POWERED BY CRESCENTON

Python Basics for Beginners

A comprehensive, beginner-friendly guide to learning Python programming from scratch. Whether you are a student, a career changer, or a curious mind, this book will take you step by step from writing your first line of code to building small real-world programs. No prior programming experience required — just curiosity and a willingness to learn.

Compiled and authored by **Syed Sajid Ul Haq**

CrescaDemy

CHAPTER 1

Introduction to Python

Pythonis one ofthemostpopularandbeginner-friendly programming languagesin the world today. Created by **Guido van Rossum** and first released in **1991**, Python was designed with a focus on code readability and simplicity. Its clean syntax, resembling plain English, makes it an ideal first language for anyone interested in programming. Python is an interpreted language, meaning code is executed line by line, which makes debugging easier for beginners.

Python is used across a wide range of fields — from web development and data science to artificial intelligence, automation, and game development. Major companies like Google, NASA, Instagram, and Netflix rely on Python for critical parts of their technology stack. Learning Python opens doors to a vast ecosystem of libraries, frameworks, and career opportunities.

Easy to Learn

Python's syntax is clean and intuitive, making it the perfect starting point for absolute beginners with no prior coding experience.

Versatile

Use Python for web development, data analysis, machine learning, automation, scripting, scientific computing, and more.

Large Community

Millions of developers worldwide contribute to Python's ecosystem with libraries, tutorials, forums, and open-source projects.

Cross-Platform

Python runs on Windows, macOS, and Linux without any changes to your code, making it highly portable and flexible.

Your First Python Program

In Python, you can display output using the `print()` function. This is traditionally the first program every programmer writes. It displays text or values on the screen. Try typing the following into your Python interpreter or editor:

```
print("Hello, World!")
print("Welcome to Python Programming!")
print(2 + 3)
print("Python is", "fun", "and", "powerful!")
```

The `print()` function can display strings (text in quotes), numbers, and even the result of calculations. You can pass multiple values separated by commas, and Python will print them with spaces in between. This simple function is your primary tool for seeing what your programs are doing.

Did You Know? Python is named after the British comedy group *Monty Python*, not the snake! Guido van Rossum was a fan of their show and wanted the language name to be short, unique, and slightly mysterious.

CHAPTER 2

Environment Setup

Before writingPython code,you needtoset upyourdevelopmentenvironment. The good news is that Python is free, open-source, and available onall majoroperating systems. Thischapter walksyouthroughinstalling Python, choosing a code editor, and running your first program on your machine.

Installing Python on Windows

1.Visit **python.org** and download the latest version (Python 3.x)
2.Run the installer and **check "Add Python to PATH"** during setup
3.Click "Install Now" and wait for completion
4.Open Command Prompt and type `python --version` to verify
5. Launch IDLE (Python's built-in editor) or use VS Code

Installing Python on macOS / Linux

1.macOS: Python 3 may already be installed. Check with `python3 --version`
2.If not installed, download from **python.org** or use Homebrew: `brew install python`
3. Linux (Ubuntu/Debian): Run `sudo apt install python3`
4.Verify installation with `python3 --version`
5. Use any text editor or install VS Code for a better experience

Choosing a Code Editor

A good code editor makes writing Python much more enjoyable and productive. Here are the most popular options for beginners:

IDLE

Comes bundled with Python. Simple and perfect for absolute beginners. No extra installation needed.

VS Code

Free, powerful, and highly recommended. Install the Python extension for syntax highlighting and debugging.

PyCharm

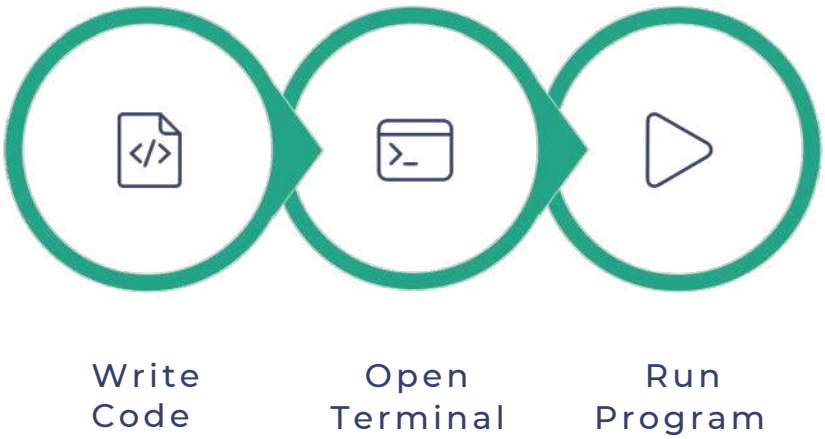
A full-featured IDE for Python. The Community Edition is free and excellent for larger projects.

Online Editors

Try **Replit.com** or **Google Colab** — no installation needed, just open your browser and start coding.

Running Your First Program

Once Python is installed, you can run programs in two ways. The **Interactive Mode** lets you type commands directly into the Python shell — great for testing small snippets. The **Script Mode** involves writing code in a .py file and running it. Create a file named hello.py, write print("Hello, World!"), and run it from your terminal with python hello.py.



Following these three steps will help you run any Python program on your computer. Once you are comfortable with this workflow, you are ready to start exploring Python's features in depth.

CHAPTER 3

Variables & Data Types

Variablesarelikelabeledcontainersthatstoredatainyourprogram'smemory.InPython, you create a variable by giving it a name and assigning a value using the = operator. Pythonis**dynamicallytyped**,meaningyoudonotneedtodeclarethedatatypeexplicitly — Python figures it out automatically based on the value assigned.

Variable Naming Rules

- Must start with a letter or underscore (_)
- Cannot start with a number
- Can only contain letters, numbers, and underscores
- Case-sensitive: `name` and `Name` are different
- Avoid Python keywords like `if`, `for`, `while`
- Use descriptive names: `student_name` is better than `x`

Convention: Use **snake_case** for variable names in Python — all lowercase with underscores separating words, e.g., `first_name`, `total_price`.

Examples of Variables

```
name = "Aisha"
age = 21
height = 5.6
is_student = True
course = "Python Basics"

print(name)
print(age)
print(type(name))
print(type(age))
print(type(height))
print(type(is_student))
```

Python Data Types

Every value in Python belongs to a specific data type. Understanding data types helps you write correct and efficient programs. Here are the core built-in types every beginner must know:

Data Type	Example	Description
`int`	`age = 25`	Integer — whole numbers, positive or negative
`float`	`price = 19.99`	Floating-point — decimal numbers
`str`	`name = "Sajid"`	String — text enclosed in quotes
`bool`	`is_active = True`	Boolean — represents True or False
`list`	`nums = [1, 2, 3]`	List — ordered, mutable collection
`tuple`	`coords = (10, 20)`	Tuple — ordered, immutable collection
`dict`	`person = {"name": "Ali"}`	Dictionary — key-value pairs
`set`	`unique = {1, 2, 3}`	Set — unordered, unique elements
`NoneType`	`value = None`	Represents the absence of a value

Type Conversion

You can convert between data types using built-in functions like `int()`, `float()`, `str()`, and `bool()`. This is especially useful when reading user input, since `input()` always returns a string.

```
num_str = "42"
num_int = int(num_str)      # Convert string to integer
price = float("19.99")      # Convert string to float
text = str(100)             # Convert number to string
is_true = bool(1)           # Convert to boolean (True)

# Getting user input
age = int(input("Enter your age: "))
print("Next year you will be:", age + 1)
```

Exercise: Create variables for your name, age, city, and favourite colour. Print each one using `print()`. Then use `type()` to check the data type of each variable. Bonus: Ask the user for their name using `input()` and greet them personally!

CHAPTER 4

Operators & Expressions

Operatorsaresymbolsthatperformoperationsonvaluesandvariables.An**expression**isacombination of values, variables, and operators that Python evaluates to produce a result. Understanding operators is fundamental to writing meaningful and dynamic programs.

Types of Operators in Python

Category	Operators	Description & Examples
Arithmetic	`+ - * / % // **`	Add, subtract, multiply, divide, modulus, floor divide, power
Comparison	`== != > < >= <=`	Compare values; return True or False
Logical	`and or not`	Combine or negate boolean conditions
Assignment	`= += -= *= /=`	Assign and update values in one step
Membership	`in not in`	Check if a value exists in a sequence
Identity	`is is not`	Check if two variables point to the same object

Arithmetic Operators

```
x = 10
y = 3

print(x + y)    # 13 Addition
print(x - y)    # 7 Subtraction
print(x * y)    # 30 Multiplication
print(x / y)    # 3.33 Division
print(x % y)    # 1 Modulus (remainder)
print(x // y)   # 3 Floor division
print(x ** y)   # 1000 Power (x raised to y)
print(abs(-10)) # 10 Absolute value
print(round(3.14159, 2)) # 3.14
```

Comparison & Logical Operators

```
a = 15
b = 7

# Comparison
print(a > b)    # True
print(a == b)   # False
print(a != b)   # True
print(a >= 15)  # True

# Logical
age = 20
has_id = True

print(age >= 18 and has_id) # True
print(age > 60 or has_id)   # True
print(not has_id)           # False

# Membership
fruits = ["apple", "banana", "cherry"]
print("banana" in fruits)    # True
print("mango" not in fruits) # True
```

Operator Precedence

When an expression contains multiple operators, Python evaluates them in a specific order known as **operator precedence**. This is similar to the BODMAS rule in mathematics. Use parentheses `()` to override the default order and make your intentions clear.

Highest Priority
`()` — Parentheses (always evaluated first)

Power& Unary
`**` — Exponentiation, then `+x`, `-x`, `~x`

Multiplicative
`*`, `/`, `//`, `%` — Multiplication, division, floor divide, modulus

Additive
`+`, `-` — Addition and subtraction

Lowest Priority
Comparison → Logical `not` → `and` → `or`

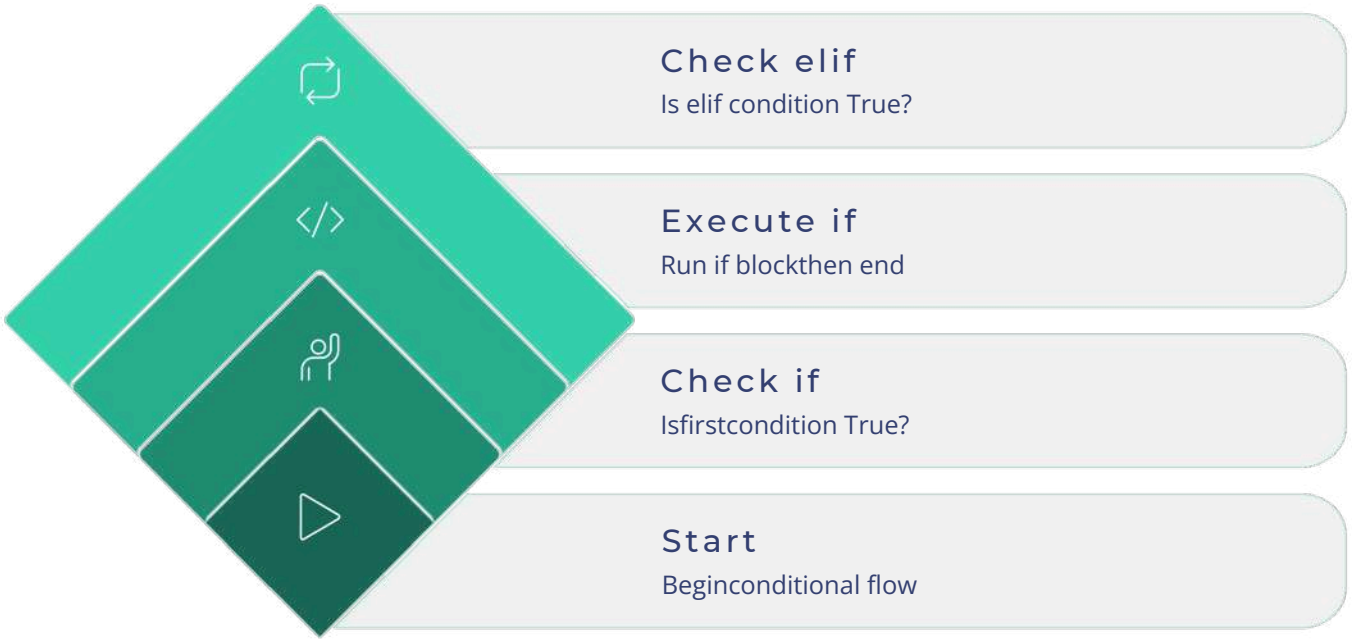
```
result = 2 + 3 * 4      # 14 (not 20!)
result = (2 + 3) * 4    # 20 (parentheses first)
result = 2 ** 3 + 1     # 9
result = 10 // 3 + 2    # 5
```

Exercise: Write a program that asks the user for two numbers and prints their sum, difference, product, quotient, and remainder. Then check if the first number is greater than the second using a comparison operator.

CHAPTER 5

Conditional Statements

Conditionalstatementsallowyour programto make decisions. Basedon whetheracondition is True or False, Python executes different blocks of code. This is what makes programs dynamic and responsive to different inputs and situations.



This flowchart illustrates how Python evaluates conditions in sequence. It checks the `if` condition first, then any `elif` conditions, and finally falls back to `else` if none are true. Only one block is ever executed per conditional structure.

if / elif / else Syntax

```
age = int(input("Enter age: "))

if age >= 18:
    print("You are an adult!")
elif age >= 13:
    print("You are a teenager.")
else:
    print("You are a child.")

# Multiple conditions
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
else:
    grade = "F"

print("Your grade:", grade)
```

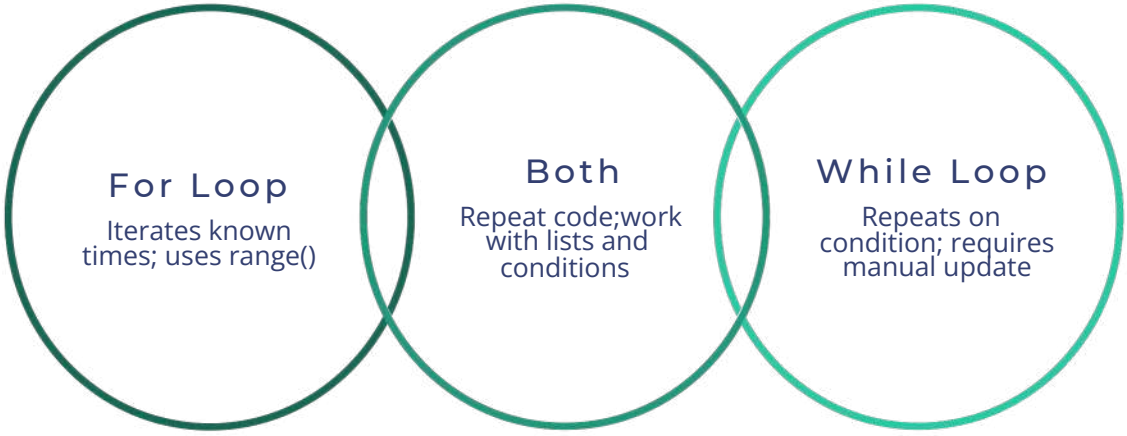
Important Rules to Remember

- Every conditional block must start with `if`.

CHAPTER 6

Loops

Loopsallow you torepeat a block of code multiple times. Python provides two main types of loops: for loops (used when you know how many times to iterate) and while loops (used when iteration depends on a condition). Loops areessential for automating repetitive tasks efficiently.



Choose a `for` loop when you know the number of iterations in advance, such as looping through a list or repeating something N times. Use a `while` loop when the number of iterations is not known beforehand and depends on a condition being met.

For Loop

```
# Basic for loop
for i in range(1, 6):
    print(i)        # 1, 2, 3, 4, 5

# Loop through a list
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# Loop with index
for i in range(len(fruits)):
    print(i, fruits[i])

# Nested for loop
for i in range(1, 4):
    for j in range(1, 4):
        print(i, "x", j, "=", i*j)
```

While Loop

```
# Basic while loop
count = 1
while count <= 5:
    print(count)
    count += 1

# Guessing game
secret = 7
guess = 0

while guess != secret:
    guess = int(input("Guess (1-10): "))
    if guess < secret:
        print("Too low!")
    elif guess > secret:
        print("Too high!")
print("Correct! You won!")

# Infinite loop with break
while True:
 cmd = input("Enter command: ")
 if cmd == "quit":

        break
 print("You entered:", cmd)
```

Loop Control Statements

Python provides three special statements to control loop execution: `break` exits the loop immediately, `continue` skips the current iteration and moves to the next, and `else` runs after the loop completes normally (without a `break`).

```
# break - exit loop early
for i in range(1, 11):
    if i == 6:
        break
    print(i)        # Prints 1 to 5

# continue - skip iteration
for i in range(1, 8):
    if i == 4:
        continue
    print(i)        # Skips 4

# else with for loop
for i in range(1, 4):
    print(i)
else:
 print("Loop completed!")

# else with while (not executed if break)
n = 1
while n <= 3:
    print(n)
    n += 1
else:
 print("While loop ended")
```

Exercise: Write a program that prints the multiplication table of 5 (from 5×1 to 5×10) using a for loop. Then write another program using a while loop that asks the user to enter numbers until they type "stop", and prints the sum of all entered numbers.

CHAPTER 7

Functions

Afunctionis a reusableblock ofcodethat performs a specific task. Instead of writing the same code multiple times, you define it once as a function and call it whenever needed. Functions make your codecleaner, moreorganised, andeasier to debug and maintain. Python uses the def keyword to define functions.

Define Call Execute Return

Every function follows these four stages. You first define it with a name and optional parameters, then call it from other parts of your code, Python executes the function body, and finally it returns a result if specified.

Defining & Calling Functions

```
# Function without parameters
def greet():
    print("Hello, World!")

greet()
greet()
# Function with parameters
def greet_person(name):
    print("Hello,", name, "!")

greet_person("Aisha")
greet_person("Sajid")
# Function with return value
def add(a, b):


    return a + b

result = add(5, 3)
print(result)      # 8
print(add(10, 20))   # 30
```

Types of Arguments

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

greet()          # Hello, Guest
greet("Ali")     # Hello, Ali
# Keyword arguments
def describe_pet(pet, name):
    print("I have a", pet, "named", name)

describe_pet(pet="dog", name="Rex")
describe_pet(name="Kitty", pet="cat")
# Variable-length arguments (*args)
def add_all(*numbers):


    total = 0
    for n in numbers:
        total += n
    return total

print(add_all(1, 2, 3, 4, 5)) # 15
```

Scope & Best Practices

Variables created inside a function are **local** — they exist only within that function. Variables defined outside all functions are **global** and can be accessed anywhere. Always use `return` to send results back from a function. Keep functions focused on a single task and give them descriptive names.

```
# Local vs Global scope
x = 10        # Global variable

def my_func():
    y = 5     # Local variable
    print(x)  # Can read global
    print(y)  # Can read local

my_func()
# print(y)    # Error! y is not defined here

# Using global keyword
count = 0
def increment():

    global count
    count += 1

increment()
increment()
print(count)  # 2
```

Exercise: Write a function `is_even(n)` that returns `True` if a number is even and `False` otherwise. Then write a function `find_max(a, b, c)` that returns the largest of three numbers without using `max()`. Test both functions with different inputs.

CHAPTERS 8 & 9

Data Structures & Strings

Pythonprovidespowerfulbuilt-indatastructurestoorganiseandmanipulatedataefficiently.Understanding when to use each one is a key skill. Additionally, strings are one of the most commonly used data types — Python offers a rich set of string methods to process and transform text.

The Four Core Data Structures

Structure	Example	Mutable	Ordered	Allows Duplicates
list	[1,2,3]	Yes	Yes	Yes Yes
tuple	(1,2,3)	No	Yes	No
set	{1,2,3}	Yes	No	Keys only
dict	{"a":1}	Yes	Yes	

Lists — Ordered & Mutable

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
fruits.insert(1, "grape")
fruits.remove("banana")
print(fruits[0])    # apple
print(fruits[-1])   # orange
print(len(fruits))  # 4
# List comprehension
squares = [x**2 for x in range(1, 6)]
print(squares)      # [1,4,9,16,25]
```

Tuples — Ordered & Immutable

```
coords = (10, 20)
person = ("Ali", 25, "Engineer")
print(coords[0])    # 10
# coords[0] = 5     # Error!
print(person.count("Ali"))
```

Sets — Unique & Unordered

```
nums = {1, 2, 3, 3, 4}
print(nums)         # {1,2,3,4}
nums.add(5)
nums.discard(2)
print(3 in nums)    # True
# Set operations
a = {1, 2, 3}
b = {3, 4, 5}
print(a | b)
print(a & b)        # Union: {1,2,3,4,5}
print(a - b)        # Intersection: {3}
                    # Difference: {1,2}
```

Dictionaries — Key-Value Pairs

```
student = {
"name": "Aisha",
  "age": 21,
  "course": "Python"
}
print(student["name"])
student["age"] = 22
student["grade"] = "A"
print(student.keys())
print(student.values())
```

Strings & String Methods

Strings in Python are sequences of characters enclosed in single or double quotes. They are **immutable** — you cannot change individual characters. Python provides many built-in methods to manipulate strings.

```
text = " Hello, Python World! "

print(text.upper())         # HELLO, PYTHON WORLD!
print(text.lower())         # hello, python world!
print(text.strip())         # Removes spaces
print(text.split(","))      # [' Hello', ' Python World! ']
print(text.replace("Python", "Java"))
print(text.find("Python"))  # Index of substring
print(text.count("o"))      # Count occurrences
print("Python" in text)     # True
# String formatting
name = "Sajid"
age = 25
print(f"My name is {name} and I am {age}")
print("Name: {}, Age: {}".format(name, age))

# Slicing
s = "Python"
print(s[0:3])
print(s[::-1])  # Pyt
print(s[1:4])   # nohtyP (reverse)
                # yth
```

Exercise: Create a list of your five favourite foods. Add one item, remove one, sort the list, and print it. Then create a dictionary for a book with keys: title, author, year, and price. Print each value using its key. Finally, take a sentence as input from the user and count how many vowels it contains.

FINAL CHAPTER

Mini Projects, Exercises & About CrescaDemy

Congratulationsonmakingitthroughallthechapters!Thebestwaytosolidifyyour Pythonknowledgeisbybuildingsmallprojectsandsolvingpracticeproblems. Below you will find three beginner-friendly mini projects, a set of practice exercises with solutions, and information about CrescaDemy — your partner in continuous learning.

Mini Project 1: Number Guessing Game

```
import random secret = random.randint(1, 100)
attempts = 0
print("I'm thinking of a number between 1 and 100.")

while True:

 attempts += 1

   guess = int(input("Your guess: "))


   if guess < secret:
      print("Too low!")
    elif guess > secret:
      print("Too high!")
   else:
      print(f"Correct! You won in {attempts} attempts.")
      break
```

Mini Project 2: To-Do List Manager

```
tasks = [] while True:
 print("\n1. Add Task 2. View Tasks 3. Remove Task 4. Exit")


   choice = input("Choose (1-4): ")

   if choice == "1":
      task = input("Enter task: ")
      tasks.append(task)
      print("Task added!")
   elif choice == "2":
      for i, t in enumerate(tasks, 1):
         print(f"{i}. {t}")
   elif choice == "3":
      num = int(input("Task number to remove: "))
      tasks.pop(num - 1)
      print("Task removed!")
   elif choice == "4":
      print("Goodbye!"); break
```

Mini Project 3: Simple Calculator

```
def add(x, y): return x + y def sub(x, y): return
x - y def mul(x, y): return x * y
def div(x, y): return x / y if y != 0 else "Error"
print("Simple Calculator")
print("1.Add 2.Subtract 3.Multiply 4.Divide")
op = input("Choose (1-4): ")
a = float(input("First number: "))
b = float(input("Second number: "))

if op == "1": print("Result:", add(a, b))
elif op == "2": print("Result:", sub(a, b))
elif op == "3": print("Result:", mul(a, b))
elif op == "4": print("Result:", div(a, b))
else: print("Invalid choice")
```

Exercise Solutions

```
# Ch 3: Check data types name, age = "Aisha", 21
print(type(name), type(age))
# Ch 4: Two-number calculator
a = float(input("Num 1: "))
b = float(input("Num 2: "))
print("Sum:", a+b, "Diff:", a-b, "Prod:", a*b)
# Ch 5: Day of week
day = int(input("Day (1-7): "))
days = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
print(days[day-1] if 1<=day<=7 else "Invalid")
# Ch 6: Multiplication table of 5
for i in range(1, 11):


   print(f"5 x {i} = {5*i}")

# Ch 7: is_even and find_max
def is_even(n): return n % 2 == 0
def find_max(a, b, c):
 m = a
 if b > m: m = b
 if c > m: m = c
 return m

# Ch 8-9: Vowel counter
s = input("Enter text: ")
vowels = "aeiouAEIOU"
count = sum(1 for c in s if c in vowels)
print("Vowels:", count)
```

About CrescaDemy

CrescaDemy, powered by **CRESCENTON**, is an innovative learning platform offering hands-on education in **Programming**, **Website Development**, **AI & Data Science**, and **Prompt Engineering.**

Visit Us

crescademy.netlify.app

Compiler

Syed Sajid Ul Haq — Python Educator & Developer

Keep Learning

Practice daily, build projects, and never stop exploring!

> *"The best way to learn to code is to write code. Start small, stay curious,andbuildsomethingeveryday."* — Syed Sajid Ul Haq, CrescaDemy