch**: Machine Learning ke liye.
- **Integration**: Python doosri languages (C, C++, Java) aur tools ke saath easily integrate ho jata hai.

Python Ki Features

Python ka future bohot bright hai! AI, Data Science, aur Web Development jese fields me iska istemal barhta ja raha hai. Beginners aur professionals dono ke liye ye ek ideal language hai.

Ek Simple Python Example

Aayiye ek chhota sa program likhte hain jo numbers ka sum calculate kare:

```
# Es program ko dekh kar ghabraiye mat ye just ek example hai
# Do numbers ka sum calculate karne ka program
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

sum = num1 + num2

print("Sum of", num1, "and", num2, "is", sum)
# Es program ko aagy mazeed tafseel say samjhay gay
```

Conclusion

Python ek all-rounder language hai jo easy aur powerful hai. Agar aapko programming seekhni hai ya career banana hai, to Python ek perfect start hai. Ab der kis baat ki? Aaj hi Python seekhna shuru karein aur naye opportunities explore karein! 🚀

The full lesson is now available on the Needi Developer YouTube channel.

Installing Python and VS Code

Python aur VS Code install karna programming ki journey ka pehla aur important step hai. Aayiye step-by-step process ko samajhte hain:

Step 1: Python Install Karna

Python ko install karna bohat simple hai es mai koi rocket science nahi hai, bas neeche waly steps ko follow karein:

1. Python Website Visit Karein

- Python ki official website open karein: https://www.python.org.

2. Download Python

- Website par "Download Python" ka button dikhai dega, jo aapke system ke hisaab se version suggest karega (Windows, Mac, ya Linux).
- Tip: Hamesha latest stable version download karein.

3. Installer Run Karein

- Python installer file ko open karein.

Windows me es tarah ki file hogi us ko run kary

4. Add Python to PATH (Important)

- Installer shuru karte hi ek checkbox dikhai dega: "Add Python to PATH".

Is box ko zaroor check karein, warna baad me manual configuration karna padega.

5. Install Karna

- Customize Installation ya Install Now ka option dikhai dega. Beginners ke liye "Install Now" par click karna best hai.
- Install hone ke baad "Close" par click karein.

Step 2: Python Installation Verify Karna

Python install hone ke baad check karein ke sab kuch theek se kaam kar raha hai ya nahi.

1. Command Prompt Open Karein (Windows)

- Windows Key + R press karein, aur “cmd” type karein.

2. Python Version Check Karein

- Command prompt me type karein:

```
python --version
```

- Agar output me Python ka version (Python 3.12.6) dikhai de ya es say zyada, to iska matlab installation sahi hai.

3. Python REPL Test Karein

- Command prompt me type karein:

```
python
```

- Aapko Python ka interactive shell dikhai dega, jaha aap code likh sakte hain.

Step 3: VS Code Install Karna

VS Code ek powerful aur lightweight code editor hai jo Python ke liye perfect hai. Installation process neeche diya gaya hai:

1. VS Code Website Visit Karein

- Official website open karein: https://code.visualstudio.com.

2. Download VS Code

- "Download for Windows/Mac/Linux" ka button dikhai dega. Apne operating system ke hisaab se installer download karein.

3. Installer Run Karein

- Download hone ke baad installer file ko open karein.

4. Installation Settings Configure Karein

- Terms and Conditions accept karein.
- Recommended:
    - "Add to PATH" option check karein.
    - "Create Desktop Shortcut" bhi select karein.
    - Behtar ye hai kay sary Checkboxes hi click kr dein

5. Install Karein

- Install button par click karein aur complete hone ke baad "Finish" karein.

Step 4: Python Environment VS Code Me Set Karna

Python aur VS Code ko connect karna important hai. Follow these steps:

1. Open VS Code

- Desktop shortcut ya start menu se VS Code open karein.

2. Python Extension Install Karein

- Extensions Icon (left side bar me box icon) par click karein.
- Search Bar me Python type karein.
- Official Microsoft Python extension ko install karein.

3. Python Interpreter Select Karein

- Bottom-right corner me "Select Interpreter" ka option dikhai dega.
- Apne installed Python version ko select karein (Python 3.12) ya es say above jo apny latest install kiya.

4. Test Python Script in VS Code

- Ek new file create karein aur extension .py save karein, jaise:

main.py

- Isme likhein:

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

- File ko run karne ke liye Run Button ya Ctrl + F5 press karein.

Common Issues and Solutions

1. Python Not Recognized Error

- Aapne **"Add Python to PATH"** option miss kar diya hoga.
- **Solution**: PATH environment variable manually configure karein ya Python reinstall karein.

2. VS Code Me Python Run Nahi Ho Raha

- Check karein ke Python interpreter sahi select hua hai ya nahi.
- Solution: Bottom bar me interpreter select karein.

Conclusion

Ab aapke system me Python aur VS Code install hai, aur aap ready hain apna pehla program likhne ke liye!
Yahi se aapki Python journey shuru hoti hai. Practice karte rahiye aur naye concepts explore karein.

The full lesson is now available on the Needi Developer YouTube channel.

Writing and Running Python Scripts.

Python ki scripts likhne aur run karne ka process bohot simple aur user-friendly hai. Aayiye step-by-step esko samajhte hain:

Step 1: Python Script Kya Hai?

Python script ek text file hoti hai jisme Python code likha jata hai. Is file ka extension .py hota hai, jaise:

main.py

Python script ko aap multiple ways se likh aur run kar sakte hain.

Step 2: Python Script Likhnay Ke Tools

Python script likhne ke liye aapko ek code editor ya IDE chahiye.

Recommended Tools:

1. **VS Code** (Visual Studio Code): Lightweight aur beginner-friendly.
2. **PyCharm**: Thora heavy hai low-end pc ky liye recommend nahi hai or use karna thora sa complex hai.
3. **IDLE**: Python ke saath default aata hai.
4. **Notepad++**: Simple text editor.
5. **Jupyter Notebook**: Data science aur experimentation ke liye best.

Step 3: Python Script Likhnay Ka Tareeqa

1. VS Code Me Script Likhein

- File Create Karein:
  - VS Code open karein aur ek nayi file banayein.
  - File ko **.py** extension ke saath save karein. Example:

main.py

- Code Likhein:
- File me apna Python code likhein.

Example:

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

2. IDLE Me Script Likhein

- IDLE open karein aur File → New File select karein.
- Code likhne ke baad file ko .py extension ke saath save karein.

Step 4: Python Script Ko Run Karna

Python script ko run karne ke liye aapke system me Python installed hona chahiye.
Agar apny uper waly steps follow kiye hai to apkay computer me python install ho gaya hai.

1. Command Prompt (Windows) Ya Terminal (Mac/Linux) Se Script Run Karna

- Apni **.py** file ka path find karein.
  - Example: Agar file C:\PythonScripts\main.py me hai, to path yahi hoga.

- Terminal/Command Prompt open karein.
- Ye command type karein

```
python C:\PythonScripts\main.py
```

Agar output me **Hello World!** print ho, to aapka code sahi run ho gaya.

2. VS Code Me Script Run Karna

- Apni Python file open karein.
- Top bar me "Run" button dikhai dega, uspe click karein ya Ctrl + F5 press karein.
- Output VS Code ke terminal me dikhai dega.

3. IDLE Me Script Run Karna

- IDLE me apni .py file open karein.
- Press F5 ya Run → Run Module select karein.
- Output IDLE shell me dikhai dega.

Conclusion

Python scripts likhna aur run karna ek beginner-friendly process hai.

The full lesson is now available on the Needi Developer YouTube channel.

Python Syntax and Comments

1. Python Syntax

Python me syntax ek set of rules hota hai jo ye define karta hai ki program ka code kaise likhna hai aur kaise execute hoga. Agar syntax ka dhyan nahi diya gaya, to Python interpreter error throw karega.

Python ka syntax simple aur beginner-friendly hai. Iski readability isko dusri languages se alag banati hai. Yahaan kuch basic syntax rules hain:

1. Indentation:

Python indentation (spaces) ka use karta hai block of code ko define karne ke liye. Ye dusri languages jese **{}** ya **;** ke jagah use hota hai.

Example:

```
if True:
   print("This is indented!") #proper indentation
```

Second line me print sy pehlay jo space hai use indent kehtay hai.

2. Case Sensitivity:

Python mein variables aur keywords case-sensitive hote hain.

Example:

```
my_var = 5
My_var = 10
print(my_var) #Output: 5
print(My_var) #Output: 10
```

3. Statements:

Har statement ek alag line me likha jata hai. Agar ek hi line me multiple statements likhne hon, to **;** (semicolon) ka use hota hai.

Example:

```
print("Python");print("Knight") #multiple statements in one line
```

4. Keywords:

Python me kuch reserved words hain jinko as a variable ya function name use nahi kar sakte.

Example: **if**, **else**, **for**.

2. Python Comments

Comments ko code me documentation ke liye use kiya jata hai. Ye code ka wo part hote hain jo execute nahi hota. Comments ka purpose code ko readable aur understandable banana hai.

Agar ap apni help kay liye koi note wagera kisi block-of-code par lgana chahty hai to comments ki help say lga saktay hai

Types of Comments:

1. Single-Line and inline Comments:

symbol se start hote hain.

Example:

```
# This is a single line comment
print("Hello, Comments!") # This is an inline comment
```

2. Multi-Line Comments:

Triple quotes (''' or """) ka use karke multi-line comments likhe ja sakte hain.

Example:

```
"""
This is a multi-line comment. It spans multiple lines.
"""
print("Multi-line, Comments!")
```

Why Use Comments?

1. Code Explanation:

Complex code ko explain karne ke liye.

Example:

```
# Check is a number is even
number = 10
if number % 2 == 0:
  print("Number is even")
```

2. Code Debugging:

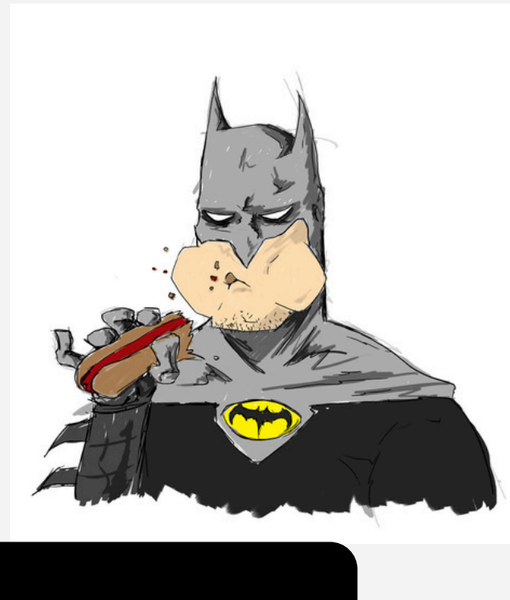
Specific lines ko temporarily disable karne ke liye.

Example:

```
# print("This won't execute")
print("This will execute")
```

3. Documentation:

Code ka purpose aur logic batane ke liye helpful.



Chaliye kuch funny example dekhtay hai:

```
# This program checks if Batman is hungry or not
is_hungry = True
if is_hungry:
  print("Batman need snacks")
else:
  print("Batman is full!")
```

Conclusion:

Python syntax aur comments code ko clean aur understandable banate hain. Indentation ka dhyan rakhein, aur comments ka use karke apna code aur readable banayen.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

1. Multiple Choice

Python me comment likhne ka sahi tareeqa kya hai?

a) // Ye ek comment hai

b) # Ye ek comment hai

c) /* Ye ek comment hai */

d) Inme se koi nahi

2. True or False

Python case-sensitive hota hai, matlab **Var** aur **var** ko ek hi variable maana jata hai. ☐

Python me blocks of code define karne ke liye indentation zaroori hai. ☐

3. Fill in the Blank

Python me comments likhne ke liye _____ symbol ka use hota hai.

4. Identify the Error

Niche diye gaye code me kya galat hai?

```
if 10 > 5:
print("10 is greater than 5")
```

Answers :

1. **(b)**
2. **(False) (True)**
3. **(#)**
4. **print statement indented nahi hai.**

Lesson complete—great job!

Lesson 2: Variables and Data Types

Variables

Data Containers

Variables Python me aise containers hote hain jisme hum data store karte hain, just like a jar, jis me hum cheezein store karte hain. Har jar ka apna naam hota hai (**variable name**), aur isme hum alag alag type ka data rakh sakte hain.

Example:

Sochiye aap ek shop me kaam kar rahe hain aur aapko alag alag cheezon ke price save karne hain. Har cheez ke price ko ek variable me store karte hain:

```
apple_price = 100
banana_price = 50
mango_price = 170

print("Apple price is",apple_price)
print("Banana price is",banana_price)
print("Mango price is",mango_price)
```

Output:

```
Apple price is 100
Banana price is 50
Mango price is 170
```

Rules for Naming Variables:

- Naam letter (a-z, A-Z) ya underscore (_) se shuru hona chahiye.
- Naam me numbers aa sakte hain, lekin pehle position pe nahi jaise: (**price1** valid hai, **1price** invalid hai).
- Variables case-sensitive hote hain Jaise: (**age** aur **Age** alag hain).

Constants

Fixed Values

Constants wo hoti hain jo kabhi change nahi hoti, jaise hamari daily life me ek kilogram = 1000 grams hota hai. Python me constants ko **ALL_CAPS** me likha jata hai.

Example:

Sochiye, aap ek calculator bana rahe hain aur **PI** ka value fix rakhna chahte hain:

```
PI = 3.14159 # Fixed Value of PI
GRAVITY = 9.8 # Acceleration due to gravity

circle_radius = 5
area = PI * (circle_radius ** 2)

print("Area of the circle is: ", area)
```

Although Python me aap constants ko change kar sakte hain, lekin aapko achi coding practice ke liye unhe badalne nahi chahiye.

Keywords

Reserved Words

Keywords Python ke **special words** hote hain jo programming ke rules define karte hain. Inka use variable names ke liye nahi kar sakte.

Example:

Jaise traffic signals me **STOP** ka matlab hamesha rukna hota hai, waise hi Python ke keywords ka specific meaning hota hai.

Some Common Keywords:

if, **else**, **for**, **while**, **def**, **return**, **import**, etc.

Keywords in Action:

```
if 10 > 5: # if is a keyword
  print("10 is greater than 5")
else: # else is also a keyword
  print("5 is greater than 10")
```

Conclusion

Variables, constants, aur keywords Python ke foundation hain. Variables data ko handle karne me madad karte hain, constants un values ko fix rakhte hain jo change nahi hoti, aur keywords programming ka basic syntax banate hain.

The full lesson is now available on the Needi Developer YouTube channel.

Data Types

Python me har data ka ek type hota hai jo define karta hai ke wo data kya hai aur uske sath kya operations kiye ja sakte hain. Data types ki madad se aap Python ko batate ho ke kisi value ko kaise handle karna hai.

1. Integer (int)

Definition:

Integer wo numbers hote hain jo bina decimal ke hote hain, jaise **1**, **-5**, **100**, etc.

Example:

```
age = 20
apples = -10
distance = 300

print("Age:",age)
print("Apples:",apples)
print("Distance:",distance)
```

2. Float (Decimal Numbers)

Definition:

Float wo numbers hote hain jo decimal ke sath hote hain, jaise **3.14**, **-2.5**, **0.0**, etc.

Example:

```
PI = 3.14
temperature = 6
height = 6

print("PI:", PI)
print("Temperature:", temperature)
print("Height:", height)
```

3. String (str)

Definition:

String wo data type hai jo text ya characters store karta hai. Strings hamesha quotation marks (' ' ya " ") me likhe jate hain.

Example:

```
name = "Ali"
greeting = 'Hello, World!'
city = "Karachi"

print("Name:", name)
print("Greeting:", greeting)
print("City:", city)
```

4. Boolean (bool)

Definition:

Boolean sirf do values store karta hai: **True** ya **False**.

Example:

```
is_raining = True
has_license = False

print("Is it raining?", is_raining)
print("Has driving license?", has_license)
```

4. Boolean (bool)

Definition:

None ka matlab hai **kuch bhi nahi**. Jab aapko ek variable banana ho lekin abhi koi value assign nahi karni ho, to aap **None** use karte hain.

Example:

```
future_plan = None

print("Future Plan:", future_plan)
```

Quick Comparison of Data Types

Data Type	Example	Use Case
int	10, -5, 100	Age, quantity, counters.
float	3.14, -2.5	Temperature, height, area.
str	Hello	Names, addresses, messages.
bool	True, False	Status checks, conditions.
None	None	Placeholder, empty values.

Conclusion

- int: For whole numbers.
- float: For decimal numbers.
- str: For text data.
- bool: For true/false logic.
- None: For "no value yet."

The full lesson is now available on the Needi Developer YouTube channel.

Type Conversion

Type conversion ka matlab hai ek data type ko doosre data type me badalna. Python me aap easily ek type se doosre type me convert kar sakte ho, is process ko type casting bhi kehte hain.

Type conversion do tarah ka hota hai:

1. **Implicit Type Conversion**
   Python khud se (automatically) ek data type ko doosre me convert kar deta hai.
2. **Explicit Type Conversion**
   Programmer manually ek data type ko doosre me badalta hai.

1. Implicit Type Conversion

Yeh tab hota hai jab Python khud data type ko convert karta hai bina kisi error ke.
Python yeh sirf un situations me karta hai jaha koi data loss na ho.

Example:

```
# Implicit Conversion
num_int = 5        # Integer
num_float = 2.5    # Float
result = num_int + num_float

print("Result:", result)        # 7.5
print("Type of result:", type(result))  # Float
```

Explanation:

- **5** ek integer hai, aur **2.5** ek float hai.
- Jab in dono ko add kiya gaya, Python ne integer **5** ko automatically float me convert kar diya **5.0** taake calculation smooth ho.

2. Explicit Type Conversion

Is process me programmer khud se type conversion karta hai using built-in functions.

Common Type Conversion Functions:

Function	Use	Example
int()	Kisi value ko integer me badalna	int(3.5) → 3
float()	Kisi value ko float me badalna	float(5) → 5.0
str()	Kisi value ko string me badalna	str(5) → "5"
bool()	Kisi value ko boolean me badalna	bool(1) → True

Converting to Integer

- Float ya string ko integer me badalne ke liye **int()** function use hota hai.
- Decimal wali value truncate (remove) ho jati hai.

Example:

```
num_float = 4.7
num_str = "10"

# Convert to int
int_from_float = int(num_float)  # 4
int_from_str = int(num_str)      # 10

print(int_from_float, int_from_str)
```

Converting to Integer

- Kisi bhi value ko string me convert karne ke liye **str()** use hota hai.

Example:

```
num = 10
pi = 3.14
# Convert to string
str_num = str(num)   # "10"
str_pi = str(pi)     # "3.14"
print("String:", str_num + " is an integer")
print("String:", str_pi + " is a float")
```

Converting to Boolean

- Kisi value ko boolean me badalne ke liye **bool()** use hota hai.
- Python me, kuch values hamesha False hoti hain:
  - **0**
  - **None**
  - Empty values: **""**, **[]**, **{}**, **()**
- Baaki sab **True** hoti hain.

Example:

```
val1 = 0
val2 = 10
val3 = " "

# Convert to boolean
bool1 = bool(val1)  # False
bool2 = bool(val2)  # True
bool3 = bool(val3)  # False

print(bool1, bool2, bool3)
```

Conclusion

- Implicit conversion automatic hoti hai jab data loss ka koi risk na ho.
- Explicit conversion me aapko khud se data type convert karna padta hai.
- Python ke built-in functions jaise **int()**, **float()**, **str()**, aur **bool()** is process ko simple banate hain.

Type conversion har programming task me zaroori hoti hai, especially jab user input ya data processing ke waqt data type consistent rakhna ho.

The full lesson is now available on the Needi Developer YouTube channel.

Input and Output

Python me input aur output handle karne ke liye built-in functions hote hain.

- **Input**: User se data lena.
- **Output**: Screen par data dikhana.

Output (print())

print() function ka use output ko screen par display karne ke liye hota hai.

Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- ***objects**: Multiple items ko output kar sakte hain.
- **sep**: Objects ke beech separator, default **' '**.
- **end**: Line end hone ke baad kya likhna hai, default **'\n'**.
- **file**: Jaha output chahiye (default screen).
- **flush**: Output ko turant dikhane ke liye.

Example: Basic Usage

```
# Simple print statement
print("Hello, Python!")  # Output: Hello, Python!

# Printing multiple items
name = "Ali"
age = 20
print("Name:", name, "Age:", age)  # Output: Name: Ali Age: 20

# Using custom separators and endings
print("Python", "is", "fun", sep="-", end="!")  # Output: Python-is-fun!
```

Input (input())

input() function ka use user se data lene ke liye hota hai. Jo bhi user type karta hai, wo string format me return hota hai.

Syntax:

```
input(prompt)
```

prompt: User ko message ya instruction dene ke liye.

Example: Basic Input

```
# Asking the user's name
name = input("Enter your name: ")

print("Hello,", name + "!")
```

Converting Input

Input hamesha string hota hai, lekin agar aapko number ya kisi aur type ka data chahiye ho, to aapko explicit type conversion karna padega.

Example: Integer Input

```
# Taking two numbers and adding them
num1 = int(input("Enter first number: "))  # Convert input to integer
num2 = int(input("Enter second number: "))  # Convert input to integer

result = num1 + num2
print("The sum is:", result)
```

Example: Float Input

```
height = float(input("Enter your height in meters: "))

print("Your height is:", height, "meters")
```

Conclusion

- **print()** is used for displaying output, and you can style it using f-strings or separators.
- **input()** allows you to get user data, but remember to convert it to the desired type for calculations.

Input and output are essential tools for creating interactive programs.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

Create a Python program that acts as a simple calculator.
The program should:

- Take two numbers from the user.
- Perform the following operations using variables: addition, subtraction, multiplication, and division.
- Display the results for each operation.

Lesson complete—great job!

Lesson 3: Strings

Strings

Strings Python me text ko represent karne ke liye use hote hain. Strings ek sequence of characters ka collection hote hain jo quotes (single ya double) ke andar likhe jate hain.

What is a String?

- A string ek sequence hoti hai, jo letters, digits, aur special characters ka collection ho sakti hai.
- Strings ko **single quotes** ('), **double quotes** (""), ya **triple quotes** (''' or """) ke andar likha jata hai.

Example:

```
# Single and double quotes
name = 'Ali'
greeting = "Hello, World!"

# Triple quotes for multiline strings
multiline_string = '''This is a
multiline string!'''
```

Creating Strings

1. Single Quotes:

```
single_quote_string = 'Hello'
```

2. Double Quotes:

```
double_quote_string = "World"
```

3. Triple Quotes:

```
triple_quote_string = '''Python
Strings
Are Cool!'''
```

1. String Indexing

Indexing ka use karke aap string ke kisi bhi individual character ko access kar sakte hain.
Python me indexing 0 se start hoti hai.

Syntax:

```
string_name[index]
```

Examples:

```
text = "Python"
print(text[0])  # Output: P (first character)
print(text[5])  # Output: n (last character)
```

Negative Indexing:

Negative indexing string ke end se access karne ke liye hoti hai.

```
text = "Python"
print(text[-1])  # Output: n (last character)
print(text[-6])  # Output: P (First character)
```

2. String Slicing

Slicing ka use string ke ek portion ya substring ko access karne ke liye hota hai. Slicing ka syntax start index, end index, aur step ko define karta hai.

Syntax:

```
string_name[start : end : step]
```

- **start**: Index jahan se slicing shuru hogi (included).
- **end**: Index jahan tak slicing chalegi (excluded).
- **step**: Har step me kitne characters skip karne hain (optional).

Examples:

```
text = "PythonProgramming"

# Basic slicing
print(text[0:6])  # Output: Python (characters 0 to 5)
print(text[6:])   # Output: Programming (from index 6 to end)

# Negative slicing
print(text[-11:-1])  # Output: Programmin

# Using step
print(text[0:12:2])  # Output: Pto rg (alternate characters)
```

Shortcut for Slicing:

- **string[:]**: Puri string return karega.
- **string[:end]**: Start se end tak substring.
- **string[start:]**: Start se end tak.
- **string[::-1]**: String ko reverse karega.

3. String Operations

Python me strings ke upar bahut saare useful operations perform kar sakte hain.

Concatenation (Join Strings):

Strings ko **+** operator ka use karke joda ja sakta hai.

```
first_name = "Needi"
last_name = "Developer"
full_name = first_name + " " + last_name
print(full_name)  # Output: Needi Developer
```

Repetition:

Strings ko * operator ka use karke repeat karna.

```
greet = "Hi! "
print(greet * 3)  # Output: Hi! Hi! Hi!
```

Membership Testing (**in** / **not in**):

Check karega ki string me koi substring hai ya nahi.

```
text = "Hello World"
print("Hello" in text)     # Output: True
print("Python" not in text)  # Output: True
```

Summary:

- Indexing: Access individual characters of a string.
- Slicing: Extract portions of strings.
- Operations: Combine, repeat, or check substrings.

The full lesson is now available on the Needi Developer YouTube channel.

String Methods

String methods Python me predefined functions hote hain jo strings ke saath operations ko asaan banate hain. Inka use karna bohot easy hota hai, aur ye commonly text processing ke liye kaam aate hain.

Why String Methods Are Useful?

- Strings ke operations ko asaan aur efficient banate hain.
- Real-world scenarios me, jaise user input ko clean karna, email validation, aur text formatting ke liye kaam aate hain.

Commonly Used String Methods

Here's a list of frequently used string methods with examples:

1. **lower()** and **upper()**

- **lower()**: String ke saare characters ko lowercase me convert karta hai.
- **upper()**: String ke saare characters ko uppercase me convert karta hai.

Example:

```
text = "Hello World"
print(text.lower())  # Output: hello world
print(text.upper())  # Output: HELLO WORLD
```

2. **strip()**, **lstrip()**, and **rstrip()**

- **strip()**: String ke start aur end ke extra spaces ko remove karta hai.
- **lstrip()**: Sirf left side se spaces remove karta hai.
- **rstrip()**: Sirf right side se spaces remove karta hai.

Example:

```
text = "   Python   "
print(text.strip())   # Output: "Python"
print(text.lstrip())  # Output: "Python   "
print(text.rstrip())  # Output: "   Python"
```

3. **replace()**

Ek string ke andar kisi specific character ya substring ko replace karne ke liye use hota hai.

Example:

```
text = "I love Java"
print(text.replace("Java", "Python"))  # Output: I love Python
```

4. **find()** and **index()**

- **find()**: Substring ka index return karta hai (agar substring exist kare). Agar substring na mile to -1 return karta hai.
- **index()**: Similar to **find()**, lekin agar substring na mile to error throw karta hai.

Example:

```
text = "Python Programming"
print(text.find("Prog"))   # Output: 7
print(text.find("Java"))   # Output: -1
print(text.index("Prog"))  # Output: 7
```

5. **startswith()** and **endswith()**

- **startswith()**: Check karta hai agar string kisi specific substring se shuru hoti hai.
- **endswith()**: Check karta hai agar string kisi specific substring par end hoti hai.

Example:

```
text = "Python is fun"
print(text.startswith("Python"))  # Output: True
print(text.endswith("fun"))       # Output: True
```

6. **split()** and **join()**

- **split()**: String ko ek list me convert karta hai, ek specific delimiter ke basis par.
- **join()**: List ke elements ko ek string me combine karta hai, ek delimiter ke saath.

Example:

```
text = "apple,banana,orange"
fruits = text.split(",")  # Split by comma
print(fruits)             # Output: ['apple', 'banana', 'orange']

joined_text = " - ".join(fruits)
print(joined_text)        # Output: apple - banana - orange
```

7. **count()**

String me kisi specific character ya substring ki occurrence count karta hai.

Example:

```
text = "banana"
print(text.count("a"))  # Output: 3
```

8. **capitalize()**, **title()**, and **swapcase()**

- **capitalize()**: Sirf pehle character ko uppercase banata hai.
- **title()**: Har word ka first character uppercase banata hai.
- **swapcase()**: Uppercase ko lowercase aur lowercase ko uppercase me convert karta hai.

Example:

```
text = "python programming"
print(text.capitalize())  # Output: Python programming
print(text.title())       # Output: Python Programming
print(text.swapcase())    # Output: PYTHON PROGRAMMING
```

9. **isalpha()**, **isdigit()**, and **isalnum()**

- **isalpha()**: Check karta hai agar string sirf letters contain karti hai.
- **isdigit()**: Check karta hai agar string sirf numbers contain karti hai.
- **isalnum()**: Check karta hai agar string sirf letters aur numbers contain karti hai.

Example:

```
text = "Python123"
print(text.isalpha())   # Output: False
print("123".isdigit())  # Output: True
print(text.isalnum())   # Output: True
```

Summary:

String methods Python me kaafi powerful hain, aur unka use karna simple hai. Ye methods har jagah useful hote hain, chahe wo user input sanitize karna ho, text formatting ho, ya strings me searching aur replacing karna ho.

The full lesson is now available on the Needi Developer YouTube channel.

Escape Sequences

Escape sequences Python me special characters represent karte hain jo directly print nahi kiye ja sakte. Inka use karna bohot zaroori hota hai jab aapko strings ke andar specific formatting ya special effects chahiye hote hain.

What Are Escape Sequences?

- Escape sequences strings ke andar backslash (\) ke saath likhe jaate hain.
- Ye Python ko batate hain ke agle character ko special treat karna hai.

Why Use Escape Sequences?

- Strings me special characters (newline, tab, quotes) ko include karne ke liye.
- Format aur align karne ke liye.
- Specific symbols ya actions perform karne ke liye.

Common Escape Sequences

Escape Sequence	Description	Example Output
\n	Newline (next line)	Moves text to a new line
\t	Horizontal Tab	Adds a tab space
\'	Single Quote	Prints ' inside quotes
\"	Double Quote	Prints " inside quotes
\\	Backslash	Prints \
\r	Carriage Return	Moves to the start of line
\b	Backspace	Removes one character

Examples of Escape Sequences

1. Newline (`\n`)

Ek new line create karta hai:

Example:

```
print("Hello\nWorld")
# Output:
# Hello
# World
```

2. Tab (`\t`)

Ek horizontal tab space add karta hai:

Example:

```
print("Name:\tNeedi")
# Output:
# Name:   Needi
```

3. Single Quote (`\'`) and Double Quote (`\"`)

Single ya double quotes ko string ke andar include karne ke liye:

Example:

```
print('It\'s a sunny day')  # Output: It's a sunny day
print("She said, \"Hello!\"")  # Output: She said, "Hello!"
```

4. Backslash (`\\`)

String me ek backslash print karne ke liye:

Example:

```
print("This is a backslash: \\")
# Output: This is a backslash: \
```

5. Carriage Return (**\r**)

Carriage return text ko line ke start par le jata hai:

Example:

```
print("Hello\rWorld")
# Output: World
```

6. Backspace (**\b**)

Ek character ko erase karne ke liye:

Example:

```
print("Helloo\b!")
# Output: Hello!
```

Summary:

Escape sequences are essential for controlling how text is displayed in strings. Whether it's formatting output, adding special symbols, or managing alignment, they make text handling more powerful and flexible.

The full lesson is now available on the Needi Developer YouTube channel.

f-Strings (Formatted Strings)

Python me **f-strings** string formatting ke liye use hoti hain. Ye tool programming ko asaan aur zyada readable banate hain.

What are f-Strings?

f-Strings (formatted strings) Python 3.6 me introduce hui thi. Inka use strings me variables aur expressions directly add karne ke liye hota hai.

Why use f-Strings?

- **Simple Syntax**: **{}** ka use karke string me variables ya expressions insert karte hain.
- **Readable Code**: Zyada clean aur understandable hota hai.
- **Fast Performance**: Purani methods (like **%** aur **.format()**) se zyada efficient hoti hai.

How to use f-Strings?

String se pehle **f** ya **F** lagayein aur curly braces **{}** ke andar variable ya expression likhein.

Examples of f-Strings

1. Variables Add Karna

```
name = "Naveed"
age = 20
print(f"My name is {name} and I am {age} years old.")
# Output: My name is Naveed and I am 20 years old.
```

2. Expressions Use Karna

```
x = 8
y = 4
print(f"The sum of {x} and {y} is {x + y}.")
# Output: The sum of 8 and 4 is 12.
```

3. Number Formatting

```
price = 1234.567
print(f"Total price is {price:.2f} rupees.")
# Output: Total price is 1234.57 rupees.
```

4. Function Ke Saath Use Karna

```
def square(num):
    return num * num

print(f"The square of 5 is {square(5)}.")
# Output: The square of 5 is 25.
```

The full lesson is now available on the Needi Developer YouTube channel.

Docstrings (Documentation Strings)

Docstrings code ko document karne ke liye use hoti hain. Ye tool programming ko asaan aur zyada readable banate hain.

What are docstrings?

Docstrings Python ka ek feature hai jo functions, classes, aur modules ko describe karne ke liye use hota hai. Ye triple quotes (""" """ ya ''' ''') me likhe jaate hain.

Why use docstrings?

- **Code ko Samajhne Me Asaani**: Dusre developers ya khud ke liye.
- **Documentation Tools Me Useful**: Docstrings automatically documentation banate hain (using **help()** function).

How to write docstrings?

1. Function Ke Liye Docstring

```
def greet(name):
    """
    This function greets the person whose name is passed as an argument.
    """
    return f"Hello, {name}!"

print(greet("Naveed"))
# Output: Hello, Naveed!
```

Docstring Explain Kar Raha Hai:

- Function kya karta hai.
- Argument (input) kya hona chahiye.

2. Class Ke Liye Docstring

```
class Calculator:
    """
    A simple calculator class to perform basic operations.
    """

    def add(self, x, y):
        """
        Returns the sum of x and y.
        """
        return x + y
```

3. Docstring Access Karna

Python ka **help()** function docstrings ko dekhne ke liye use hota hai:

```
print(help(greet))
```

Summary :

- **f-Strings**: Strings me variables aur expressions ko asaani se include karte hain.
- **Docstrings**: Code ko describe karte hain aur uski documentation banate hain.

In dono tools ka use karne se code readable, clean, aur efficient banta hai!

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

Task: String Manipulation Program

Write a Python program that takes a string input from the user. Perform the following tasks:

- Convert the string to uppercase.
- Find the length of the string.
- Replace all spaces in the string with underscores (_).

Lesson complete—great job!

Lesson 4: Control Flow

if, elif, else Statements

Python me if, elif, else statements decision-making ke liye use hote hain. Ye program ko allow karte hain ki woh alag-alag conditions ke basis par alag-alag actions le.

Understanding If, Elif, Else

1. if Statement:

If ka use ek condition check karne ke liye hota hai. Agar condition true ho, to code execute hota hai.

```
if condition:
    # code to execute
```

2. elif Statement (Else If):

Jab pehli if condition false ho, to elif ka use karte hain dusri condition check karne ke liye.

```
elif another_condition:
    # code to execute
```

3. else Statement:

Jab if aur elif dono false ho, tab else ka code execute hota hai.

```
else :
    # code to execute
```

Syntax of if, elif, else

```
if condition:
    # Block of code if condition is true
elif another_condition:
    # Block of code if elif condition is true
else:
    # Block of code if all above conditions are false
```

Examples of if, elif, else

1. Basic Example:

```
age = 18

if age < 18:
    print("You are a minor.")
elif age == 18:
    print("Congratulations! You are an adult now.")
else:
    print("You are an adult.")
```

Output:

```
Congratulations! You are an adult now.
```

2. Multiple Example:

```
marks = 85

if marks >= 90:
    print("Grade: A+")
elif marks >= 80:
    print("Grade: A")
elif marks >= 70:
    print("Grade: B")
else:
    print("Grade: C")
```

Output:

```
Grade: A
```

How It Works: **Key Points**

- Python conditions use comparison operators like:**<**, **>**, **<=**, **>=**, **==**, **!=**
- Code blocks under **if**, **elif**, **else** statements must be indented (usually 4 spaces).

Comparison Operators name:

- **==**: Equal to
- **!=**: Not equal to
- **>**: Greater then
- **<**: Less then
- **>=**: Greater then or equal to
- **<=**: Less then equal to

Examples for practice:

1. Weather Decision:

Aik aesa program jo ye btata hai k weather kesa hai:

```
temperature = 25

if temperature > 30:
    print("It's hot outside. Drink plenty of water!")
elif temperature >= 20:
    print("The weather is pleasant.")
else:
    print("It's cold outside. Wear warm clothes!")
```

Output:

```
The weather is pleasant.
```

2. ATM Withdrawal:

Aik aesa program jo ye btata hai k es transaction ky bad user ka balance ktna reh jai ga:

```
balance = 5000
withdrawal = 3000

if withdrawal > balance:
    print("Insufficient balance.")
elif withdrawal == balance:
    print("Your account will be empty after this transaction.")
else:
    print(f"Transaction successful! Remaining balance: {balance - withdrawal}")
```

Output:

```
Transaction successful! Remaining balance: 2000
```

3. Traffic Light:

Aik aesa program jo ye btata hai k kis light ky on hony par kya krna hai:

```
light = "Red"

if light == "Red":
    print("Stop! Wait for the green light.")
elif light == "Yellow":
    print("Get ready to go.")
else:
    print("Go! Drive safely.")
```

Output:

```
Stop! Wait for the green light.
```

Summary:

- **if**: Checks the first condition.
- **elif**: Adds additional conditions.
- **else**: Executes if all conditions are false.

By mastering these statements, you can create programs that make decisions and respond to different inputs!

The full lesson is now available on the Needi Developer YouTube channel.

Loops

Python me loops ka use tab hota hai jab hume ek block of code baar-baar repeat karna ho.
Do main loops hain:

- **for loop**
- **while loop**

1. For Loop

For loop ka use kisi sequence (list, tuple, string, ya range) ko iterate karne ke liye hota hai.
Ye har element ko ek-ek karke uthata hai aur code ko execute karta hai.

Syntax:

```
for variable in sequence:
    # code to execute
```

1: List Iterate Karna

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

for fruit in fruits:
    print(f"I love {fruit}!")
```

Output:

```
I love apple!
I love banana!
I love cherry!
```

2: Range ka Use Karna

```
for i in range(1, 6):
    print(f"Number: {i}")
```

Output:

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

2. While Loop

While loop ka use tab hota hai jab hume ek condition ke basis par code baar-baar chalana ho. Jab tak condition True hai, loop chalti rahegi.

Syntax:

```
while condition:
    # code to execute
```

1: Basic While Loop

```
count = 1

while count <= 5:
    print(f"Count: {count}")
    count += 1  # Increment the count
```

Output:

```
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

2: Infinite Loop (**Avoid Karna!**)

```
while True:
    print("This is an infinite loop!")  # Use `break` to stop
    break  # Stops the loop
```

Differences Between For and While

Feature	For Loop	While Loop
Use Case	Sequence (list, string, range)	Condition-based
When to Use	Fixed number of iterations	Unknown number of iterations
Example	Iterate over a list	Run until a condition is false

Examples for practice:

1: Attendance Check:

Aik aesi **For Loop** jo students ko present print kary gi:

```
students = ["Naveed", "Noor", "Shahzad", "Ahmed", "Ali"]

for student in students:
  print(f"{student} is present.")
```

Output:

```
Naveed is present.
Noor is present.
Shahzad is present.
Ahmed is present.
Ali is present.
```

Note:
ye program ek real attendance checker nahi hai ye just **For Loop** ki practice ky liye hai.

2. Countdown Timer:

While Loop ky zariye ek timer:

```
timer = 5

while timer > 0:
    print(f"Time remaining: {timer} seconds")
    timer -= 1

print("Time's up!")
```

Output:

```
Time remaining: 5 seconds
Time remaining: 4 seconds
Time remaining: 3 seconds
Time remaining: 2 seconds
Time remaining: 1 seconds
Time's up!
```

Note:

ye timer abhi foran hi chl pry ga es ko thik sy chlany ky liye **time** Module ko use kar ky **time.sleep(1)** loop me lgai gy to ek proper **Countdown Timer** bny ga.

Modules ko detail me **Lesson 10** me seekhy gy.

Summary:

- **For Loop**: Fixed iterations (loop over a list or range).
- **While Loop**: Runs until a condition is false.

Dono loops ko samajhne se aap repetitive tasks easily handle kar sakte hain!

The full lesson is now available on the Needi Developer YouTube channel.

Break, Continue, and Pass

Python me **break**, **continue**, aur **pass** special keywords hain jo loops aur control flow ko modify karte hain.
Ye keywords hume loops aur conditions ko zyada flexible aur controlable banane ka option dete hain.

1. Break Statement

Break ka use loop ko turant (immediately) terminate karne ke liye hota hai. Jab **break** execute hota hai, loop ka execution wahi khatam ho jata hai, chahe condition true ho ya nahi.

Syntax:

```
for/while loop:
    if condition:
        break
```

1: Break in a While Loop

```
count = 0

while count < 10:
    if count == 6:
        print("Loop stopped at count:", count)
        break
    print(count)
    count += 1
```

Output:

```
0
1
2
3
4
5
Loop stopped at count: 6
```

2: Break in a For Loop

```
for num in range(1, 10):
    if num == 5:
        print("Stopping the loop!")
        break
    print(num)
```

Output:

```
1
2
3
4
Stopping the loop!
```

2. Continue Statement

Continue ka use kisi specific iteration ko skip karne ke liye hota hai, lekin loop ke baaki iterations chalte hain. Ye turant loop ke agle iteration par chala jata hai.

Syntax:

```
for/while loop:
    if condition:
        continue
```

1: Continue in a For Loop

```
for num in range(1, 6):
    if num == 3:
        print("Skipping number 3")
        continue
    print(num)
```

Output:

```
1
2
Skipping number 3
4
5
```

2: Continue in a While Loop

```
count = 0

while count < 5:
    count += 1
    if count == 3:
        print("Skipping count:", count)
        continue
    print("Count:", count)
```

Output:

```
Count: 1
Count: 2
Skipping count: 3
Count: 4
Count: 5
```

3. Pass Statement

Pass ka use tab hota hai jab hume loop ya block of code likhna ho, lekin temporarily usse blank chodna ho. Ye kuch nahi karta, sirf syntax error avoid karta hai.

Syntax:

```
if condition:
    pass  # Placeholder for future code
```

2: Pass in an Empty Function

```
def my_function():
    pass  # Function logic will be added later
```

1: Pass in a Loop

```
for num in range(1, 6):
    if num == 3:
        pass  # Placeholder for future code
    print(num)
```

Output:

```
1
2
3
4
5
```

Comparison Between Break, Continue, and Pass

Keyword	Purpose	Effect
Break	Loop ko turant stop kar deta hai	Loop se bahar nikal jata hai
Continue	Current iteration ko skip karta hai	Agli iteration par jump karta hai
Pass	Placeholder, kuch nahi karta	Code ko temporarily blank chhodta hai

Summary:

- **Break**: Loop ko terminate karta hai.
- **Continue**: Current iteration ko skip karta hai.
- **Pass**: Placeholder, kuch nahi karta.

Inka use karke loops aur conditions ko efficiently manage kar sakte hain!

The full lesson is now available on the Needi Developer YouTube channel.

Shorthand If-Else

Python me **shorthand if-else** ka use tab hota hai jab aap simple conditions ko ek hi line me likhna chahte ho. Ye code ko concise aur readable banata hai.

1. Shorthand If

Agar ek hi condition hai aur uspar ek hi statement likhna ho, to **shorthand if** ka use kar sakte hain.

Syntax:

```
statement1 if condition else statement2
```

1: Simple Shorthand If

```
age = 18
if age >= 18: print("You are eligible to vote.")
```

Output:

```
You are eligible to vote.
```

2. Shorthand If-Else

Agar **if-else** ka use ek hi line me karna ho, to shorthand format ka use karte hain.

Syntax:

```
value_if_true if condition else value_if_false
```

2: Shorthand If-Else

```
age = 16
status = "Adult" if age >= 18 else "Minor"
print(status)
```

Output:

```
Minor
```

3. Nested Shorthand If-Else

Agar multiple conditions hain, to nested shorthand if-else ka use kar sakte hain.

3: Nested Shorthand If-Else

```
marks = 85
result = "Excellent" if marks > 80 else "Good" if marks > 50 else "Needs Improvement"
print(result)
```

Output:

```
Excellent
```

Summary:

Shorthand if-else ka use karke aapka code short aur clean ban jata hai. Ye sirf tab use karein jab conditions simple ho

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

ATM Simulation

Objective: Simulate a simple ATM machine with these features:

1. Show a menu with the following options:
    i. Check Balance
    ii. Withdraw Money
    iii. Exit
2. Use if-elif-else to handle the options.
3. Use a while loop to keep the program running until the user selects "Exit."
4. Implement the following logic:
    - Allow withdrawal only if the balance is sufficient.
    - Use shorthand if-else to update the balance.
    - Exit the loop when the user chooses "Exit."

Lesson complete—great job!

Lesson 5: Functions

Defining Functions and Return

Python mein functions aise code blocks hote hain jo ek specific task perform karte hain. Functions ka use code ko organize karne aur repeat karne se bachne ke liye hota hai. Isse code zyada readable aur maintainable ban jata hai.

Function Kya Hai?

Function ek block of code hota hai jo kisi specific kaam ko perform karta hai. Jab bhi zarurat ho, aap us function ko call kar sakte hain. Functions input le sakte hain, us input ko process kar sakte hain, aur result return kar sakte hain.

Function Ko Define Kaise Karte Hain?

Python mein function define karne ke liye **def** keyword ka use hota hai, aur phir function ka naam aur parentheses **()** likhna padta hai.

Syntax:

```
def function_name(parameters):
    # Code ka block
    return result  # Optional
```

1: Simple Function

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

greet()  # Function ko call kar rahe hain
```

Output:

```
Hello, welcome to Python Knight!
```

Function Mein Parameters

Aap function ko information dene ke liye parameters ka use kar sakte hain.

2: Function with Parameters

```
def greet_user(name):
    print(f"Hello, {name}!")

greet_user("Needi Developer")
```

Output:

```
Hello, Needi Developer!
```

Return Statement

return statement ka use function ke result ko wapas bhejne ke liye hota hai. Jab aap kisi calculation ya processing ke baad result ko function se bahar bhejna chahte hain, tab aap return use karte hain.

3: Function with Return

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

# Function ko call karke result print karte hain
sum_result = add(5, 3)
print("Sum:", sum_result)
```

Output:

```
Sum: 8
```

Why Use **return**?

Agar aapko function ke andar ki calculation ya result ko kisi aur part of program mein use karna ho, to **return** statement ke through wo value function ke bahar bhej sakte hain.

Summary:

- Function ko **def** keyword se define karte hain.
- Parameters function ko values dene ke liye use kiye jaate hain.
- Return statement function ka result wapas bhejne ke liye hota hai.

Is tarah se functions aapko apne code ko modular, clean, aur reusable banane mein madad karte hain.

The full lesson is now available on the Needi Developer YouTube channel.

Function Arguments

Python mein functions ke saath different types ke arguments use kiye ja sakte hain. In arguments ko hum **default arguments**, ***args**, aur ****kwargs** ke naam se jaane hain. Ye aapke function ko zyada flexible aur dynamic bana dete hain.

1. Default Arguments

Agar aap function define karte waqt kisi argument ko ek default value de dete hain, to agar user wo argument pass nahi kare, to wo default value use hoti hai.

Syntax:

```
def function_name(param1=value1, param2=value2):
    # Function body
```

Example:

```
def greet(name="User"):
    print(f"Hello, {name}!")

greet("Naveed")   # Output: Hello, Naveed!
greet()       # Output: Hello, User!
```

Explanation:

Yahan, agar **name** argument ko user pass nahi karta to default value **"User"** use hoti hai.

2. *args (Non-Keyword Variable Length Arguments)

***args** ka use hum tab karte hain jab hum function ko variable number of arguments dena chahte hain. **args** ek tuple hota hai jo user ke diye hue extra arguments ko store karta hai.

Syntax:

```
def function_name(*args):
    # Function body
```

Example:

```
def add_numbers(*args):
    total = 0
    for num in args:
        total += num
    return total

print(add_numbers(1, 2, 3))    # Output: 6
print(add_numbers(10, 20, 30, 40))  # Output: 100
```

Explanation:

***args** allow karta hai multiple numbers ko function mein pass karne ko. Function ke andar hum un sabhi numbers ko sum kar ke return kar rahe hain.

3. **kwargs (Keyword Variable Length Arguments)

****kwargs** ka use tab hota hai jab hume function ko multiple keyword arguments (key-value pairs) dene hote hain. **kwargs** ek dictionary hota hai jo key-value pairs ko store karta hai.

Syntax:

```
def function_name(**kwargs):
    # Function body
```

Example:

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Ali", age=25, city="Karachi")
```

Output:

```
name: Ali
age: 25
city: Karachi
```

Explanation:

****kwargs** allow karta hai aapko multiple keyword arguments ko handle karne mein. Function ke andar hum in key-value pairs ko print kar rahe hain.

Combining Default, *args, and **kwargs

Aap ek hi function mein **default arguments**, ***args**, aur ****kwargs** ko combine kar sakte hain. Lekin, default arguments ko hamesha sabse aakhri mein rakhna hota hai.

Example:

```
def student_info(name, age=18, *subjects, **marks):
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"Subjects: {subjects}")
    print(f"Marks: {marks}")

# Function ko call karte hain
student_info("Ali", 20, "Math", "Science", Math=85, Science=90)
```

Output:

```
Name: Ali
Age: 20
Subjects: ('Math', 'Science')
Marks: {'Math': 85, 'Science': 90}
```

Explanation:

- **name** ek required argument hai.
- **age** ek default argument hai, agar value pass nahi hoti to 18 use hota.
- ***subjects** multiple subjects ko accept karta hai.
- ****marks** multiple subjects ke marks ko key-value pairs ke roop mein accept karta hai.

Summary:

- **Default Arguments**: Function ko default values dena agar user kisi argument ko pass nahi karta.
- ***args**: Function ko variable number of non-keyword arguments dena. Ye arguments tuple ke form mein hote hain.
- ****kwargs**: Function ko variable number of keyword arguments dena. Ye arguments dictionary ke form mein hote hain.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

Prime Number Check

Write a Python function to check whether a given number is prime or not. A prime number is a number greater than 1 that has no divisors other than 1 and itself.

Write a function **is_prime()** that:

1. Accepts an integer as input.
2. Returns **True** if the number is prime.
3. Returns **False** if the number is not prime.

Example Input and Output:

```
is_prime(5)  # Output: True (5 is a prime number)
is_prime(10)  # Output: False (10 is not a prime number)
```

Lesson complete—great job!

Lesson 6: Data Structures

Lists

Lists Python mein ek data structure hain jo multiple items ko ek variable mein store karne ki facility dete hain. Lists kaafi flexible hoti hain aur inko hum asaan tareeqe se modify kar sakte hain.
Aayein basics samajhte hain:

Basics of Lists

- Lists ko square brackets **[]** ke andar banaya jata hai.
- Har item ko comma **,** se separate karte hain.
- Lists mein koi bhi data type store ho sakta hai: numbers, strings, aur even lists khud bhi.
- Lists mutable hoti hain, yani aap unke elements ko badal sakte hain.

Examples:

```
# List banani
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "cherry"]
mixed = [1, "hello", 3.5, True]

# List ko print karna
print(numbers)  # [1, 2, 3, 4, 5]
print(fruits)   # ['apple', 'banana', 'cherry']
print(mixed)    # [1, 'hello', 3.5, True]
```

Indexing in Lists

- Lists ke har element ka ek index hota hai, jo 0 se start hota hai.
- Negative indexing se hum list ke end se elements ko access karte hain.

Examples:

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

# Positive indexing
print(fruits[0])  # 'apple' (pehla element)
print(fruits[1])  # 'banana' (dusra element)

# Negative indexing
print(fruits[-1])  # 'cherry' (last element)
print(fruits[-2])  # 'banana' (dusra last element)
```

Slicing in Lists

- **Slicing** ka matlab hai list ke kisi part ko access karna.
- Syntax: **list[start:end:step]**
  - **start**: Jahaan se slicing shuru karni hai (inclusive).
  - **end**: Jahaan tak slicing karni hai (exclusive).
  - **step**: Kitne gap ke saath elements ko pick karna hai.

Examples:

```
numbers = [10, 20, 30, 40, 50, 60]

# Simple slicing
print(numbers[1:4])  # [20, 30, 40] (index 1 se 3 tak)
print(numbers[:3])   # [10, 20, 30] (start missing => 0 se start)
print(numbers[2:])   # [30, 40, 50, 60] (end missing => last tak)

# Step ke saath slicing
print(numbers[0:6:2])  # [10, 30, 50] (har 2nd element pick karo)
print(numbers[::-1])   # [60, 50, 40, 30, 20, 10] (list reverse karna)
```

Methods in Lists

Python mein lists ke saath kaafi useful **built-in methods** hote hain jo humein unhe manipulate karne ka tareeqa dete hain.

Common List Methods:

1. **append()**: Ek naye element ko list ke end mein add karta hai.
2. **extend()**: Ek list ke elements ko doosri list mein add karta hai.
3. **insert()**: Kisi specific position par element ko add karta hai.
4. **remove()**: Specific element ko remove karta hai.
5. **pop()**: Last element ko remove karta hai aur return karta hai.
6. **sort()**: List ke elements ko ascending order mein arrange karta hai.
7. **reverse()**: List ke elements ko reverse karta hai.

Examples of Methods:

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

# append
fruits.append("orange")
print(fruits)  # ['apple', 'banana', 'cherry', 'orange']

# insert
fruits.insert(1, "grape")
print(fruits)  # ['apple', 'grape', 'banana', 'cherry', 'orange']

# remove
fruits.remove("banana")
print(fruits)  # ['apple', 'grape', 'cherry', 'orange']

# pop
last_item = fruits.pop()
print(last_item)  # 'orange'
print(fruits)     # ['apple', 'grape', 'cherry']

# reverse
fruits.reverse()
print(fruits)  # ['cherry', 'grape', 'apple']
```

The full lesson is now available on the Needi Developer YouTube channel.

Tuples

What are Tuples?

Tuples Python me ek data structure hain jo lists ki tarah lagte hain, lekin ek major difference hota hai: **Tuples immutable** hote hain, yani once aap tuple create kar lete hain, uske values ko change nahi kar sakte. Tuples un data ko store karne ke liye use hote hain jo **constant** rehna chahiye.

- Tuples ko round brackets **()** ka use karke banaya jata hai.
- Ye kisi bhi type ka data hold kar sakte hain (int, string, float, etc.).
- Tuples mixed data types bhi store kar sakte hain.

How to Create Tuples

```
# Ek tuple create karna
fruit_tuple = ("apple", "banana", "cherry")

# Mixed data types ka tuple
mixed_tuple = (1, "hello", 3.14)

# Empty tuple
empty_tuple = ()

# Ek element ka tuple (comma zaruri hai)
single_element = (5,)
print(type(single_element))  # Output: <class 'tuple'>
```

Why Are Tuples Immutable?

Immutable ka matlab hai ki aap tuple ko banane ke baad uska data change, add, ya remove nahi kar sakte. Ye useful hota hai jab aapko apna data accidentally modify hone se bachana ho.

Example: Immutability

```
fruit_tuple = ("apple", "banana", "cherry")

# Ye error dega
fruit_tuple[1] = "orange"
# TypeError: 'tuple' object does not support item assignment
```

Accessing Tuple Elements

Tuple ke elements ko indexing ka use karke access kiya ja sakta hai, bilkul lists ki tarah.

Example: Immutability

```
fruit_tuple = ("apple", "banana", "cherry")

# Elements ko access karna
print(fruit_tuple[0])  # Output: apple
print(fruit_tuple[-1])  # Output: cherry
```

Use Case of Tuple

1. Fixed Data:

Jab aapko data ko change nahi karna ho, jaise:

- Week ke days: **days = ("Monday", "Tuesday", "Wednesday")**
- RGB color codes: **colors = (255, 0, 0)**

2. Dictionary Keys:

Tuples ko dictionary ke keys ke roop me use kar sakte hain (lists ko nahi).

```
coordinates = {(10, 20): "Point A", (30, 40): "Point B"}
print(coordinates[(10, 20)])  # Output: Point A
```

3. Multiple Values Return Karna:

Functions me tuples ka use multiple values return karne ke liye hota hai.

```
def get_coordinates():
    return (10, 20)

x, y = get_coordinates()
print(x, y)  # Output: 10 20
```

Tuples ke Advantages

- Lists se fast hote hain kyunki immutable hain.
- Data ko safe rakhte hain accidental modification se.
- Jahan immutability ki zarurat ho, wahan use hote hain (dictionary keys).

The full lesson is now available on the Needi Developer YouTube channel.

Sets

Python me **sets** ek special data structure hain jo **unordered** aur **unique elements** ko store karte hain. **Sets** real-life examples jese classroom ke students ya fruits ki list ko handle karne ke liye use hote hain, jahan duplicates ko ignore karna zaroori hai.

Set Basics

- Sets ko curly brackets **{}** ya **set()** function ka use karke banaya jata hai.
- Sets unordered hain, yani elements ka order important nahi hota.
- Sets ke andar duplicates nahi ho sakte.

Example: Set Creation

```
# Ek set create karna
fruits = {"apple", "banana", "cherry", "apple"}
print(fruits)
# Output: {'banana', 'cherry', 'apple'} (duplicates remove ho jayenge)
# Empty set
empty_set = set()  # {} ka matlab dictionary hota hai, na ki set
```

Set Operations

Sets ko use karke hum mathematical operations kar sakte hain, jaise **union**, **intersection**, **difference**, **symmetric difference**, etc.

1. Union

- Union ka matlab hai do sets ke **sabhi unique elements** ko combine karna.
- Use the **|** operator ya **union()** method.

```
A = {1, 2, 3}
B = {3, 4, 5}

# Union
result = A | B
print(result)  # Output: {1, 2, 3, 4, 5}

# Using method
result = A.union(B)
print(result)  # Output: {1, 2, 3, 4, 5}
```

2. Intersection

- Intersection ka matlab hai do sets ke common elements.
- Use the **&** operator ya **intersection()** method.

```
A = {1, 2, 3}
B = {3, 4, 5}

# Intersection
result = A & B
print(result)  # Output: {3}

# Using method
result = A.intersection(B)
print(result)  # Output: {3}
```

3. Difference

- Difference ka matlab hai ek set me jo elements hain, lekin doosre set me nahi.
- Use the **-** operator ya **difference()** method.

```
A = {1, 2, 3}
B = {3, 4, 5}

# Difference (A - B)
result = A - B
print(result)  # Output: {1, 2}

# Using method
result = A.difference(B)
print(result)  # Output: {1, 2}
```

4. Symmetric Difference

- Symmetric Difference ka matlab hai do sets ke unique elements, jo sirf ek set me hain, lekin dono me nahi.
- Use the **^** operator ya **symmetric_difference()** method.

```
A = {1, 2, 3}
B = {3, 4, 5}

# Symmetric Difference
result = A ^ B
print(result)  # Output: {1, 2, 4, 5}

# Using method
result = A.symmetric_difference(B)
print(result)  # Output: {1, 2, 4, 5}
```

Set Methods

Sets me kaafi useful methods hote hain jo humari life easy banate hain.

Common Methods

1. add():

Ek naya element set me add karta hai.

```
fruits = {"apple", "banana"}
fruits.add("cherry")
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

2. remove():

Ek specific element ko remove karta hai (error deta hai agar element na ho).

```
fruits = {"apple", "banana"}
fruits.remove("banana")
print(fruits)  # Output: {'apple'}
```

3. discard():

Ek element ko remove karta hai (error nahi deta agar element na ho).

```
fruits = {"apple", "banana"}
fruits.discard("orange")
print(fruits)  # Output: {'apple', 'banana'}
```

4. pop():

Randomly ek element ko remove karta hai (kyunki sets unordered hote hain).

```
fruits = {"apple", "banana", "cherry"}
fruits.pop()
print(fruits)  # Output: {'banana', 'cherry'} (order random hoga)
```

5. clear():

Set ko completely empty karta hai.

```
fruits = {"apple", "banana", "cherry"}
fruits.clear()
print(fruits)  # Output: set()
```

6. copy():

Ek naya set banata hai jo existing set ka copy hota hai.

```
fruits = {"apple", "banana"}
new_fruits = fruits.copy()
print(new_fruits)  # Output: {'apple', 'banana'}
```

Why Use Sets?

- Jab duplicates ko avoid karna ho.
- Jab fast membership testing chahiye (element exists in set or not).
- Jab mathematical set operations jese union ya intersection chahiye.

Sets simple aur efficient hain for unique data ke saath kaam karna!

The full lesson is now available on the Needi Developer YouTube channel.

Dictionaries

Dictionaries Python me ek **unordered**, **mutable**, aur **indexed** data structure hain jo **key-value** pairs ko store karte hain. Yeh ek bahut powerful tool hai data ko logically organize karne ke liye, jese ek contact list ya students ki records.

Dictionaries Basics

- Dictionary ko curly brackets **{}** ka use karke banate hain.
- **Keys** unique hote hain aur **values** duplicate ho sakti hain.
- Keys immutable hote hain (string, int, tuple), aur values kuch bhi ho sakti hain (string, list, another dictionary).

Example: Set Creation

```
# Basic dictionary
student = {
    "name": "Naveed",
    "age": 20,
    "grade": "A"
}
print(student)  # Output: {'name': 'Naveed', 'age': 20, 'grade': 'A'}
```

Accessing Values in Dictionary

1. Using Keys

```
print(student["name"])  # Output: Naveed
```

2. Using **get()** Method

get() ka fayda hai agar key exist na kare, to error nahi hota, instead default value return hoti hai.

```
print(student.get("age"))  # Output: 20
print(student.get("address", "Not Available"))
 # Output: Not Available
```

Adding and Updating Values

1. Adding New Key-Value Pair

```
student["address"] = "Gujranwala"
print(student)
# Output: {'name': 'Naveed', 'age': 20, 'grade': 'A', 'address': 'Gujranwala'}
```

2. Updating Existing Value

```
student["age"] = 21
print(student)
# Output: {'name': 'Naveed', 'age': 21, 'grade': 'A', 'address': 'Gujranwala'}
```

Removing Elements

1. Using **pop()**

Ek specific **key-value** pair ko remove karta hai.

```
student.pop("grade")
print(student)
# Output: {'name': 'Naveed', 'age': 21, 'address': 'Gujranwala'}
```

2. Using **popitem()**

Randomly last inserted **key-value** pair ko remove karta hai (Python 3.7+ me last element remove hota hai).

```
student.popitem()
print(student)
# Output: {'name': 'Naveed', 'age': 21'}
```

3. Using **del**

Kisi specific key ko delete karne ke liye.

```
del student["age"]
print(student)  # Output: {'name': 'Naveed'}
```

4. Using **clear()**

Dictionary ko completely empty karne ke liye.

```
student.clear()
print(student)  # Output: {}
```

Common Dictionary Methods

Method	Description	Example
keys()	Returns all keys in dictionary	**student.keys() → dict_keys(['name', 'age'])**
values()	Returns all values in dictionary	**student.values() → dict_values(['Ali', 20])**
items()	Returns all key-value pairs as tuples	**student.items() → dict_items([('name', 'Ali')])**
update()	Adds or updates key-value pairs	**student.update({"grade": "A"})**
copy()	Creates a shallow copy of the dictionary	**new_student = student.copy()**

Why Use Dictionaries?

1. **Fast Lookups**: Keys ki wajah se data ko quickly access kar sakte hain.
2. **Structured Data**: Real-world data ko logically organize karne ke liye perfect hain.
3. **Flexible**: Values me kuch bhi store kar sakte hain (list, another dictionary).

Dictionaries are powerful and versatile, aur aapko har practical project me inka use milega!

The full lesson is now available on the Needi Developer YouTube channel.

Enumerate and Zip Functions

Python ki **enumerate()** aur **zip()** functions bohot useful hain, jo looping aur data ko combine karne ke tasks ko simple aur effective banate hain.

Enumerate Function

enumerate() ka use iterables (like lists, tuples, strings) ke elements ke saath-saath unka index access karne ke liye hota hai.

Syntax:

```
enumerate(iterable, start=0)
```

- **iterable**: Koi bhi iterable object (list, tuple, string).
- **start**: Optional. Ye specify karta hai index kis number se shuru hoga (default is 0).

1: Basic Usage

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

for index, fruit in enumerate(fruits):
    print(f"Index: {index}, Fruit: {fruit}")
```

Output:

```
Index: 0, Fruit: apple
Index: 1, Fruit: banana
Index: 2, Fruit: cherry
```

2: Custom Start Index

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

for index, fruit in enumerate(fruits, start=1):
    print(f"Fruit {index}: {fruit}")
```

Output:

```
Fruit 1: apple
Fruit 2: banana
Fruit 3: cherry
```

Zip Function

zip() ka use multiple iterables ko parallel combine karne ke liye hota hai. Ye function tuples ka ek list banata hai, jisme har tuple respective elements ko combine karta hai.

Syntax:

```
zip(iterable1, iterable2, ...)
```

1: Basic Usage

```
names = ["Naveed", "Sara", "Fatima"]
scores = [90, 85, 88]

for name, score in zip(names, scores):
    print(f"{name} scored {score}")
```

Output:

```
Naveed scored 90
Sara scored 85
Fatima scored 88
```

2: Zipping Unequal Lengths

Agar iterables ke lengths unequal hain, **zip()** shorter iterable ke length tak hi combine karega.

```
colors = ["red", "blue"]
shapes = ["circle", "square", "triangle"]

for color, shape in zip(colors, shapes):
    print(f"{color} {shape}")
```

Output:

```
red circle
blue square
```

3: Unzipping

Agar zipped object ko separate karna ho, to **zip(*zipped)** ka use karte hain.

```
zipped = [("Naveed", 90), ("Sara", 85), ("Fatima", 88)]
names, scores = zip(*zipped)

print(names)  # ('Naveed', 'Sara', 'Fatima')
print(scores)  # (90, 85, 88)
```

Difference Between Enumerate and Zip

Feature	**enumerate()**	**zip()**
Purpose	Provides index and value	Combines multiple iterables
Input	Single iterable	Two or more iterables
Output	Tuples with index and value	Tuples with combined elements

Summary:

- **enumerate()** helps you get index and value from an iterable.
- **zip()** combines elements from multiple iterables into tuples.

Dono functions looping ko zyada organized aur readable banate hain.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

Data Structures Puzzle

Create a program that performs the following tasks:

1. You have two data structures:
    - A list of students:

    ["Ali", "Naveed", "Sara", "Zoya", "Ali", "Sara"]
    - A tuple of their marks: **(85, 90, 78, 88, 85, 78)**
2. Perform these operations:
    - **Find unique** students (remove duplicates).
    - **Combine** students and marks into a dictionary where names are keys, and marks are values (for duplicate names, keep the latest mark).

Lesson complete—great job!

Lesson 7: Error Handling

Introduction to Errors and Exceptions

Errors aur **exceptions** Python programming ka important part hain. Ye humein apne code mein problems ko samajhne aur handle karne mein madad karte hain. Aayiye, inko asaan words mein samajhte hain.

What Are Errors?

Errors wo problems hain jo code ko chalne nahi deti. Jab Python ko kuch aisa mile jo samajhne ya chalane layak na ho, to user ko python ki trf say error mita hai.

Types of Errors:

1. Syntax Errors:

Jab Python ke rules tod diye jaate hain.

Example:

```
print("Hello  # Closing quote missing
```

Output:

```
SyntaxError: EOL while scanning string literal
```

2. Runtime Errors:

Jab program chal raha hota hai aur tab issue hota hai.

Example:

```
result = 10 / 0  # Zero se divide karne ki koshish
```

Output:

```
ZeroDivisionError: division by zero
```

What Are Exceptions?

Exceptions runtime errors hain jo program ko crash kiye bina handle ki jaa sakti hain.

Example Without Exception Handling:

```
num = int(input("Enter a number: "))
print(f"You entered: {num}")
# Agar user number ki jagah text input karega, program crash karega.
```

Output:

```
ValueError: invalid literal for int() with base 10
```

Why Handle Exceptions?

- Program ko crash hone se bachane ke liye.
- User-friendly experience dene ke liye.
- Unexpected situations ko handle karne ke liye.

The full lesson is now available on the Needi Developer YouTube channel.

How to Handle Exceptions?

Python mein **errors** aur **exceptions** handle karne ke liye **try**, **except**, aur **finally** blocks ka use hota hai. Ye blocks program ko crash hone se bachate hain aur unexpected errors ko properly handle karte hain. Aayiye, inko asaan tarike se samajhte hain!

try Block

try block wo code contain karta hai jo execute karna hai, lekin usmein koi error aane ka chance ho sakta hai.

Example:

```
try:
    num = int(input("Enter a number: "))  # Risky code
    print(f"You entered: {num}")
```

Agar **try** block mein koi error hoti hai, to code **except** block mein chala jata hai.

except Block

except block tab execute hota hai jab **try** block mein koi error occur hoti hai. Ye error handle karta hai aur user ko friendly message provide karta hai.

Example:

```
try:
    num = int(input("Enter a number: "))  # User se number lena
    print(f"Result: {10 / num}")  # Division by user input
except ValueError:  # Agar user number na de (ValueError)
    print("Please enter a valid number.")
except ZeroDivisionError:  # Agar user 0 input kare
    print("You cannot divide by zero!")
```

Output Scenarios:

- User enters "abc":

```
Please enter a valid number.
```

- User enters "0":

```
You cannot divide by zero!
```

finally Block

finally block hamesha execute hota hai, chahe koi error aaye ya nahi. Ye clean-up tasks ke liye use hota hai, jaise file close karna, database connection terminate karna, etc.

Example:

```python
try:
    file = open("data.txt", "r")  # File ko open karna
    content = file.read()
    print(content)
except FileNotFoundError:  # Agar file exist nahi karti
    print("File not found!")
finally:
    print("Closing the file.")  # Hamesha execute hoga
    file.close()  # File close karna
```

Output:

Agar file mil jaye:

```
(File content displayed)
Closing the file.
```

Agar file na mile:

```
File not found!
Closing the file.
```

Multiple **except** Block

Aap ek **try** block ke liye multiple **except** blocks use kar sakte ho. Ye alag-alag errors ko handle karte hain.

Example:

```
try:
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
    print(f"Result: {result}")
except ValueError:  # Agar invalid input ho
    print("Please enter valid numbers.")
except ZeroDivisionError:  # Agar zero se divide ho
    print("Cannot divide by zero.")
finally:
    print("Thank you for using the program.")
```

Output Scenarios:

User enters valid numbers:

```
Result: (calculated result)
Thank you for using the program.
```

User enters invalid numbers:

```
Please enter valid numbers.
Thank you for using the program.
```

User divides by zero:

```
Cannot divide by zero.
Thank you for using the program.
```

Summary:

1. **try block**: Risky code jo error raise kar sakta hai.
2. **except block**: Error ko handle karta hai aur program ko crash hone se bachata hai.
3. **finally block**: Clean-up tasks ko execute karta hai aur hamesha run hota hai.

By using **try**, **except**, and **finally**, aap apne Python programs ko **error-proof** aur **user-friendly** bana sakte ho.

The full lesson is now available on the Needi Developer YouTube channel.

Raising Custom Errors

Python mein aap apne custom errors raise kar sakte ho jab aapko lagta hai ki koi specific condition galat hai. Ye feature program ko robust aur readable banata hai, kyunki aap apne errors ko identify kar sakte ho aur proper error messages provide kar sakte ho.

Why Raise Custom Errors?

- Jab koi specific logic fail ho.
- User ko meaningful error message dena ho.
- Program ko controlled way mein stop karna ho.

Example:

```
raise ExceptionType("Custom error message")
```

Common Use Case

Imagine karo ek function hai jo kisi user ki age accept karta hai. Lekin, agar age 0 se chhoti ya 150 se zyada ho, to wo invalid hai.

1: Division by Zero Custom Error

```
def divide(a, b):
    if b == 0:
        raise ZeroDivisionError("You cannot divide by zero!")
    return a / b

try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print(f"Error: {e}")
```

Output:

```
Error: You cannot divide by zero!
```

2: Raising Custom Errors

Example:

```
def check_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative!")  # Custom error
    elif age > 150:
        raise ValueError("Age cannot be more than 150!")
        # Custom error
    else:
        print(f"Your age is valid: {age}")

# Test cases
try:
    check_age(-5)  # Invalid age
except ValueError as e:
    print(f"Error: {e}")

try:
    check_age(200)  # Invalid age
except ValueError as e:
    print(f"Error: {e}")

check_age(25)  # Valid age
```

Output:

```
Error: Age cannot be negative!
Error: Age cannot be more than 150!
Your age is valid: 25
```

3: Custom Class for Errors

Python mein aap apne custom error classes bhi bana sakte ho by inheriting from the **Exception** class.

Example:

```
class NegativeNumberError(Exception):
    """Raised when a number is negative"""
    pass

def check_number(num):
    if num < 0:
        raise NegativeNumberError("Number cannot be negative!")
    print(f"Valid number: {num}")

# Test case
try:
    check_number(-10)
except NegativeNumberError as e:
    print(f"Custom Error: {e}")
```

Output:

```
Custom Error: Number cannot be negative!
```

Using **assert** for Error Raising

Python mein **assert** bhi use hota hai specific conditions ko test karne ke liye. Agar condition fail ho, to error raise hoti hai.

Example:

```
def check_positive(num):
    assert num > 0, "Number must be positive!"
    print(f"Valid number: {num}")

# Test cases
check_positive(5)  # Valid
check_positive(-3)  # Raises AssertionError
```

Output:

```
Valid number: 5
AssertionError: Number must be positive!
```

Key Points to Remember:

1. Use **raise** to manually trigger an error.
2. Always provide meaningful error messages.
3. Create custom exceptions by inheriting from the **Exception** class.
4. Combine **try-except** blocks with custom errors for clean code.

Custom errors aapko program ko predictable aur user-friendly banane mein help karte hain.

The full lesson is now available on the Needi Developer YouTube channel.

Debugging Basics

Debugging ka matlab hota hai apne code ki errors ko dhoondhna aur fix karna. Python debugging ke liye kai tareeqe provide karta hai, jisme se **print() debugging** aur **pdb module** sabse common aur effective hain.

1. print() Debugging

Yeh debugging ka sabse simple aur common tareeqa hai. Aap code ke different parts mein print() statements add karte ho taaki variables ki values aur flow ka pata chal sake.

- Advantages:
  - Simple aur quick.
  - Beginners ke liye best.

Example:

```
def calculate_sum(a, b):
    print(f"a: {a}, b: {b}")  # Debugging
    result = a + b
    print(f"result: {result}")  # Debugging
    return result

# Calling the function
sum_result = calculate_sum(5, 3)
print(f"Sum is: {sum_result}")
```

Output:

```
a: 5, b: 3
result: 8
Sum is: 8
```

When to Use print() Debugging?

- Jab aapko variable ki value check karni ho.
- Flow of execution samajhna ho.
- Small programs mein.

Limitations of print() Debugging

- Large programs mein clutter create kar deta hai.
- Temporary fixes ke liye theek hai, lekin permanent debugging ke liye nahi.

2. pdb (Python Debugger)

Python ka built-in **pdb module** ek interactive debugging tool hai jo aapko code ko line-by-line run karke errors dhoondhne deta hai.

How to Use pdb?

1. Code mein **import pdb** likhein.
2. Debugging karne ke liye **pdb.set_trace()** ka use karein.
3. Program execution wahin stop hota hai jahan **pdb.set_trace()** lagaya ho.
4. Interactive commands ka use karke debugging karein.

Example:

```
def calculate_product(a, b):
    import pdb; pdb.set_trace()  # Debugging point
    result = a * b
    return result

# Calling the function
product = calculate_product(4, 5)
print(f"Product is: {product}")
```

Common pdb Commands:

Command	Description
n	Next line execute karne ke liye.
c	Continue program execution.
q	Quit the debugger.
p	Variable ki value print karne ke liye.
l	Current line aur uske aas-paas ka code dekhne ke liye.

pdb Example with Commands

```
def divide_numbers(a, b):
    import pdb; pdb.set_trace()  # Start debugging
    result = a / b
    return result

# Call the function
result = divide_numbers(10, 2)
print(f"Result: {result}")
```

Output:

```
(Pdb) p a
10
(Pdb) p b
2
(Pdb) n
(Pdb) p result
5.0
(Pdb) c
Result: 5.0
```

3. Using Breakpoints in VS Code

Agar aap **VS Code** use kar rahe hain, to breakpoints debugging ke liye aur bhi easy ho jaati hai.

1. **Set a Breakpoint**: Code ke kisi line pe click karke breakpoint lagao.
2. **Run in Debug Mode**: "Run and Debug" option choose karo.
3. **Inspect Variables**: Execution stop hoga aur aap variables inspect kar sakte ho.

When to Use pdb?

- Jab program bada ho aur **print()** se debugging mushkil lage.
- Jab aapko line-by-line execution inspect karna ho.

Summary:

1. **print() Debugging**:
    - Simple aur quick.
    - Best for small issues.
2. **pdb Debugger**:
    - Advanced debugging tool.
    - Large programs ke liye best.

Don’t just fix errors—learn from them! Debugging is not just a skill, it's an art.

Lesson complete—great job!

Lesson 8: File Handling

Reading and Writing Files

File handling ka matlab hai files ko **read**, **write**, aur manage karna. Python ek powerful system provide karta hai files ke saath kaam karne ke liye. Ye concept real-world tasks jese logs save karna, data read karna ya report generate karna ke liye kaafi useful hai.

Why File Handling is Important?

- Data store karne ke liye.
- Logs maintain karne ke liye.
- Configuration files manage karne ke liye.

Opening a File

Python mein files ko open karne ke liye **open()** function use hota hai.

Syntax:

```
file = open("filename", "mode")
```

Mode:

Mode	Description
r	Read mode (default).
w	Write mode (overwrites file).
a	Append mode (add to the file).
x	Create a new file.

Reading a File

File ko read karne ke liye **open()** function ke saath **r** mode use hota hai.

Example:

```
# Writing to a file
file = open("example.txt", "w")
file.write("Hello, this is a new line!\n")  # Overwrites the file
file.write("Python makes file handling easy!\n")
file.close()
```

Append Mode Example:

```
# Appending to a file
file = open("example.txt", "a")
file.write("This line is added at the end.\n")
file.close()
```

Reading and Writing Together

r+ mode allow karta hai ek hi file ko read aur write karne ke liye.

Example:

```
file = open("example.txt", "r+")
content = file.read()
print("Current content:", content)
file.write("\nAdding this line after reading!")
file.close()
```

Using the with Statement

Python mein **with** statement ka use file ko automatically close karne ke liye hota hai.

Example:

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)  # No need to manually close
```

Common Errors in File Handling

1. File Not Found Error:

Jab file exist nahi karti:

Example:

```
file = open("nonexistent.txt", "r")  # Will raise an error
```

Fix: Use **try-except** ya **x** mode to create the file.

2. Permission Error:

Jab file ke liye access deny ho.

Example:

```
file = open("protected_file.txt", "w")  # May raise an error
```

Conclusion:

1. File handling ka basic concept samajhna zaroori hai.
2. Use **read()**, **write()**, **append()** for basic operations.
3. Prefer with **open()** for safer file handling.
4. Use file handling for real-world tasks like data storage, logging, and report generation.

File handling is an essential skill for every Python developer!

The full lesson is now available on the Needi Developer YouTube channel.

Important Methods in File Handling

File handling mein **read()**, **readlines(),** aur **write()** methods kaafi important hain. Ye methods file se data read karne aur write karne ke liye use hote hain. Let's understand these methods step by step in an easy way.

1. **read()** Method

read() method pura file ka content ek string ke form mein return karta hai.

Example:

```
with open("example.txt", "r") as file:
    content = file.read()  # Reads the entire file content
    print(content)
```

When to Use:

- Jab file ka sara data ek saath read karna ho.
- Best for small files.

Output:

If the file contains:

```
Hello, Python Knights!
Welcome to Python File Handling.
```

Output will be:

```
Hello, Python Knights!
Welcome to Python File Handling.
```

2. **readlines()** Method

readlines() method file ki har line ko ek list ke andar store karta hai.

Example:

```
with open("example.txt", "r") as file:
    lines = file.readlines() # Reads all lines into a list
    print(lines)
```

When to Use:

- Jab aapko file ki lines ko alag-alag process karna ho.
- Useful for looping through lines.

Output:

If the file contains:

```
Hello, Python Knights!
Welcome to Python File Handling.
```

Output will be:

```
['Hello, Python Knights! \n', 'Welcome to Python File Handling. \n']
```

Iterating Through Lines:

```
with open("example.txt", "r") as file:
    lines = file.readlines()
    for line in lines:
        print(line.strip())
# Removes extra spaces or newline characters
```

3. **write()** Method

write() method file mein data likhne ke liye use hota hai. Agar file already exist karti hai, toh content overwrite ho jata hai.

Example:

```
with open("example.txt", "w") as file:
    file.write("This is a new file content.\n")
    file.write("File handling is easy in Python!")
```

When to Use:

- Jab file mein naya content likhna ho.
- Note: Write mode purane content ko delete kar deta hai

Output in File:

```
This is a new file content.
File handling is easy in Python!
```

Appending Content Using **write()**

Agar aapko existing file mein data add karna ho, toh **a** mode ka use karen.

Example:

```
with open("example.txt", "a") as file:
    file.write("\nAppending a new line to the file.")
```

Output in File:

```
This is a new file content.
File handling is easy in Python!
Appending a new line to the file.
```

Comparison of Methods

Method	Description	Best Use Case
read()	Reads entire file as a string.	Small files.
readlines()	Reads all lines into a list.	Line-by-line processing.
write()	Writes content to a file (overwrites).	New files or overwriting.

Conclusion:

- Use **read()** to get all data at once.
- Use **readlines()** to process file line-by-line.
- Use **write()** to save data into a file.

In Python, file handling is both simple and powerful, making it easy to work with text data in real-world applications.

The full lesson is now available on the Needi Developer YouTube channel.

Understanding **seek()** and **tell()** Functions in File Handling

In Python, **seek()** aur **tell()** functions ka use file ke andar pointer ko control karne ke liye hota hai. Yeh functions tab kaam aate hain jab hume file ke specific part ko read ya write karna hota hai.

What is File Pointer?

Jab hum ek file open karte hain, file pointer ek cursor ki tarah hota hai jo yeh batata hai ki agla read ya write operation file ke kis position par hoga. By default, file pointer file ke start me hota hai.

seek() Function

seek() ka use file pointer ko kisi specific position par move karne ke liye hota hai.

Syntax:

```
file.seek(offset, whence)
```

- **offset**: Pointer ko kitne bytes move karna hai.
- **whence**: Reference point define karta hai:
  - **0** (default): File ke start se count karna.
  - **1**: Current pointer position se count karna.
  - **2**: File ke end se count karna.

Pointer ko Move Karna

Example:

```
with open("example.txt", "r") as file:
    file.seek(5)  # Pointer ko 5th byte par move karo
    content = file.read()  # 5th byte ke baad read karo
    print(content)
```

Agar **example.txt** file me yeh content hai:

```
Hello, Python Knights!
```

Output hoga:

```
, Python Knights!
```

Last 5 Bytes Read Karna

Example:

```
with open("example.txt", "rb") as file:
  file.seek(-5, 2) # Move 5 bytes before the end of the file
  content = file.read()
  print(content.decode('utf-8'))
# Decode the binary content to string
```

Output hoga:

```
ghts!
```

Es example me **rb** es liye use kiya gaya kyu kay **r** file ko text mode me kholta hai. Text mode me **seek()** function ka behavior unreliable ho jata hai, kyunki text files line endings ko normalize karte hain (**\n** vs **\r\n**), jo byte-level positioning ko bigaad sakta hai.
Es liye **seek(-5, 2)** kaam nahi karega agar text mode hai, kyunki yeh binary mode ke liye design hua hai.

Key Differences:

Aspect	**First Code (rb)**	**Second Code (r)**
Mode	Binary Mode (**rb**)	Text Mode (**r**)
Data Type	Bytes	String
seek() Behavior	Accurate Byte-Level Positioning	May Fail (due to line-ending handling)
Encoding	Requires **.decode('utf-8')** explicitly	Implicitly handled by Python
Use Case	Binary files, precise positioning	Text files with normal reading

tell() Function

tell() function current file pointer ki position batata hai (in bytes).

Syntax:

```
file.tell()
```

Pointer Position Check Karna

```
with open("example.txt", "r") as file:
    print(f"Initial position: {file.tell()}")
    file.read(6)  # Pehle 6 bytes read karo
    print(f"Position after reading 6 bytes: {file.tell()}")
```

Output:

```
Initial position: 0
Position after reading 6 bytes: 6
```

how to use together **seek()** and **tell()** function

```
with open("example.txt", "r") as file:
    file.seek(7)  # Pointer ko 7th byte par move karo
    print(f"Pointer position: {file.tell()}")
    content = file.read(5)  # Agle 5 bytes read karo
    print(f"Content read: {content}")
    print(f"Pointer ab hai: {file.tell()}")
```

Key Points to Remember:

1. **seek()** ka use file pointer ko move karne ke liye hota hai.
2. **tell()** se current pointer ki position pata chalti hai.
3. Default **whence** value **0** hoti hai, jo file ke start ko refer karta hai.
4. Pointer ko file ke data se aage le jaane par koi error nahi hoga, but content read nahi hoga.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

File-Based To-Do List Program

Ek simple program banaye jisme user apne tasks ko add, view, aur delete kar sakta hai. Ye tasks file mein save honge, taki program band hone ke baad bhi data safe rahe.

Steps:

- Program user ko **menu** options de:
  - **Add a new task**: Naya task add kare.
  - **View all tasks**: Saare tasks dikhaye.
  - **Delete a task by number**: Task number ke zariye delete kare.
  - **Exit the program**: Program ko band kare.
- File Handling:
  - Tasks ek file (**todo.txt**) mein store kare.
  - Jab program shuru ho, tab file se tasks read kare.
  - Jab user koi action kare (add ya delete), tab file ko update kare.

Lesson complete—great job!

Lesson 9: Object-Oriented Programming

Classes and Objects

Python **classes** aur **objects** object-oriented programming (**OOP**) ka foundation hain. Iska use large aur complex code ko manageable aur reusable banane ke liye hota hai. Chaliye is concept ko simple aur asaan tareeqy say samajhte hain.

Classes

Ek **class** ek blueprint ya template hai jisme aap define karte ho ki ek object kaisa hoga aur usme kya properties aur behaviors (attributes aur methods) honge.

Syntax:

```
class ClassName:
    # Class attributes and methods defined here
    pass
```

Objects

Ek **object** class ka ek real-world instance hai. Jab aap ek class ka use karte ho to ek object banate ho, jo class ke defined attributes aur methods ko follow karta hai.

Syntax for Object:

```
object_name = ClassName()
```

Example: Real-Life Analogy

Imagine karo ek class "**Car**" hai. Isme aap define karte ho ki har car ka ek color, model, aur speed hogi, aur wo drive aur stop kar sakti hai.
Agar aap ek object banao is class ka, jaise "**Honda**" ya "**Toyota**", to wo sab class ke rules follow karte hain, lekin unke attributes alag-alag ho sakte hain.

Python Example

Creating a Class and Object:

```python
# Define a class
class Car:
    # Constructor to initialize properties
    def __init__(self, brand, color):
        self.brand = brand  # Object attribute
        self.color = color  # Object attribute

    # Method to display car details
    def show_details(self):
        print(f"Car Brand: {self.brand}, Color: {self.color}")

# Create an object of the class
my_car = Car("Toyota", "Red")

# Access object attributes
print(my_car.brand)  # Output: Toyota
print(my_car.color)  # Output: Red

# Call object method
my_car.show_details()  # Output: Car Brand: Toyota, Color: Red
```

Key Points About Classes and Objects

1. **Attributes**:
   - These are variables that store data related to the object.
   - In the example, **brand** and **color** are attributes.
2. **Methods**:
   - These are functions defined inside a class that describe the behavior of objects.
   - In the example, **show_details()** is a method.
3. **Constructor** (**__init__**):
   - A special method used to initialize an object when it’s created.
   - Automatically called when an object is made.

Why Use Classes and Objects?

1. **Reusability**: Ek class ko multiple objects ke liye use kiya ja sakta hai.
2. **Organization**: Code ko structured aur readable banata hai.
3. **Encapsulation**: Data aur methods ko ek saath rakhna.
4. **Scalability**: Large programs mein classes aur objects ka use code ko scalable banata hai.

Summary:

- **Class**: A template for creating objects.
- **Object**: An instance of a class with real values.
- Use **__init__()** to initialize objects.
- Methods define the behaviors of objects.

Real-world analogy se classes aur objects ko samajhna easy ho jata hai. Jaise har car ek **object** hai jo ek **Car class** ke rules follow karta hai.

The full lesson is now available on the Needi Developer YouTube channel.

Constructors and Instance Methods

Python mein **constructors**(**__init__**) aur **instance** methods object-oriented programming (**OOP**) ka important part hain. Ye dono concepts aapko classes aur objects ko efficiently use karne mein madad karte hain.

Constructors (**__init__**)

Constructor ek special method hota hai jo class ka object banate waqt automatically call hota hai. Iska main kaam object ko initialize karna (initialize karte waqt values assign karna) hota hai.

- **__init__** method ko constructor kaha jata hai.
- Jab aap **class** ka **object** banate hain, ye method automatically call hota hai.

Syntax:

```
class ClassName:
    def __init__(self, param1, param2):
        self.attribute1 = param1
        self.attribute2 = param2
```

Explanation:

- **self**: Ye ek reference hai jo object ko represent karta hai. Jab bhi aap class ke andar kisi attribute ko access karte hain, self ka use lazmi hota hai.
- **__init__** method: Jab aap object create karte hain, ye method automatically call hota hai aur isme diya gaya data object ke attributes mein store ho jata hai.

Example:

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand  # Object ka attribute
        self.color = color  # Object ka attribute

# Object create karte waqt constructor ko call kiya jaata hai
my_car = Car("Honda", "Blue")

print(my_car.brand)  # Output: Honda
print(my_car.color)  # Output: Blue
```

Explanation:

- **__init__()** method ne **brand** aur **color** ko initialize kiya, jab **my_car** ka object bana.
- Jab hum **my_car.brand** ya **my_car.color** ko access karte hain, to wo object ke attributes se value le leta hai jo constructor ne set ki thi.

Instance Methods

Instance methods wo methods hain jo object ke specific data (attributes) ko modify ya access karne ke liye use kiye jate hain. Ye methods class ke objects ke saath interact karte hain.

- **self** parameter instance method ka hissa hota hai, jo current object ko refer karta hai.
- Ye methods class ke objects ko modify ya unke saath operations perform karte hain.

Syntax:

```
class ClassName:
    def instance_method(self):
        # Method ka body
        pass
```

Example:

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

    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Object banate hain
person1 = Person("Naveed", 20)

# Instance method call karte hain
person1.greet()
 # Output: Hello, my name is Naveed and I am 20 years old.
```

Explanation:

- **__init__** method ne **name** aur **age** ko initialize kiya.
- **greet** method ek instance method hai jo object ke attributes (**name** aur **age**) ke saath kaam karta hai aur user ko greet karta hai.

Instance Methods Example with Additional Attributes

Agar aapko object ke state ko modify karna ho, to instance methods ka use karte hain.

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrew {amount}. Remaining balance: {self.balance}")
        else:
            print("Insufficient balance!")

# Object banate hain
account = BankAccount("Naveed", 1000)

# Instance methods ko call karte hain
account.deposit(500)
# Output: Deposited 500. New balance: 1500
account.withdraw(200)
# Output: Withdrew 200. Remaining balance: 1300
account.withdraw(1500)
# Output: Insufficient balance!
```

Explanation:

- **deposit** aur **withdraw** methods instance methods hain, jo **self.balance** ko modify karte hain.
- Ye methods object ki state ko modify karte hain, jaise balance ko update karna.

Key Points:

1. **__init__ Constructor**:
   - Class ka object banate waqt automatically call hota hai.
   - Object ko initialize karta hai, attributes ko set karta hai.
2. **Instance Methods**:
   - Object ke attributes ke saath interact karte hain.
   - **self** parameter ko use karke object ke state ko modify karte hain.

Summary:

- Constructors (**__init__**) object create karte waqt data ko initialize karne ke liye use hote hain.
- **Instance Methods** object ke attributes ke saath kaam karte hain aur unko modify karte hain.

Is tarah se aap classes aur objects ko effectively use kar sakte hain apne programs ko organize aur reusable banane ke liye.

The full lesson is now available on the Needi Developer YouTube channel.

Inheritance

Inheritance ek fundamental concept hai object-oriented programming (**OOP**) ka, jo ek **class** ko doosri **class** ki properties aur methods **inherit** karne ki permission deta hai. Iska matlab hai, ek **class** doosri **class** ke **functionality** ko use kar sakti hai bina usse dobara likhne ke.

What is Inheritance?

Inheritance ka matlab hai ki ek class (**child class**) dusri class (**parent class**) ke properties aur methods ko inherit (le leti hai). Iska fayda yeh hota hai ki hume baar-baar wohi code likhne ki zarurat nahi hoti.

Types of Inheritance

- **Single Inheritance**: Ek child class ek hi parent class se inherit karti hai.
- **Multiple Inheritance**: Ek child class do ya zyada parent classes se inherit karti hai.
- **Multilevel Inheritance**: Ek class ek aur class se inherit karti hai, aur woh class kisi teesri class se inherit karti hai.

Ghabrany ki zaroort nahi hai chaliye es concept ko sab sy asaan tareeqy or kuch real-life examples ky zariye clear karty hai 😊

Single Inheritance

Ek parent class ka saara code ek child class mein reuse hota hai.

Example:

```
class Animal:
    def speak(self):
        print("Animals make sounds.")

# Child Class
class Dog(Animal):  # Dog inherits from Animal
    def bark(self):
        print("Woof! Woof!")

# Create object of child class
dog = Dog()
dog.speak()  # Output: Animals make sounds..
dog.bark()   # Output: Woof! Woof!
```

Explanation:

- **Animal** parent class hai, aur **Dog** child class hai.
- **Dog** class ne **Animal** class ke method **speak()** ko inherit kiya.
- **Dog** class apne method **bark()** ko bhi define karti hai.

Another Example:

```
# Base Class
class Vehicle:
    def start(self):
        print("Vehicle starts.")

# Derived Class
class Car(Vehicle):
    def drive(self):
        print("Car is driving.")

# Example Usage
car = Car()
car.start()  # Output: Vehicle starts.
car.drive()  # Output: Car is driving.
```

Explanation:

- **Vehicle** ek general concept hai (parent class).
- **Car** usi ka ek specific type hai jo general functionality ke saath apni unique functionality (drive) rakhta hai.

Multiple Inheritance

Child class do ya zyada parent classes ke methods aur properties ko inherit karti hai.

Example:

```
# Parent Class 1
class Mother:
    def care(self):
        print("Mother takes care.")

# Parent Class 2
class Father:
    def protect(self):
        print("Father provides protection.")

# Child Class
class Child(Mother, Father):
    def play(self):
        print("Child loves to play!")

# Create object of child class
child = Child()
child.care()      # Output: Mother takes care.
child.protect()   # Output: Father provides protection.
child.play()      # Output: Child loves to play!
```

Explanation:

- **Mother** aur **Father** parent classes hain.
- **Child** in dono classes se inherit karta hai aur inka functionality use karta hai.

Multilevel Inheritance

Ek class ek aur class se inherit karti hai, aur woh class kisi aur class se inherit karti hai.

Example:

```
# Base Class
class Vehicle:
    def info(self):
        print("Vehicles are used for transportation.")

# Intermediate Class
class Car(Vehicle):
    def car_type(self):
        print("Car is a personal vehicle.")

# Derived Class
class SportsCar(Car):
    def speed(self):
        print("SportsCar is very fast!")

# Create object of derived class
sports_car = SportsCar()
sports_car.info()      # Output: Vehicles are used for
transportation.
sports_car.car_type()  # Output: Car is a personal vehicle.
sports_car.speed()     # Output: SportsCar is very fast!
```

Explanation:

- **Vehicle** base class hai, jo basic properties provide karti hai.
- **Car** se **Vehicle** inherit karta hai, aur uske saath apna method add karta hai.
- **SportsCar** se **Car** inherit karta hai, aur further apne features add karta hai.

Chaliye **Multilevel Inheritance** ki ek or example dekhty hai or es concept ko mazeed clear karty hai.

Chain of Specialization

Example: Company hierarchy, jaise ek "Employee", ek "Manager", aur ek "Senior Manager".

```python
# Base Class
class Employee:
    def work(self):
        print("Employee works.")

# Intermediate Class
class Manager(Employee):
    def manage(self):
        print("Manager oversees work.")

# Derived Class
class SeniorManager(Manager):
    def strategy(self):
        print("Senior Manager develops strategies.")

# Example Usage
senior_manager = SeniorManager()
senior_manager.work()      # Output: Employee works.
senior_manager.manage()    # Output: Manager oversees work.
senior_manager.strategy()
 # Output: Senior Manager develops strategies.
```

Explanation:

- **Employee** ka kaam hota hai kaam karna.
- **Manager** ka kaam hota hai kaam ka management.
- **Senior Manager** ka role hota hai strategy banana.

Key Features of Inheritance

- **Code Reusability**: Parent class ka code child class me reuse hota hai.
- **Extensibility**: Child class apni functionality add kar sakti hai.
- **DRY Principle**: "Don't Repeat Yourself" kaam karta hai, kyunki hume bar-bar code likhne ki zarurat nahi hoti.

Summary:

- **Single Inheritance**: Ek class ek hi parent class se inherit karti hai.
- **Multiple Inheritance**: Ek class do ya zyada parent classes se inherit karti hai.
- **Multilevel Inheritance**: Ek class ek aur class se inherit karti hai jo kisi teesri class se inherit karti hai.

Inheritance ka use karke aap apne programs ko modular aur efficient bana sakte hain.

The full lesson is now available on the Needi Developer YouTube channel.

Method Overriding

Method Overriding ka matlab hai ki agar ek derived class (child class) apni parent class ke method ko apne hisaab se redefine kare, toh parent class ka method "**override**" ho jata hai.
Yeh concept tab useful hota hai jab child class ka behavior parent class se alag ho ya specific ho.

Key Points of Method Overriding

1. **Same Method Name**: Child class ka method aur parent class ke method ka naam same hota hai.
2. **Polymorphism**: Isse hum runtime par decide kar sakte hain ki kaunsa method call karna hai (parent ya child).
3. **Super() Function**: Agar child class me redefine karte waqt bhi parent class ka method use karna ho, toh hum **super()** function ka use karte ha

Basic Method Overriding

Scenario: Ek parent class hai **Animal**, aur ek derived class hai **Dog**. Hum chahte hain ki **Dog** class ka **speak()** method alag kaam kare.

Example:

```
# Parent Class
class Animal:
    def speak(self):
        print("Animal makes a sound.")

# Child Class
class Dog(Animal):
    def speak(self):
        print("Dog barks.")

# Example Usage
animal = Animal()
dog = Dog()

animal.speak()  # Output: Animal makes a sound.
dog.speak()     # Output: Dog barks.
```

Explanation:

- Parent class ka **speak()** method ek general behavior define karta hai.
- Child class (Dog) ka **speak()** method specific behavior define karta hai, jo parent method ko override karta hai.

Using **super()** Function

Scenario: Ek **Vehicle** class hai jo ek general **start()** method define karti hai. Ek derived class **Car** apna alag kaam karegi lekin parent ka **start()** method bhi call karegi.

Example:

```
# Parent Class
class Vehicle:
    def start(self):
        print("Vehicle is starting.")

# Child Class
class Car(Vehicle):
    def start(self):
        super().start()  # Call parent class's start method
        print("Car is now ready to drive.")

# Example Usage
car = Car()
car.start()
```

Output:

```
Vehicle is starting.
Car is now ready to drive.
```

Explanation:

- Child class (**Car**) ke **start()** method ne parent ka **start()** method bhi use kiya using **super()**.

Bank Account System

Scenario: Ek **BankAccount** parent class hai jo funds deposit karne ka method provide karti hai. Ek derived class **SavingsAccount** interest add karegi aur method ko override karegi.

Example:

```
# Parent Class
class BankAccount:
    def __init__(self, balance):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: ${amount}. New balance: ${self.balance}")

# Child Class
class SavingsAccount(BankAccount):
    def deposit(self, amount):
        super().deposit(amount)  # Call parent method
        interest = amount * 0.02  # 2% interest
        self.balance += interest
        print(f"Interest added: ${interest}. New balance: ${self.balance}")

# Example Usage
account = SavingsAccount(100)
account.deposit(50)
```

Output:

```
Deposited: $50. New balance: $150
Interest added: $1.0. New balance: $151.0
```

Explanation:

- **SavingsAccount** ka deposit method parent method ko extend karta hai aur extra functionality (interest calculation) add karta hai.

Advantages of Method Overriding:

- **Specialized Behavior**: Child class apna specific behavior define kar sakti hai.
- **Code Reusability**: Parent class ke common methods reuse ho jate hain.
- **Flexibility**: Runtime par hum decide kar sakte hain ki kaunsa method call karna hai.

When to Use Method Overriding?

- Jab parent class ka behavior child class ke liye suitable na ho.
- Jab functionality extend karni ho parent class ke method ki.
- Jab runtime par alag behavior chahiye based on object type.

Summary:

Method overriding inheritance ka ek powerful feature hai jo classes ko customize karne ki flexibility deta hai. Isse hum apne program ko modular aur easy-to-maintain bana sakte hain!

The full lesson is now available on the Needi Developer YouTube channel.

Class Methods vs Static Methods

Python me methods do main types ke ho sakte hain jo ek class ke andar define kiye jaate hain:

1. **Class Methods**
2. **Static Methods**

Dono hi methods class ke andar hote hain, lekin inka purpose aur kaam alag hota hai.

Class Methods

- **Definition**: Ye methods class ke liye kaam karte hain, na ki kisi particular object ke liye.
- **How to Declare**: Class methods ko **@classmethod** decorator ke sath define karte hain.
- **First Parameter**: Ye methods apne first parameter ke taur par **cls** accept karte hain, jo ki class ko represent karta hai.
- **Use Case**:
  - Class-level data ko access ya modify karna.
  - Factory methods banane ke liye (alternative constructors).

Example:

```
class Employee:
    company_name = "Needi Developer"  # Class-level attribute

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    @classmethod
    def change_company_name(cls, new_name):
        cls.company_name = new_name

# Usage
print(Employee.company_name)  # Output: Needi Developer

Employee.change_company_name("AuraOfSurety")
print(Employee.company_name)  # Output: AuraOfSurety
```

Explanation:

- **change_company_name** ek class method hai jo **cls** use karta hai class-level attribute ko modify karne ke liye.
- Is method ka object se koi lena-dena nahi, sirf class ke attributes ko change karta hai.

Static Methods

- **Definition**: Ye methods kisi bhi object ya class-level data ko access nahi karte. Ye methods sirf class ke andar logically related operations ke liye hote hain.
- **How to Declare**: Static methods ko **@staticmethod** decorator ke sath define karte hain.
- **First Parameter**: Ye methods kisi parameter ko automatically nahi accept karte (na **self**, na **cls**).
- **Use Case**:
    - Helper ya utility functions banane ke liye jo class se related hote hain, lekin kisi object ya class attribute ko use nahi karte.

Example:

```
class MathHelper:
    @staticmethod
    def add_numbers(a, b):
        return a + b

# Usage
result = MathHelper.add_numbers(5, 10)
print(result)  # Output: 15
```

Explanation:

- **add_numbers** ek static method hai jo input parameters par kaam karta hai.
- Is method ka class ke attributes ya methods se koi lena-dena nahi.

Static Methods

Feature	Class Method	Static Method
Decorator	**@classmethod**	**@staticmethod**
First Parameter	**cls** (represents the class)	None
Access to Class Data	Yes	No
Access to Object Data	No	No
Use Case	Modify class-level data, alternative constructors	Utility or helper functions

When to Use Class Methods and Static Methods?

- **Class Methods**:
  - Jab hume class-level data ko manipulate karna ho.
  - Jab hume ek alternative constructor banana ho jo object creation ko simplify kare.
- **Static Methods**:
  - Jab hume helper ya utility functions banane ho jo class ke andar logically fit hote hain.
  - Jab kisi operation me class ya object ka involvement na ho.

Summary:

- **Class Methods**: Class-level operations ke liye.
- **Static Methods**: Utility functions ke liye.
- **Both**: Code ko organize aur readable banate hain.

The full lesson is now available on the Needi Developer YouTube channel.

Magic (Dunder) Methods

Magic methods, also called dunder methods (kyunki yeh double underscores se surrounded hote hain), Python me special methods hote hain jo apke objects ka behavior customize karte hain. **__str__** aur **__repr__** inme se common hain, aur yeh decide karte hain ke object ko string me kaise represent kiya jaye.

__repr__ Method

- **Purpose**: Yeh object ki official ya "formal" string representation provide karta hai. Mainly debugging aur development ke liye use hota hai.
- **Kab Call Hota Hai**: Jab aap **repr(object)** use karte ho ya object ko Python shell me directly print karte ho.
- **Output Kaisa Hona Chahiye**: Output clear aur unambiguous hona chahiye, aur ideally aap **eval()** se object ko recreate kar sako.

Example:

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

    def __repr__(self):
        return f"Person(name='{self.name}', age={self.age})"

# Usage
person = Person("Naveed", 20)
print(repr(person))  # Output: Person(name='Naveed', age=20)
```

Explanation:

- **__repr__** ek formal description deta hai jo debugging ke liye helpful hoti hai.

__str__ Method

- **Purpose**: Yeh object ki readable aur "informal" string representation deta hai. End-users ke liye information display karne ke liye use hota hai.
- **Kab Call Hota Hai**: Jab aap **str(object)** ya **print(object)** use karte ho.
- **Output Kaisa Hona Chahiye**: Output user-friendly aur readable hona chahiye.

Example:

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

    def __str__(self):
        return f"{self.name} is {self.age} years old."

# Usage
person = Person("Naveed", 20)
print(person)  # Output: Naveed is 20 years old.
```

Explanation:

- **__str__** ek concise aur readable description deta hai jo end-users ke liye bana hota hai.

Difference between __repr__ and __str__

Feature	__repr__	__str__
Purpose	Formal aur unambiguous	Informal aur readable
Audience	Developers (debugging)	End-users (display)
Call Method	**repr(object)** ya Python shell	**str(object)** ya **print(object)**
Fallback	Agar **__str__** define na ho to yeh use hota hai	Define ho to yeh use nahi hota

Combined Example

Agar dono methods ek class me ho to kaise kaam karte hain:

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

    def __repr__(self):
        return f"Product(name='{self.name}', price={self.price})"

    def __str__(self):
        return f"{self.name}: ${self.price}"

# Usage
product = Product("Laptop", 1500)
print(repr(product))
# Output: Product(name='Laptop', price=1500)
print(product)       # Output: Laptop: $1500
```

Explanation:

- **__repr__**: Developers ke liye detailed description.
- **__str__**: Users ke liye readable output.

Summary

- **__repr__**: Formal aur detailed output (useful for developers).
- **__str__**: Friendly aur readable output (useful for end-users).
- **Fallback**: Agar **__str__** define na ho, to Python **__repr__** ko use karega.

The full lesson is now available on the Needi Developer YouTube channel.

Operator Overloading

Operator overloading ka matlab hai kisi existing operator ka behavior customize karna jab wo user-defined objects ke sath use hota hai. Iska matlab hai ki aap Python ke built-in operators (like **+**, **-**, *****, etc.) ko apne custom classes ke objects ke liye define kar sakte ho.

Why Use Operator Overloading?

1. **Readable Code**: Aap complex functionality ko readable aur concise syntax ke through implement kar sakte ho.
2. **Real-Life Simulations**: Custom objects ko natural aur intuitive way me manipulate karne ki flexibility milti hai.
3. **Customization**: Objects ke behavior ko apne requirement ke mutabiq change kar sakte ho.

How Does It Work?

Operator overloading karne ke liye aapko special methods (dunder/magic methods) use karne hote hain. Har operator ke liye ek corresponding magic method hota hai.

Operator	Method	Example
+	__add__	obj1 + obj2
-	__sub__	obj1 - obj2
*	__mul__	obj1 * obj2
/	__truediv__	obj1 / obj2
==	__eq__	obj1 == obj2
<	__lt__	obj1 < obj2
>	__gt__	obj1 > obj2

Overloading the + Operator

Without Overloading:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

p1 = Point(2, 3)
p2 = Point(4, 5)

# This will raise an error
result = p1 + p2
```

Without Overloading:

```
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"({self.x}, {self.y})"

# Usage
p1 = Point(2, 3)
p2 = Point(4, 5)
result = p1 + p2
print(result)  # Output: (6, 8)
```

Explanation:

- **__add__** method ko redefine karke humne **+** operator ka behavior customize kiya.
- Do **Point** objects add hone par unke **x** aur **y** coordinates ko add karke ek naya Point return hota hai.

Other Examples

Overloading * for Custom Multiplication:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __mul__(self, scalar):
        return Point(self.x * scalar, self.y * scalar)

    def __str__(self):
        return f"({self.x}, {self.y})"

# Usage
p1 = Point(2, 3)
result = p1 * 3
print(result)  # Output: (6, 9)
```

Overloading Comparison Operators (<, >):

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

    def __lt__(self, other):
        return self.age < other.age

# Usage
person1 = Person("Naveed", 20)
person2 = Person("Waqas", 15)

print(person1 < person2)  # Output: False
```

Explanation:

- **__lt__** method define karke age ke basis par comparison ka behavior customize kiya gaya.

Key Points to Remember

1. Operator overloading se readability aur functionality dono improve hoti hain.
2. Har operator ke liye ek specific magic method hota hai.
3. Overloading ka misuse na karein; operations ka behavior logical aur intuitive hona chahiye.

Summary

Operator overloading Python me ek powerful feature hai jo user-defined objects ke behavior ko natural aur readable banata hai. Iska proper use complex tasks ko simplify kar sakta hai aur code ko zyada maintainable banata hai.

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

Build a Simple Banking System

Create a banking system using object-oriented programming in Python. The system should allow a user to perform the following actions:

1. **Check Balance**: Display the current account balance.
2. **Deposit Money**: Add money to the account.
3. **Withdraw Money**: Deduct money from the account if the balance is sufficient.
4. **Exit**: Close the application.

Requirements:

- Use a class **BankAccount** to represent the bank account.
- Implement methods for each operation (**check_balance**, **deposit**, **withdraw**).
- Use a simple loop to allow the user to perform multiple actions until they choose to exit.

Lesson complete—great job!

Lesson 10: Python Modules

Modules

Modules Python ke ready-made tools hote hain jo aapke kaam ko easy banate hain. Yeh ek tarah ke pre-written code libraries hote hain jinko aap apne programs me directly use kar sakte ho. Iska fayda yeh hota hai ki aapko sab kuch scratch se likhne ki zarurat nahi padti.

Why Use Modules?

- **Code Reusability**: Ek baar likha code baar-baar alag programs me use kar sakte ho.
- **Time Saving**: Ready-made solutions milte hain, to time bachta hai.
- **Organized Code**: Modules use karne se code clean aur manage karne me easy hota hai.
- **Specialized Tools**: Har module ek specific kaam ke liye hota hai, jaise math calculations, file handling, ya random numbers generate karna.

How to use Modules?

Module ko use karne ke liye import keyword ka use karte hain.

Example:

```
import module_name
```

Examples of Python's Built-in Modules

1. os Module

os module ka use operating system ke sath kaam karne ke liye hota hai.

Kahan Use Hota Hai?

- Current directory check karne me
- Files aur folders create/delete karne me

Example:

```
import os

# Current directory ka naam
print("Current Directory:", os.getcwd())

# Naya folder banaiye
os.mkdir("NewFolder")
```

2. math Module

math module advanced mathematical functions provide karta hai, jaise square root, power calculation, etc.

Kahan Use Hota Hai?

- Complex calculations ke liye
- Rounding numbers ke liye

Example:

```
import math

# Square root nikalna
print("Square root of 16:", math.sqrt(16))

# Pi ki value dekhna
print("Value of Pi:", math.pi)
```

3. random Module

random module ka use random numbers ya random choices generate karne ke liye hota hai.

Kahan Use Hota Hai?

- Games banane me
- Test data banane ke liye

Example:

```
import random

# 1 se 10 ke beech ek random number generate kare
print("Random number:", random.randint(1, 10))

# Ek random item choose kare list me se
choices = ["Apple", "Banana", "Cherry"]
print("Random choice:", random.choice(choices))
```

Benefits of Modules

1. Development time save hota hai.
2. Code easy aur optimized hota hai.
3. Programs ko samajhna aur maintain karna easy hota hai.

os, **math**, aur **random** jaise modules aapko powerful aur efficient programs banane me help karte hain, bina zyada effort ke!

The full lesson is now available on the Needi Developer YouTube channel.

if __name__ == "__main__"

Python me **if __name__ == "__main__"** ek special construct hai jo aapke program ka behavior control karne ke liye use hota hai, jab aapka Python file directly run hota hai ya kisi aur file me import hota hai.

What is its basic function?

- **Direct Execution ke liye Code Chalana**:
  Jab file ko directly execute kiya jata hai (**python myfile.py**), tab iska code chalega.
- **Import ke Samay Code Avoid Karna**:
  Jab file ko kisi aur file me import kiya jata hai, to **if __name__ == "__main__"** ke andar ka code nahi chalega.

What is **__name__**?

- Har Python file ka ek special variable hota hai **__name__**.
- Jab file directly chalti hai, tab **__name__** ki value **"__main__"** hoti hai.
- Agar file import hoti hai, to **__name__** us file ke naam ke barabar hoti hai (**"myfile"**).

How its Work?

1: File Directly Run Ho Rahi Hai

Example:

```
# myfile.py
def greet():
    print("Hello from myfile!")

if __name__ == "__main__":
    print("This file is running directly.")
    greet()
```

Output (when running python **myfile.py**):

```
This file is running directly.
Hello from myfile!
```

2: File Import Ho Rahi Hai

Example:

```python
# main.py
import myfile

print("This is main.py.")
```

Output (when running **python main.py**):

```
This is main.py.
```

Yahan par **myfile** ka code jo **if __name__ == "__main__"** ke andar hai, nahi chalega.

Why Use This?

- **Code Reusability**:
  Aap ek file ka code import karte ho bina uske execution code ko chalaye.
- **Testing**:
  File ke andar kuch functions aur logic test karne ke liye use hota hai, jab wo file akeli chal rahi ho.
- **Cleaner Structure**:
  Isse program ka flow clear hota hai, aur unnecessary code execution avoid hota hai.

Summary

if __name__ == "__main__" ek best practice hai jo ensure karta hai ki:

- File ka kuch code tabhi chale jab wo file directly execute ho.
- Import ke samay unnecessary code execute na ho.

The full lesson is now available on the Needi Developer YouTube channel.

Installing Third-Party Libraries with pip

Python ka **pip** ek package manager hai jo aapko third-party libraries aur tools install karne ki facility deta hai. Ye libraries aapke projects ke kaam ko asaan aur efficient banate hain.

What is pip?

- **pip** ka full form hai: "**Pip Installs Packages**".
- Ye ek command-line tool hai jo **Python Package Index** (**PyPI**) se libraries download aur install karta hai.
- Iske through aap Python ki built-in functionality ko extend kar sakte ho

How to Use pip

1. pip Ko Check Karna (Installed Hai Ya Nahi):

Pehle ensure karein ki pip installed hai:

```
pip --version
```

Output kuch is tarah ka hona chahiye:

```
pip 21.3.1 from ... (python 3.x)
```

Agar pip install nahi hai, to aap is command se install kar sakte ho:

```
python -m ensurepip --upgrade
```

2. Third-Party Library Install Karna:

Koi library install karne ke liye command:

```
pip install <library-name>
```

Example: NumPy library install karna:

```
pip install numpy
```

Output:

```
Collecting numpy
Downloading numpy-1.23.3-cp39-cp39-win_amd64.whl (14 MB)
Installing collected packages: numpy
Successfully installed numpy-1.23.3
```

3. Library Upgrade Karna:

Aap existing library ka latest version install kar sakte ho:

```
pip install --upgrade <library-name>
```

Example: NumPy library install karna:

```
pip install --upgrade numpy
```

4. Library Uninstall Karna:

Agar kisi library ki zarurat nahi hai, to aap ise uninstall kar sakte ho:

```
pip uninstall <library-name>
```

Example:

```
pip uninstall numpy
```

How to check if a library is installed?

Aap installed libraries ki list dekhne ke liye ye command use kar sakte ho:

```
pip list
```

Output:

```
Package    Version
---------- -------
numpy      1.23.3
requests   2.28.1
```

Requirements File

Agar aap ek project ke liye multiple libraries install karna chahte ho, to unhe ek file me list kar sakte ho:

1. Create a file named **requirements.txt**:

```
numpy
requests
pandas
```

2. Install all libraries in one command:

```
pip install -r requirements.txt
```

Why Use pip?

- **Easy Access to Libraries**: Thousands of pre-built libraries PyPI par available hain.
- **Time-Saving**: Complex tasks ke liye already optimized solutions use kar sakte hain.
- **Project Management**: Virtual environments ke saath pip libraries ko alag-alag projects ke liye manage kar sakta hai.

Summary

- pip ek powerful tool hai jo Python developers ka kaam asaan banata hai.
- Aap libraries ko install, update, aur uninstall kar sakte ho.
- Libraries ko manage karne ke liye **requirements.txt** ka use karo.

The full lesson is now available on the Needi Developer YouTube channel.

Virtual Environments (venv)

Python ke **virtual environments (venv)** ek tarah ke isolated workspace hote hain jisme aap apne project-specific dependencies (libraries, packages, etc.) manage kar sakte ho bina system-wide Python installation ko affect kiye.

Why is a Virtual Environment necessary?

- **Dependency Conflicts Avoid Karna**
  Agar ek project me ek package ki specific version chahiye aur dusre project me usi package ki different version chahiye, to virtual environment yeh problem solve karta hai.
- **Clean Project Setup**
  Har project ka apna environment hota hai, jo dependencies aur configurations ko isolated rakhta hai.
- **Easy Deployment**
  Virtual environment me sirf wahi packages hote hain jo project ko chahiye, isliye deployment smooth hota hai.

How to create Virtual Environment

1. Create Virtual Environment

Command:

```
python -m venv myenv
```

- **python**: System Python interpreter.
- **-m venv**: Module to create virtual environment.
- **myenv**: Virtual environment ka naam.

2. Activate Virtual Environment

Windows:

```
myenv\Scripts\activate
```

Mac or **Linux:**

```
source myenv/bin/activate
```

Aapko terminal ke left side me **(myenv)** likha milega, jo batata hai ki environment active hai.

3. Deactivate Virtual Environment

Command:

```
deactivate
```

Yeh system-wide Python me wapas le aata hai.

Installing Packages in Virtual Environment

1. Packages Install Karna

Virtual environment activate karne ke baad normal tarike se install karte hain:

```
pip install package_name
```

2. Packages Ki List Dekhna

```
pip list
```

3. Requirements File Banana

Jo bhi packages install hain unko list karne ke liye:

```
pip freeze > requirements.txt
```

4. Requirements File Se Install Karna

```
pip install -r requirements.txt
```

Benefits of Virtual Environment

1. Alag-alag projects ke liye alag-alag environments ban jate hain.
2. System-wide Python environment ko clean aur stable rakhta hai.
3. Deployment aur collaboration easy banata hai.

Pro Tip: Hamesha virtual environment ka use karo jab bhi aap kisi naye project pe kaam shuru karo!

The full lesson is now available on the Needi Developer YouTube channel.

Exercise

File Management Script

Write a script that uses the **os**, **math**, and **random** modules to perform the following tasks:

- **Create a Directory**:
  - Use the **os** module to create a directory named **RandomFiles** in the current working directory.
- **Generate Random Files**:
  - Use the **random** module to generate 5 random filenames.
  - Each filename should be a random number between 1000 and 9999 with a **.txt** extension.
  - Example filenames: **1054.txt**, **8762.txt**.
- **Write Data to Files**:
  - For each file, generate a random number and calculate its square root using the **math** module.
  - Write the random number and its square root into the file in this format:

```
Random Number: 25
Square Root: 5.0
```

- **Display the Created Files**:
  - List all the files created in the **RandomFiles** directory using the **os** module.

Lesson complete—great job!

Lesson 11: Advanced Python Concepts

Introduction to Generators and Iterators

Python me generators aur iterators kaafi powerful tools hain jo hum data ko efficiently handle karne ke liye use karte hain, especially jab large datasets ke saath kaam karte hain ya data ko dynamically generate karna ho.

What is Iterators?

Iterator ek aisi object hoti hai jo kisi collection (like list, tuple, ya string) ke elements ko ek-ek karke access karne me madad karti hai.

Iterator kaise kaam karta hai?

- Iterator ke paas do important methods hote hain:
  - **__iter__()** – Jo iterator object ko return karta hai.
  - **__next__()** – Jo sequence ka agla element return karta hai. Agar sequence khatam ho jaye, toh **StopIteration** exception raise hota hai.

Example:

```
numbers = [10, 20, 30]
iterator = iter(numbers)  # Iterator object banaya

print(next(iterator))  # Output: 10
print(next(iterator))  # Output: 20
print(next(iterator))  # Output: 30
# print(next(iterator))  # StopIteration error raise karega
```

Yahan hum list ke elements ko ek-ek karke access karte hain.

What is Generators?

Generator ek special type ka iterator hai jo function aur **yield** keyword ke zariye banaya jata hai.

- Ye saare data ek saath return karne ke bajaye, **ek-ek karke on-demand** generate karta hai.
- Generators memory-efficient hote hain, kyunki ye saara data memory me store nahi karte.

Generator kaise kaam karta hai?

- **yield** keyword use karke generator function se value return ki jati hai.
- Jab generator ke **__next__()** method ko call karte hain, execution wahi se resume hoti hai jahan se last **yield** hua tha.

Example:

```
def count_up_to(n):
    count = 1
    while count <= n:
        yield count  # Current value ko pause karke return karega
        count += 1

# Generator use karna
counter = count_up_to(5)
print(next(counter))  # Output: 1
print(next(counter))  # Output: 2
print(next(counter))  # Output: 3
```

What is Generators?

Feature	Iterator	Generator
Definition	Iterator ek object hota hai.	Generator ek function ke zariye banta hai.
Keyword	Manually **__iter__()** aur **__next__()** likhna padta hai.	yield keyword use hota hai.
Memory Usage	Sequence ko memory me store karta hai.	Data on-the-fly generate karta hai.
Efficiency	Less efficient for large datasets.	More efficient for large datasets.

Summary

- **Iterator** predefined objects hote hain jo sequence ko traverse karte hain.
- **Generator** ek tarika hai efficiently data generate karne ka, jab poore dataset ki zarurat na ho.

Dono hi Python ke data processing ko faster aur memory-efficient banate hain!

The full lesson is now available on the Needi Developer YouTube channel.

Map, Filter, and Reduce Function

Python me **map**, **filter**, aur **reduce** functions kaafi powerful tools hain jo functional programming concepts ke saath kaam karte hain. Ye functions aapko data processing me madad karte hain, aur operations ko short aur readable banate hain.

1. **map()** Function

map() ka kaam hai ek given function ko kisi sequence ke har element par apply karna aur ek nayi sequence return karna.

Syntax:

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

- **function**: Aapka function jo har element par apply hoga.
- **iterable**: Aapki list, tuple, ya koi aur sequence.

Example:

```
# Squaring each number in a list
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x**2, numbers)
print(list(squared))  # Output: [1, 4, 9, 16, 25]
```

Yahan har number ko square kar diya gaya.

Lambda function bnany ka ek tareeqa hai esko next chapter me details me seekhy gy.

2. **filter()** Function

filter() ka kaam hai ek given condition ke basis par sequence ke elements ko filter karna. Ye sirf un elements ko return karta hai jo condition ko satisfy karte hain.

Syntax:

```
filter(function, iterable)
```

- **function**: Ek condition-checking function jo **True** ya **False** return karega.
- **iterable**: Sequence jisme filter lagana hai.

Example:

```
# Filter odd numbers from a list
numbers = [1, 2, 3, 4, 5, 6]
odd_numbers = filter(lambda x: x % 2 != 0, numbers)
print(list(odd_numbers))  # Output: [1, 3, 5]
```

Yahan sirf odd numbers ko filter kiya gaya.

3. **reduce()** Function

reduce() function cumulative operation perform karta hai, jaise sum, multiplication, ya concatenation. Ye ek iterable ke saare elements ko ek single value me reduce kar deta hai.

Syntax:

```
reduce(function, iterable)
```

- **function**: Ek function jo do arguments leta hai aur cumulative result deta hai.
- **iterable**: Sequence jisme operation karna hai.

Example:

```
from functools import reduce

# Multiply all numbers in a list
numbers = [1, 2, 3, 4]
result = reduce(lambda x, y: x * y, numbers)
print(result)  # Output: 24
```

Yahan saare numbers ko multiply karke ek hi value me reduce kar diya.

Comparison of **map**, **filter**, and **reduce**:

Function	Purpose	Output
map	Apply function to all elements in a sequence.	Transformed sequence.
filter	Select elements based on a condition.	Filtered sequence.
reduce	Combine all elements into a single result.	Single cumulative value.

Why Use These Functions?

- **Shorter Code**: Ek line me kaam ho jata hai jo multiple lines me hota.
- **Readability**: Code zyada clean aur understandable hota hai.
- **Efficiency**: Functional programming ka benefit milta hai.

Summary

- **map()**: Apply a function to all elements.
- **filter()**: Select elements based on a condition.
- **reduce()**: Combine all elements into one result.

Ye functions aapko Python me data processing tasks ko simplify karne me madad karte hain!

The full lesson is now available on the Needi Developer YouTube channel.

Lambda Functions

Python mein **lambda functions** ek short aur concise way hai functions likhne ka. Inko anonymous functions bhi kehte hain kyunki ye bina kisi naam ke hoti hain. Lambda functions ek single line mein likhe jaate hain aur yeh tab use hote hain jab hume chhoti aur temporary functionality chahiye hoti hai.

Syntax:

```
lambda arguments: expression
```

- **arguments**: Input parameters, jaise normal functions mein hote hain.
- **expression**: Ek single line ka expression jo evaluate hoke result return karta hai.

Example:

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

Key Features of Lambda Functions

1. **Single Expression**: Lambda functions ek hi line ka expression handle karte hain.
2. **No Name**: Ye anonymous hote hain, yani inka koi naam nahi hota unless aap inko kisi variable mein store karein.
3. **Temporary Use**: Ye functions chhoti aur one-time functionality ke liye useful hain.
4. **Inline Use**: Ye functions generally inline use hote hain, jaise higher-order functions ke saath.

Why Use Lambda Functions?

- Jab aapko ek simple functionality likhni ho bina ek pura function define kiye.
- Jab ek hi line ka kaam ho aur function ko reuse karne ki zarurat na ho.
- Functional programming mein (like map, filter, reduce ke saath) kaam karne ke liye.

Examples of Lambda Functions

1. Simple Lambda Function

```
add = lambda a, b: a + b
print(add(2, 3))  # Output: 5
```

2. Using Lambda with map()

map() function ka use ek list ke har element par operation karne ke liye hota hai.

```
numbers = [1, 2, 3, 4]
squares = list(map(lambda x: x * x, numbers))
print(squares)  # Output: [1, 4, 9, 16]
```

3. Using Lambda with filter()

filter() function ka use ek list ke elements ko filter karne ke liye hota hai.

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4, 6]
```

4. Using Lambda with sorted()

sorted() function ka use custom sorting karne ke liye hota hai.

```
students = [("Naveed", 90), ("Sara", 85), ("Fatima", 95)]
sorted_students = sorted(students, key=lambda x: x[1])
print(sorted_students)
# Output: [('Sara', 85), ('Naveed', 90), ('Fatima', 95)]
```

Limitations of Lambda Functions

1. **Single Expression Only**: Lambda functions ek se zyada lines handle nahi karte.
2. **No Name**: Inka naam nahi hota, isliye debugging mushkil ho sakti hai.
3. **Not for Complex Logic**: Ye sirf simple operations ke liye suitable hain.

When to Use Lambda Functions?

1. Jab aapko ek chhoti si functionality likhni ho.
2. Jab aap temporary use ke liye ek function create karna chahte hain.
3. Functional programming methods jaise **map**, **filter**, aur **reduce** ke saath kaam karte waqt.

The full lesson is now available on the Needi Developer YouTube channel.

Walrus Operator (:=)

Walrus Operator ka asal naam **Assignment Expression Operator** hai, aur ye Python 3.8 me introduce hua tha. Iska symbol **:=** hai, jo aapko ek variable me value assign karte waqt uska use karne ki facility deta hai.
Iska naam Walrus Operator isliye rakha gaya kyunki iska symbol (:=) ek walrus ke face jaisa lagta hai. 🦭

Purpose of Walrus Operator

- **Reduces Code Length**: Aap ek hi expression me value assign aur uska use kar sakte hain.
- **Improves Readability**: Bar-bar alag line me value assign karne ki zarurat nahi hoti.

Syntax:

```
variable := expression
```

Normal Assignment vs Walrus Operator

Without Walrus Operator:

```
data = input("Enter your name: ")
if len(data) > 5:
    print(f"Your name, {data}, is long!")
```

With Walrus Operator:

```
if (data := input("Enter your name: ")) and len(data) > 5:
    print(f"Your name, {data}, is long!")
```

Where to Use the Walrus Operator

1. In Loops

Walrus Operator loops me kaafi useful hai, jab aap ek hi variable me repeatedly data assign aur check karte hain.

Without Walrus Operator:

```
numbers = [1, 2, 3, 4, 5]
while len(numbers) > 0:
    current = numbers.pop()
    print(f"Processing: {current}")
```

With Walrus Operator:

```
numbers = [1, 2, 3, 4, 5]
while (current := numbers.pop()):
    print(f"Processing: {current}")
```

2. In Conditional Statements

Conditions check karte waqt bar-bar variable assign karne ki zarurat nahi hoti.

Without Walrus Operator:

```
data = input("Enter something: ")
if len(data) > 3:
    print(f"Length is greater than 3: {data}")
```

With Walrus Operator:

```
if (data := input("Enter something: ")) and len(data) > 3:
    print(f"Length is greater than 3: {data}")
```

3. In List Comprehensions

List comprehensions me efficiently data assign aur use karna possible hota hai.

Without Walrus Operator:

```
results = []
for x in range(10):
    square = x**2
    if square > 20:
        results.append(square)
print(results)
```

With Walrus Operator:

```
results = [square for x in range(10) if (square := x**2) > 20]
print(results)
```

Purpose of Walrus Operator

- **Assignment (=)**: Ek variable me value assign karta hai, aur usse use karne ke liye dusri line me likhna padta hai.
- **Walrus (:=)**: Ek hi line me value assign aur use karne ki flexibility deta hai.

Key Points to Remember

- **Requires Python 3.8 or Newer**: Ye feature sirf Python 3.8 aur uske baad wale versions me kaam karta hai.
- **Not for Regular Assignments**: Iska use sirf tab karein jab aapko ek hi line me value assign aur use karni ho.

Why Use Walrus Operator?

- Reduces redundancy in code.
- Makes complex conditions simpler and easier to read.
- Helpful in loops and comprehensions.

Walrus Operator aapke code ko concise aur efficient banata hai, lekin iska use tab karein jab zarurat ho aur readability kharaab na ho.

The full lesson is now available on the Needi Developer YouTube channel.

Mega Knight Projects

1. Student Management System

A **Student Management System** is a comprehensive project that combines almost all the concepts from your Python course, providing a real-world scenario to test your skills.

Project Description

You will create a command-line Student Management System where users (teachers) can:

1. Add students to a database.
2. View student details.
3. Update student information.
4. Delete student records.
5. Search for students.
6. Generate a performance report.

The project will use file handling to save data persistently, and incorporate **functions**, **OOP**, **control flow**, **error handling**, and **data structures**.

The full project is now available on the Needi Developer YouTube channel.

2. Virtual Assistant Project

Create a Python-based virtual assistant that listens to user voice commands, processes them, and provides responses via voice and text. This assistant will utilize Python's speech recognition and text-to-speech libraries to interact with the user.

Key Features

1. **Greeting the User**:
    - The assistant greets the user with a voice response based on the time of day (morning, afternoon, evening).
2. **Voice Command Recognition**:
    - The assistant will understand voice commands and convert them into text using the **speech_recognition** module.
3. **Responding via Voice**:
    - The assistant replies to the user with a voice response using **pyttsx3** or **gTTS**.
4. **Performing Tasks**:
    - Open basic system applications (like Notepad or Calculator).
    - Fetch current time and say it.
    - Tell jokes from a predefined list.
    - Search for specific files or folders in a directory.
5. **Handling Errors**:
    - Gracefully handles invalid commands or voice recognition issues with a user-friendly response.

The full project is now available on the Needi Developer YouTube channel.

3. Inventory Management System

Create a Python-based Inventory Management System that allows users to manage product stock, track sales, and generate reports.

Key Features

1. **Product Management**:
    - Add, update, or delete product details (name, price, quantity, etc.).
2. **Stock Management**:
    - Check available stock for products.
    - Update stock levels after a sale or purchase.
3. **Sales Tracking**:
    - Record sales transactions (product, quantity sold, total price).
    - Calculate and display total revenue.
4. **Reports**:
    - Generate reports showing:
        - Products running low on stock.
        - Total sales and revenue.
5. **User-Friendly Interface**:
    - Use command-line interaction to display options and collect user input.

The full project is now available on the Needi Developer YouTube channel.

---

4. Personal Expense Tracker

Create a Python-based application to track and analyze personal expenses.

Key Features

1. **Expense Management**:
    - Add, update, and delete expense entries (date, category, amount, description).
2. **Category-Based Analysis**:
    - Categorize expenses (Food, Travel, Entertainment).
    - Display total spent per category.
3. **Monthly Summary**:
    - Provide a monthly report showing:
        - Total expenses.
        - Most spent category.
4. **Savings Calculator**:
    - Compare monthly expenses to a user-defined budget and calculate savings.
5. **Data Visualization (Optional)**:
    - Display bar charts or pie charts of expense distribution using Matplotlib.

The full project is now available on the Needi Developer YouTube channel.

Lesson complete—great job!

Advanced Tips for Becoming a Python Knight

1. **Practice Daily**:
   - Coding is like learning a new language—the more you practice, the better you get. Dedicate at least 30 minutes daily to writing Python code.
2. **Explore Real-World Projects**:
   - After completing your book, start working on real-world projects like web development (Django/Flask), data analysis (Pandas/Numpy), or automation scripts.
3. **Learn Advanced Concepts**:
   - Dive into data structures, algorithms, and design patterns. These are crucial for writing optimized and scalable code.
4. **Contribute to Open Source**:
   - Contributing to open-source projects will not only enhance your skills but also connect you with a community of developers worldwide.
5. **Keep Learning**:
   - Python evolves constantly, so keep up with new libraries and features. Follow platforms like Real Python, Stack Overflow, and GitHub.
6. **Understand Problem-Solving**:
   - Focus on breaking complex problems into smaller parts and solving them step by step.
7. **Read Code Written by Others**:
   - Explore repositories on GitHub to understand different coding styles and techniques.
8. **Build a Portfolio**:
   - Showcase your projects on platforms like GitHub, Kaggle, or LinkedIn.

Congratulations
You are now a Python Knight