python-codes

*What is software?

Software is a **set of instructions, data, or programs** used to operate a computer and perform specific tasks.

Hardware = the physical parts of a computer (like CPU, keyboard, monitor).
Software = the programs that tell the hardware what to do.

Types of Software:

System Software – helps run and control computer hardware.
Examples: Operating Systems (Windows, Linux, macOS), Device drivers.
Application Software – programs that help users perform tasks.
Examples: MS Word, Chrome, WhatsApp, Photoshop.
Programming software is a type of software that provides tools to write, test, and debug other programs.
Examples: VS Code, Eclipse, PyCharm, GCC compiler, Python interpreter.

*What is code?

Code is a set of instructions written in a programming language that tells a computer what to do.

Example:print("Hello, World!") is a line of code in Python.

*What is Program?

A **program** is a collection of codes or instructions written in a programming language that a computer can execute to perform a specific task.

*What is syntax?

Syntax in programming is the set of rules that defines the correct structure and format of code in a programming language.

Example: In Python, `print(“Hello”)`is correct syntax, but `print "Hello"` is wrong syntax.

*Why Do We Use Python?

We use **Python** because it is:

Simple & Easy to Learn – Its syntax is close to English, making it beginner-friendly.
Versatile – Can be used for web development, data science, AI/ML, automation, game development, etc.

Large Community & Libraries – Has huge support and ready-made libraries (like NumPy, Pandas, Django, TensorFlow).

Cross-Platform – Runs on Windows, macOS, Linux easily.

Fast Development – Requires fewer lines of code compared to many other languages.

*Variables:

A **variable** is a name given to a memory location that stores data in a program.
It acts like a container to hold values that can change during execution.

Example in Python:

name = "Salini"

age = 20

print(name, age)

Here, `name` and `age` are variables.

*Datatypes:

Datatypes define the kind of value a variable can hold in a program.

Common Python Datatypes:

- **int** → whole numbers (e.g., `10`, `-5`)
- **float** → decimal numbers (e.g., `3.14`, `-2.5`)
- **str** → text/strings (e.g., `"Hello"`, `'Python'`)
- **bool** → logical values (`True`, `False`)
- **list** → ordered collection (e.g., `[1, 2, 3]`)
- **tuple** → ordered but immutable collection (e.g., `(4, 5, 6)`)
- **set** → unordered collection of unique items (e.g., `{1, 2, 3}`)
- **dict** → key-value pairs (e.g., `{"name": "Salini", "age": 20}`)

*What is frontend,Backend and Data bases?

Frontend = What users see

Backend = What makes the app work behind the scenes

Database = Where the data is stored

*What is String?

A **string** is a datatype in programming that is used to store **text**.
 It is a sequence of characters enclosed in **quotes**.

Examples in Python**:**

name = "Salini"

greeting = 'Hello, World!'.

Strings can contain **letters, numbers, symbols, and spaces**.

***Concatenation:**

Concatenation is the process of **joining two or more strings together** to make a single string.

Example in Python**:**

first_name = "Salini"

last_name = "Degala"

full_name = first_name + " " + last_name

print(full_name)

***Type Casting:**

Type Casting is the process of **converting a variable from one data type to another**.

Example in Python**:**

x = "10"

y = int(x)

z = float(x)

```
print(y)

print(z)
```

This is useful when you need to **perform operations between different data types**.

*Slicing:

Slicing is a way to **extract a part (substring or sublist) of a sequence** like a string, list, or tuple.

Syntax:

```
Sequence[start:stop:step]
```

`start` → starting index (inclusive)
`stop` → ending index (exclusive)
`step` → step size (optional)

Example with a string**:**

```
text = "Hello, World!"

print(text[0:5])   # Output: Hello

print(text[7:12])  # Output: World
```

*What is the function we want to check the data type?

Type()

```
#Add 2 numbers pogram:

a=int(input())

b=int(input())

print(a+b)

#check even or odd:

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

  if num % 2 == 0:

      print("The number is even.")
  else:
      print("The number is odd.")
```

*Operators:

Relational Operators:

Operator	Meaning	Example	Result
==	Equal to	5==5	True
!=	Not equal to	5 != 3	True
>	Greater than	5 > 3	True
<	Less than	5 < 3	False
>=	Greater or equal	5 >= 5	True
<=	Less or equal	3 <= 5	True

Logical Operators:

Operator	Meaning	Example	Result
and	True if both are True	(5>3 and 2<4)	True

`or`	True if any one is True	`(5>3 or 2>4)`	True
`not`	Reverses True/False	`not(5>3)`	False

3. Conditional Statements

Used to **perform different actions based on conditions**.

Syntax:

```
if condition:

    # code if condition is True

elif another_condition:

    # code if this condition is True

else:

    # code if all conditions are False
```

Example:

```
num = 10

if num > 0:

    print("Positive")

elif num == 0:

    print("Zero")

else:

    print("Negative")
```

For Loop

A **for loop** is used to **repeat a block of code a fixed number of times**.

Example:

```
for i in range(5):

    print(i)
```

While Loop

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

Example:

```
i = 0

while i < 5:

    print(i)

    i += 1
```

# python-codes

* Write program that reads a number N and prints square of N rows and N columns using numbers starting from 1

```
N = int(input())

num = 1

for i in range(N):

    for j in range(N):

        print(num, end=" ")

        num += 1

    print()
```

* Write a program that reads a number N and prints two Right

```
N = int(input())

for i in range(1, N+1):

    print("* " * i)

print()

for i in range(1, N+1):

    print("* " * i)
```

* Angled Triangles of N rows, using numbers starting from 1.

```
N = int(input())

num = 1

for i in range(1, N+1):

    for j in range(i):

        print(num, end=" ")

        num += 1

    print()
```

* Write a program to print the factorial of N Factorial is the product of all positive integers less than or equal to N.

```
N = int(input())

fact = 1

for i in range(1, N+1):
    fact *= i
print(fact)
```

* Write a program that reads a string and prints the count of vowels in the string.

```
s = input()
count = 0
for ch in s:
    if ch.lower() in "aeiou":
        count += 1
print(count)
```

* Given two numbers X and N, write a program to print the sum of N terms in the given series.

Series:

(x)^2, (xx)^2, $(xxx)^2$, N terms

```
X = int(input())
N = int(input())
total = 0
num = 0
for i in range(N):
    num = num * 10 + X
    total += num ** 2

print(total)
```

* given a string write a program to print alphabets only alphabets in the given string

```
s = input()

for ch in s:

    if ch.isalpha():

        print(ch, end="")

print()
```

* given string write program that prints all the uppercases letters of the string

```
s = input()

for ch in s:

    if ch.isupper():

        print(ch, end="")

print()
```

* see about built in functions and what is the use case of builtin functions its advantages and where do we use it

Built-in functions are functions that come predefined in Python and are ready to use without importing any module. Examples: len(), sum(), max(), min(), sorted().

Use case & advantages:

Quick and easy to use without writing code from scratch

Reliable because they are tested and optimized

Saves time and improves code readability

Example:

```
numbers = [5, 10, 2, 7]

print(max(numbers))
```

* try to do a simple calculator using python

```
a = float(input())

b = float(input())

op = input()

if op == "+":

    print(a + b)

elif op == "-":

    print(a - b)

elif op == "*":

    print(a * b)

elif op == "/":

    print(a / b)

else:

    print("Invalid operator")
```

* Find perimeter of a rectangle

```
length = float(input("Enter length of rectangle: "))

width = float(input("Enter width of rectangle: "))

perimeter_rectangle = 2 * (length + width)

print("Perimeter of rectangle:", perimeter_rectangle)
```

* Triangle:

```
a = float(input("Enter side a of triangle: "))

b = float(input("Enter side b of triangle: "))

c = float(input("Enter side c of triangle: "))

perimeter_triangle = a + b + c

print("Perimeter of triangle:", perimeter_triangle)
```

* Make a mini calculator get input for 2 numbers a and b get input add,sum,div,mul then if user select add them add 2 numbers and print the result

```
x = float(input("Enter first number: "))

y = float(input("Enter second number: "))

operation = input("Choose operation (add, sub, mul, div): ")

if operation == "add":

    print("Result:", x + y)

elif operation == "sub":

    print("Result:", x - y)

elif operation == "mul":
    print("Result:", x * y)

elif operation == "div":

    print("Result:", x / y)

else:

    print("Invalid operation")
```

Module1 Air Quality Eda

Module 1: Understanding and Exploring Air Quality Data

Introduction

This module is about exploring and making sense of air quality data from real-world sources like CPCB (Central Pollution Control Board, India) and Open AQ. The goal is to understand how pollutants behave over time and to prepare the information for future predictions, all in a way that makes sense to humans rather than machines.

---

1. Collecting Data

Where to get it:

- **CPCB:** Provides air quality information from different cities in India. It includes pollutants such as PM2.5, PM10, NO2, O3, and CO.
- **Open AQ:** Offers worldwide air quality data, including historical readings from multiple cities.

How to approach it:

- Download the datasets in an easy-to-use format like CSV or Excel.
- Focus on the information that tells you the story: where, when, and what levels of pollutants were measured.

---

2. Cleaning and Preparing Data

Before analyzing, the data should be made understandable:

- **Fill in the gaps:** Look for missing measurements and decide whether to estimate or ignore them.
- **Keep the timeline straight:** Make sure dates and times are in order.
- **Remove repeats:** Avoid confusion by removing duplicated entries.
- **Focus on what matters:** Keep the columns that tell the story—location, time, and pollutant levels.
- **Smooth it out:** Combine data into daily or weekly averages so patterns are easier to see.

---

3. Exploring the Data

Try to understand the data like a detective:

- **Look for trends:** Are there days, weeks, or months when pollution is higher? Can you see seasonal patterns?
- **Compare pollutants:** How do different pollutants relate to each other? Does a rise in one mean a rise in another?
- **Check the spread:** Are there unusual spikes or drops that might indicate an event or error?

---

4. Making the Data Useful for Forecasting

- Organize the data into regular time periods, like daily or weekly averages, so it's easier to notice patterns.
- Preparing the data thoughtfully will make it easier to make informed guesses about future pollution levels.

---

5. Adding Human Insight Features

- **Previous days matter:** Sometimes yesterday's pollution tells you something about today.
- **Averaging helps:** Look at averages over a week to see the bigger picture.
- **Time context:** Remember the month or season, because air quality can change naturally over time.

Conclusion

By following this approach, you will have:

- Clean, understandable air quality data.
- Insights about patterns and relationships between pollutants.
- Organized information ready for analysis or future forecasting, all in a way that is easy for humans to interpret.