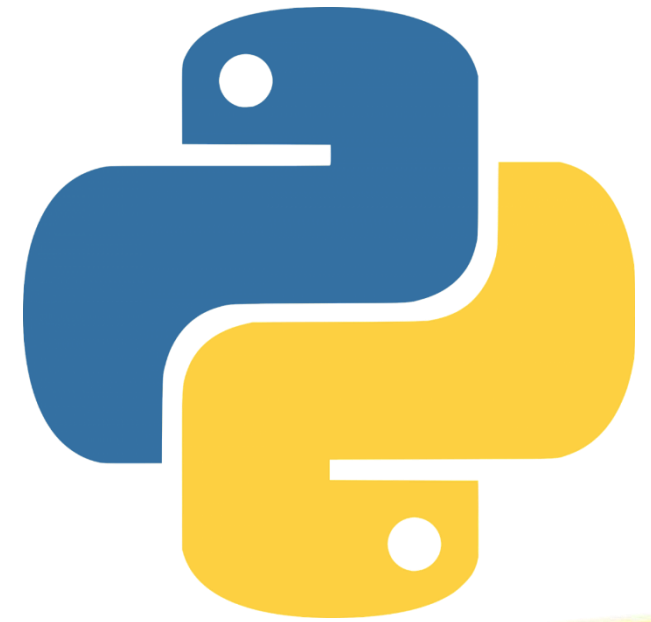


# INTRODUCTION TO PYTHON





# WHAT IS PYTHON?

- Python is world's fastest growing and most popular programming language.
- It is popular among different professions like
  - Software Engineers
  - Mathematicians
  - Data Scientist
  - Scientist
  - Network Engineers
  - Accountants
  - Kids too.

# WHERE TO USE PYTHON?

Data Analysis

AI / ML

Automation

WEB  
APPS

Mobile  
APPS

Desktop  
Apps

Software  
Testing

Hacking

# WHY PYTHON?

- It's a multipurpose language with a simple and beginner friendly syntax
- Solve problems in less times and fewer lines of code.
- Let's take one example:
- Extract the first 3 letters from string "Hello World"

C#

```
str.Substring(0, 3)
```

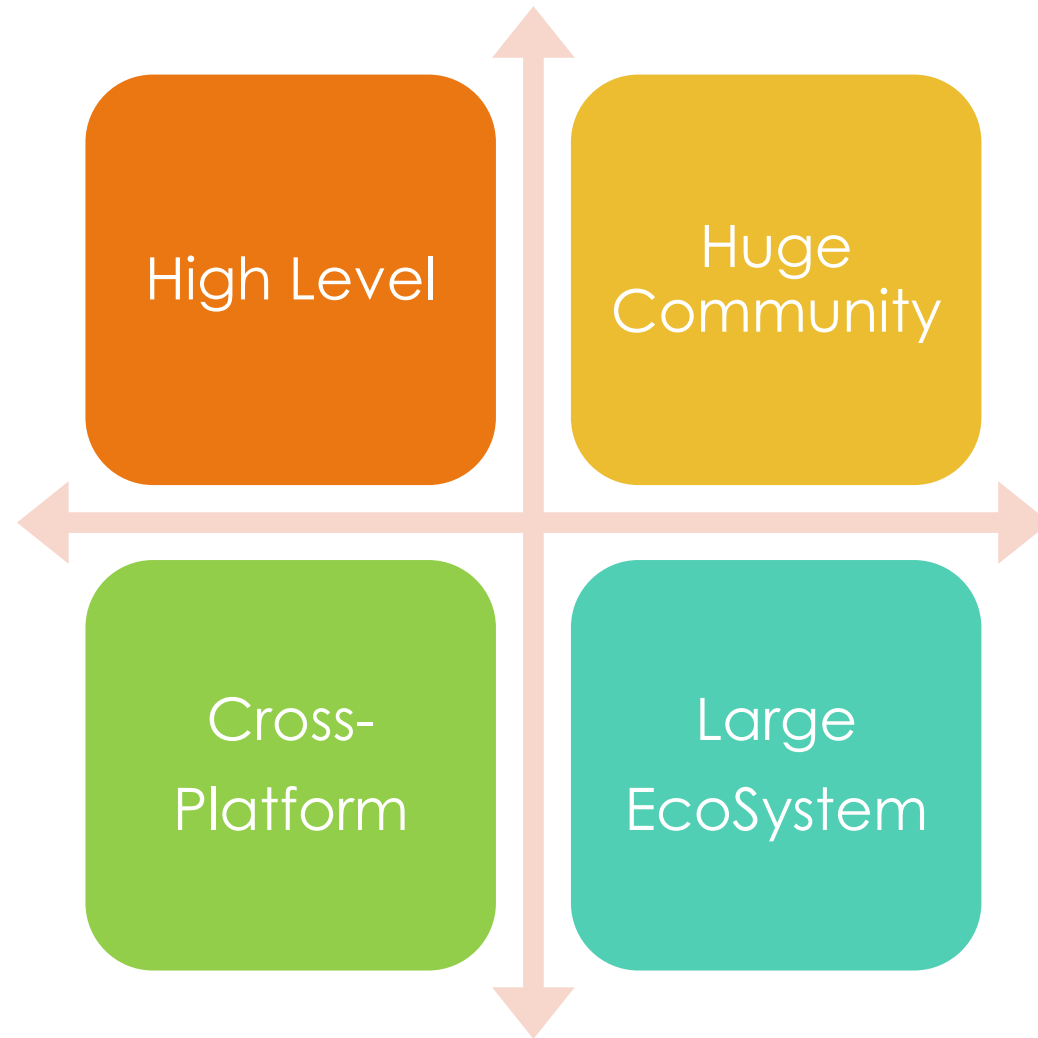
JAVASCRIPT

```
str.substr(0, 3)
```

PYTHON

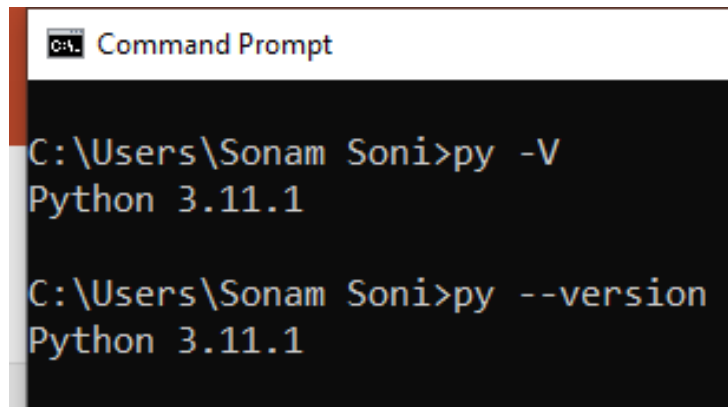
```
str[0:3]
```

# WHY PYTHON IS SO POPULAR?



# PYTHON INSTALLATION

- <https://www.python.org/downloads/>
- Download the latest version and install.
- Once it is install you can open python command line directly or type Python in cmd to open python command line over there.
- To check python version you can type below command.
- `py -V` (`py --version`)

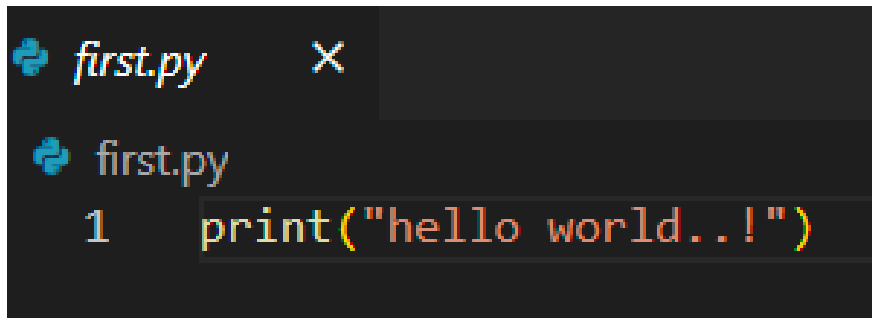


```
Command Prompt
C:\Users\Sonam Soni>py -V
Python 3.11.1

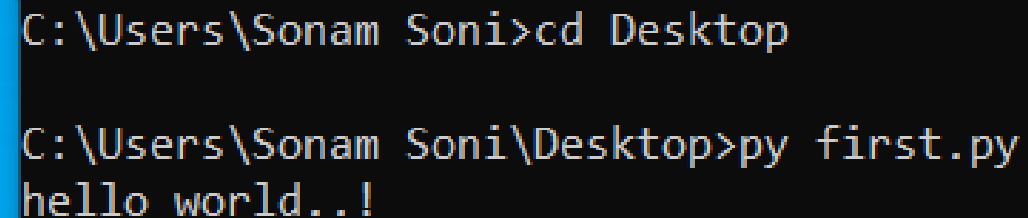
C:\Users\Sonam Soni>py --version
Python 3.11.1
```

# HOW TO RUN PYTHON SCRIPT?

- Open notepad
- add below mentioned code
- Print("Hello World")
- Save this file as first.py
- Open cmd and move to that folder where you have saved this file.
- Execute below mentioned command.



A screenshot of a Notepad window with a dark background. The title bar shows 'first.py' and a close button. The editor contains a single line of Python code: `1 print("hello world..!")`. The code is color-coded: `print` is in blue, the string `"hello world..!"` is in orange, and the number `1` is in white.



A screenshot of a Windows Command Prompt window with a dark background. The prompt shows the user navigating to the Desktop directory and running a Python script. The text is as follows:  
C:\Users\Sonam Soni>cd Desktop  
  
C:\Users\Sonam Soni\Desktop>py first.py  
hello world..!



# WHAT IS ANACONDA?

- It is a Python distribution. A Python distribution is a program that allows you to use Python. It may contain more than one program in it.
- Anaconda contains multiple programs that let you use Python.
  - **Jupyter Notebook**
  - **Spyder**
- These programs in Anaconda are specialized in data science (e.g. you can't develop a website).
- So if you want to continue on data science, learn more about Anaconda but if you want to build an app (that doesn't include data science) with Python, don't think about Anaconda much.



# WHAT IS JUPYTER NOTEBOOK?

- 1 It is a web-based program under Anaconda distribution and it let you code Python.
- 2 You can also call it a web application under Anaconda.
- 3 It is good for data analysis.
- 4 You can visualize data easily.
- 5 It is very interactive and lets you run partial codes.

# WHAT IS JUPYTERLAB?



It lets you collect multiple Jupyter Notebooks under one tab.

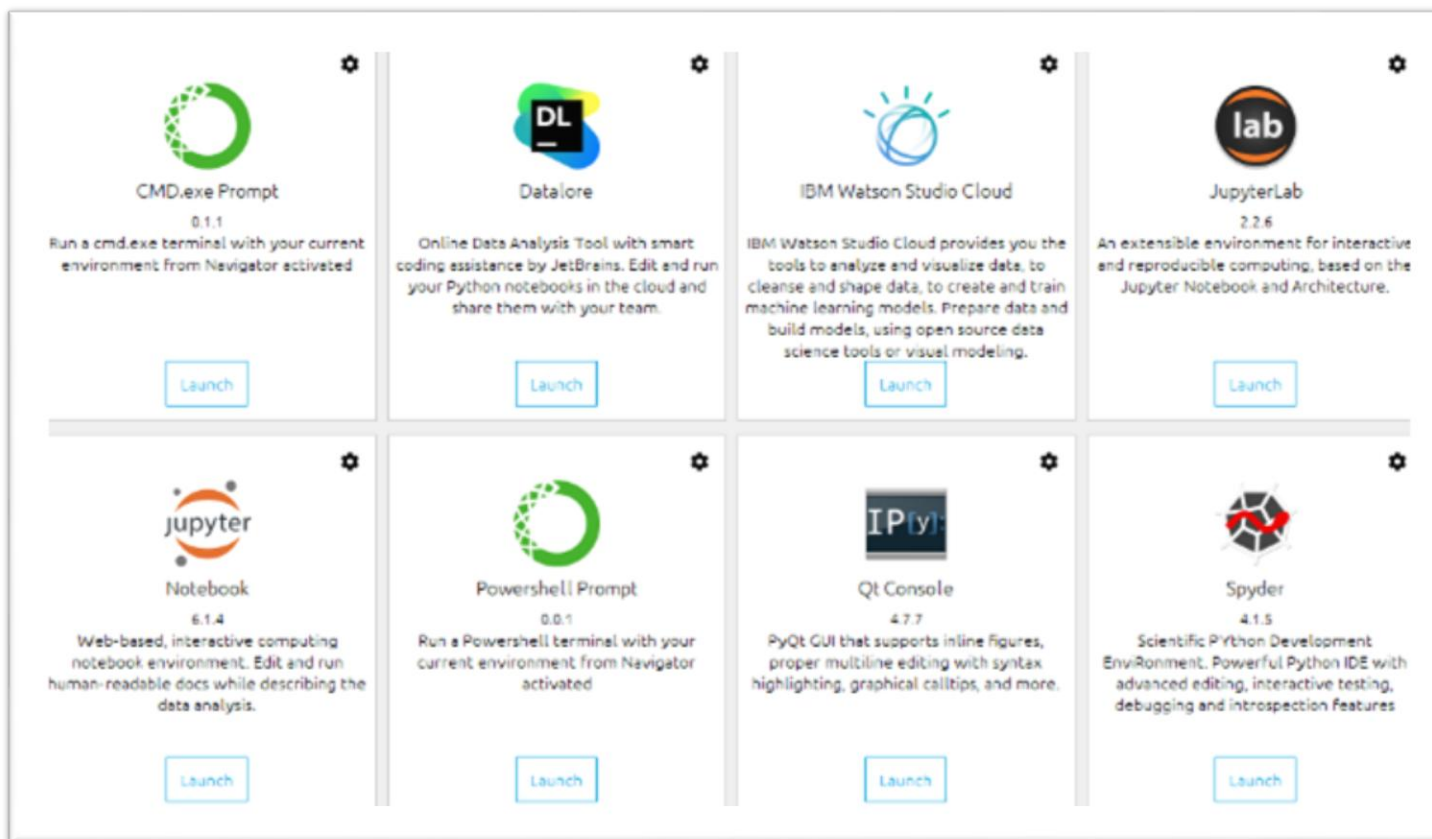
You can open Jupyter Notebooks on multiple web tabs

But if you are used to using IDEs, it might seem more user-friendly to use JupyterLab.

It is also under Anaconda.

# WHAT IS ANACONDA NAVIGATOR?

- It is the GUI of Anaconda distribution. You can manage the programs and all other features of Anaconda.



# WHAT IS SPYDER?

- It is an IDE for Python under Anaconda.
- You can code Python here but there are some differences with Jupyter Notebook.
- Spyder is not interactive as Jupyter but more efficient for building a whole data science application.

# WHAT TO USE, SPYDER OR JUPYTER NOTEBOOK?

You may prefer Jupyter Notebook for data analysis, data visualization during the process of decisions and presentations.

Once you are done with the analysis and decision process and want to build a data science application, you may prefer to jump to Spyder.

# LET'S INSTALL JUPYTER

- To install Jupyter using pip, we need to first check if pip is updated in our system. Use the following command to update pip
  - `py -m pip install --upgrade pip`
- Command to install Jupyter:
  - `py -m pip install jupyter`
- Start
  - `Py -m jupyter notebook`

```
Command Prompt
C:\Users\Sonam Soni\Desktop>py -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\sonam soni\appdata\local\programs\python\python38\python.exe (21.1.3)
To check if pip's version is up-to-date run:
  python -m pip --help
C:\Users\Sonam Soni\Desktop>py -m pip install jupyter
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Collecting notebook
  Downloading notebook-6.5.2-py3-none-any.whl (439 kB)
----- 439.1/439.1 kB 3.9 MB/s eta 0:00:00
Collecting qtconsole
  Downloading qtconsole-5.4.0-py3-none-any.whl (121 kB)
----- 121.0/121.0 kB 3.5 MB/s eta 0:00:00
Collecting jupyter-console
  Downloading jupyter_console-6.4.4-py3-none-any.whl (22 kB)
Collecting nbconvert
  Downloading nbconvert-7.2.7-py3-none-any.whl (273 kB)
----- 273.2/273.2 kB 8.5 MB/s eta 0:00:00
Collecting ipykernel
  Downloading ipykernel-6.19.4-py3-none-any.whl (145 kB)
----- 145.2/145.2 kB 8.4 MB/s eta 0:00:00
Collecting ipywidgets
  Downloading ipywidgets-8.0.4-py3-none-any.whl (137 kB)
----- 137.8/137.8 kB 8.5 MB/s eta 0:00:00
Collecting jupyterlab
  Downloading jupyterlab-3.5.1-py3-none-any.whl (6.9 MB)
----- 6.9/6.9 MB 8.5 MB/s eta 0:00:00
```



# LAUNCH JUPYTER

Command Prompt - jupyter notebook

```
Microsoft Windows [Version 10.0.19045.2364]
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Users\Sonam Soni>jupyter notebook
```

```
[I 13:01:37.902 NotebookApp] Writing notebook server cookie secret to C:\Users\Sonam Soni\AppData\Roaming\jupyter\runti
[I 13:01:39.267 NotebookApp] Serving notebooks from local directory: C:\Users\Sonam Soni
[I 13:01:39.267 NotebookApp] Jupyter Notebook 6.5.2 is running at:
[I 13:01:39.268 NotebookApp] http://localhost:8888/?token=7b3ac6f61dea3d06425220b51a6229ac830fa247225c9ae8
[I 13:01:39.268 NotebookApp] or http://127.0.0.1:8888/?token=7b3ac6f61dea3d06425220b51a6229ac830fa247225c9ae8
[I 13:01:39.268 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 13:01:39.387 NotebookApp]
```

To access the notebook, open this file in a browser:

file:///C:/Users/Sonam%20Soni/AppData/Roaming/jupyter/runtime/nbserver-3428-open.html

Or copy and paste one of these URLs:

http://localhost:8888/?token=7b3ac6f61dea3d06425220b51a6229ac830fa247225c9ae8

or http://127.0.0.1:8888/?token=7b3ac6f61dea3d06425220b51a6229ac830fa247225c9ae8

0.01s - Debugger warning: It seems that frozen modules are being used, which may

0.00s - make the debugger miss breakpoints. Please pass -Xfrozen\_modules=off

0.00s - to python to disable frozen modules.

0.00s - Note: Debugging will proceed. Set PYDEVD\_DISABLE\_FILE\_VALIDATION=1 to disable this validation.



# WHAT IS THE DIFFERENCE BETWEEN A PACKAGE AND A LIBRARY?

Package	Library
Packages are a set of modules that contain scripts and functions. You can write your own modules and packages.	When many packages come together, they build libraries.

**Package manager:** tools that help you manage the dependencies for your project.

A package manager also manages the libraries because libraries are the collections of packages. Examples: Conda, pip

# WHAT IS CONDA AND PIP?

## What is Conda?

- It is an environment and package manager.
- **A package manager** makes it easy to install, upgrade, remove packages to a virtual environment.
- Conda makes it easy to manage Python to Conda environments that are the programs under Anaconda.

## What is Pip?

- It is a package manager.
- It manages (install, upgrade, remove, etc.) Python packages in any Python virtual environment.
- Pip makes it easy to manage Python packages to any Python environments.

# WHAT IS FRAMEWORK?

- Frameworks let you build software tools by making the coding process easier and more efficient.
- They are composed of libraries but the main difference is You can call a function of a library; however, a framework calls your function.
- You have more control when you use a library but you have to follow the rules of a framework.
- E.g. **Django & Flask**
  - They are frameworks for web development. You can build websites and web applications with them.

# PYTHON COMMENTS

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- Comments starts with a # , and Python will ignore them.
- E.g.

```
#This is a comment  
print("Hello, World!")
```

# PYTHON VARIABLES

- Variables are containers for storing data values.
- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- E.g.  
    `x = 5`  
    `y = "John"`

# HOW TO GIVE VARIABLES NAMES?

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).
- Rules for Python variables:
  - **A variable name must start with a letter or the underscore character**
  - **A variable name cannot start with a number**
  - **A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)**
  - **Variable names are case-sensitive (age, Age and AGE are three different variables)**

# ASSIGN VALUES TO VARIABLES

- `x, y, z = "Orange", "Banana", "Cherry"`
- One Value to Multiple Variables: `x = y = z = "Orange"`
- Collection values:  
`fruits = ["apple", "banana", "cherry"]`  
`x, y, z = fruits`
- To print these values, we can use print functions.
  - `Print(x)`
  - `Print(x,y,z)`
  - `Print(x + y + z)`



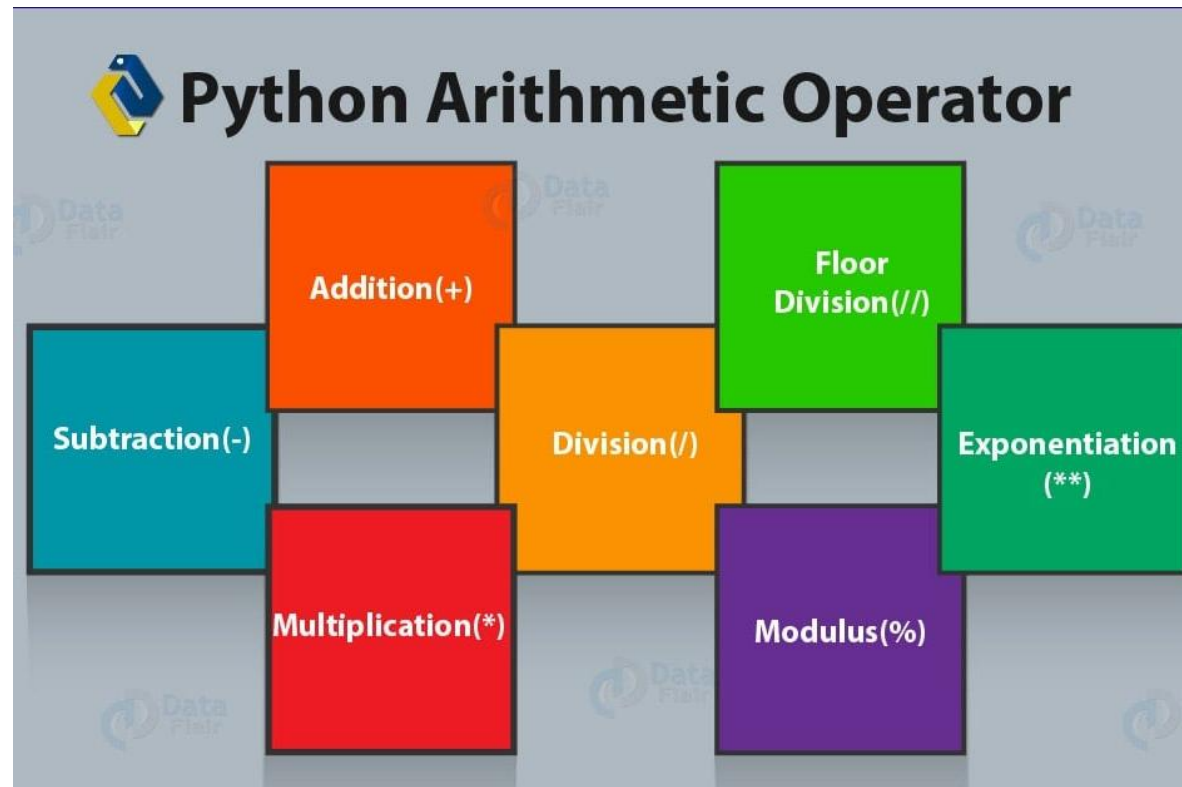
# PYTHON DATA TYPES

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: List, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview
- None Type: Nonetype

To get the datatype of any variable you can use  
type()  
e.g. **print(type(x))**

# OPERATORS

Operators are used to perform operations on variables and values.



# EXAMPLE

```
# Declare two numbers
num1 = 10
num2 = 3

# Addition
addition = num1 + num2
print("Addition:", addition)

# Subtraction
subtraction = num1 - num2
print("Subtraction:", subtraction)

# Multiplication
multiplication = num1 * num2
print("Multiplication:", multiplication)
```

```
# Division
division = num1 / num2
print("Division:", division)

# Modulus (remainder)
modulus = num1 % num2
print("Modulus:", modulus)

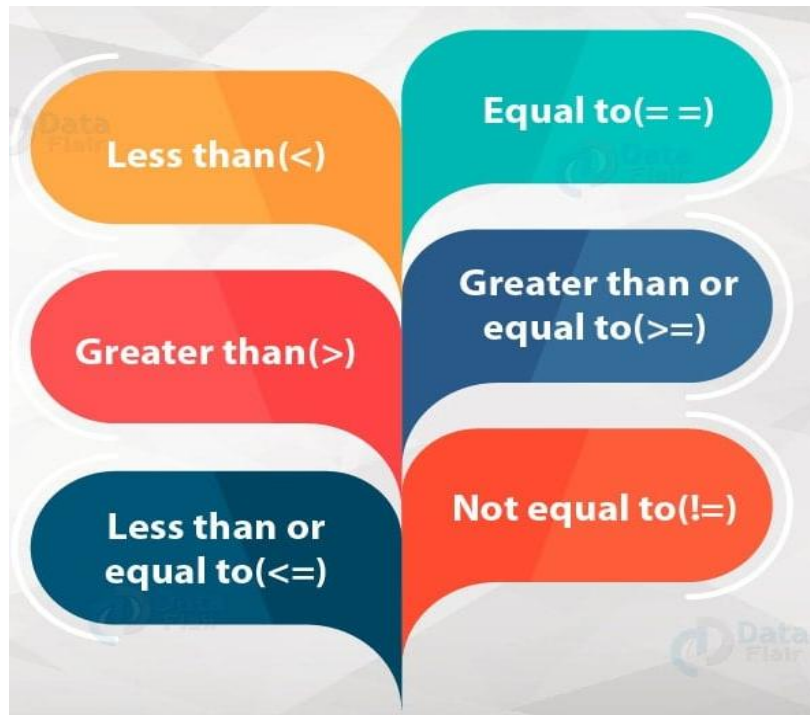
# Exponentiation (power)
exponentiation = num1 ** num2
print("Exponentiation:", exponentiation)

# Floor Division
floor_division = num1 // num2
print("Floor Division:", floor_division)
```

# OPERATORS

Operators are used to perform operations on variables and values.

## Relational Operator



# LET'S COMPARE SOME NUMBERS

```
# Equal to
print("Is num1 equal to num2?", num1 == num2)
# Not equal to
print("Is num1 not equal to num2?", num1 != num2)
# Greater than
print("Is num1 greater than num2?", num1 > num2)
# Less than
print("Is num1 less than num2?", num1 < num2)
# Greater than or equal to
print("Is num1 greater than or equal to num2?", num1 >= num2)
# Less than or equal to
print("Is num1 less than or equal to num2?", num1 <= num2)
```

# ASSIGNMENT OPERATORS

Assign(=)

Add and  
Assign(+=)

Subtract  
and  
Assign(-=)

Divide  
and  
Assign(/=)

Multiply  
and  
Assign(\*=)

Modulus  
and  
Assign(%=)

Exponent  
and  
Assign(\*\*=)

Floor-Divide  
and  
Assign(//=)



```
# Declare a variable
num = 10
print("Initial value of num:", num)

# Assign and add (+=)
num += 5 # Equivalent to num = num + 5
print("After num += 5:", num)

# Assign and subtract (-=)
num -= 3 # Equivalent to num = num - 3
print("After num -= 3:", num)

# Assign and multiply (*=)
num *= 2 # Equivalent to num = num * 2
print("After num *= 2:", num)
```

```
# Assign and divide (/=)
num /= 4 # Equivalent to num = num / 4
print("After num /= 4:", num)

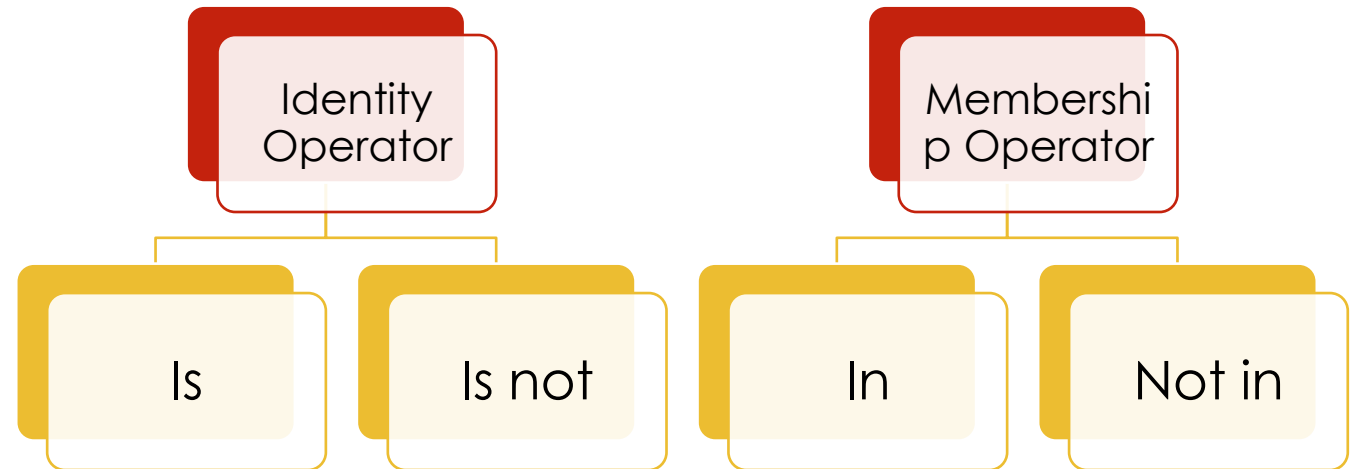
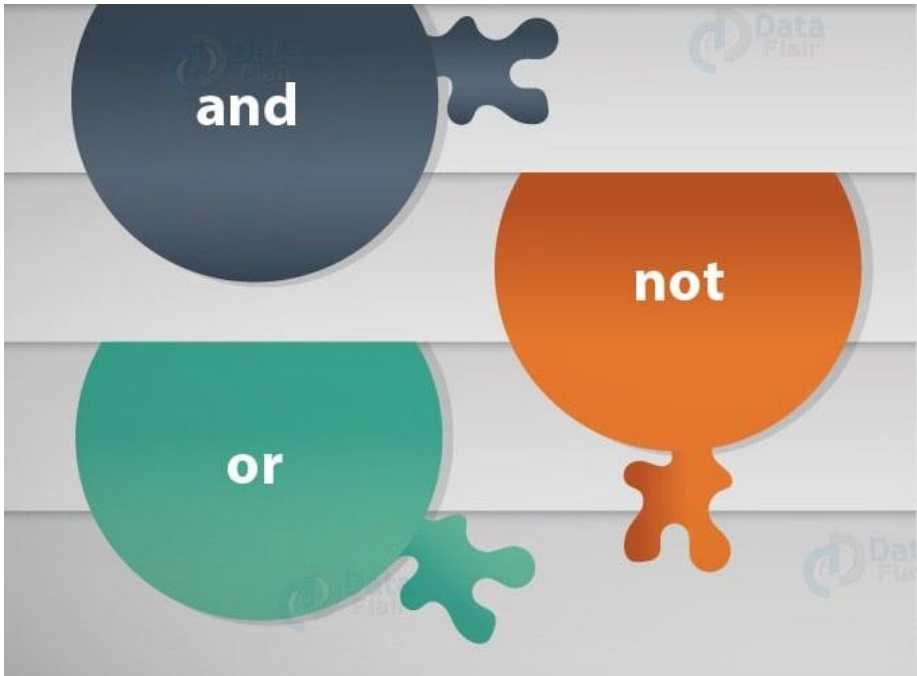
# Assign and modulus (%=)
num %= 3 # Equivalent to num = num % 3
print("After num %= 3:", num)

# Assign and exponentiation (**=)
num **= 2 # Equivalent to num = num ** 2
print("After num **= 2:", num)

# Assign and floor division (//=)
num //= 2 # Equivalent to num = num // 2
print("After num //= 2:", num)
```



# LOGICAL OPERATOR





# CONDITIONAL STATEMENTS

- If statement
- Elif Statement
- If else statement
- Ternary Operators
  - Short hand if
  - Short hand if else
- Nested If
- Pass statement

```
num = 10
if num > 0:
    print("Positive number")
```

IF

```
num = -5
if num > 0:
    print("Positive number")
else:
    print("Non-positive number")
```

IF Else

```
num = 0
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

IF elif else

```
num = 5
```

### Ternary (Shorthand) if Statement

```
print("Positive") if num > 0 else print("Non-positive")
```

```
num = -3
```

### Shorthand if else Statement

```
result = "Positive" if num > 0 else "Negative"  
print(result)
```

## Nested IF

```
num = 15
if num > 0:
    if num % 2 == 0:
        print("Positive and even number")
    else:
        print("Positive and odd number")
else:
    print("Non-positive number")
```

## Pass Statement

```
num = 5
if num > 0:
    # A placeholder for future code, avoids syntax errors
    pass
else:
    print("This won't execute")
```

# TASK 1: GRADE ASSIGNMENT

- Write a program to take marks (out of 100) as input from the user and assign grades based on the following conditions:
- **90-100:** Grade A
- **70-89:** Grade B
- **50-69:** Grade C
- **Below 50:** Fail



## TASK 2: NUMBER CATEGORIZER

- Write a program that takes a number as input and checks the following conditions:
- If the number is positive, check if it is odd or even.
- If the number is zero, print "It's zero."
- If the number is negative, check if it is less than -10.
- **Hint: you can use nested condition**

# TASK 3: DISCOUNT CALCULATOR

- Write a program to calculate the final price after a discount based on the purchase amount:
- **Above \$500:** 20% discount
- **Between \$200 and \$500:** 10% discount
- **Below \$200:** No discount
- Print the discount amount and the final price.



# LOOPS

- While loop
- For loop
- Break statement
  - With the break statement we can stop the loop even if the while condition is true.
- Continue Statement:
  - With the continue statement we can stop the current iteration, and continue with the next.
- Else statement loop:
  - With the else statement we can run a block of code once when the condition no longer is true.

## While Loop

```
count = 1
while count <= 5:
    print("Count:", count)
    count += 1
```

## For Loop

```
# Iterating over a range from 1 to 5
for num in range(1, 6):
    print("Number:", num)
```

## While with break

```
count = 0
while count < 5:
    if count == 3:
        print("Breaking at count =", count)
        break
    print("Count is", count)
    count += 1
```

## For with break

```
# Loop through numbers from 1 to 10
for num in range(1, 11):
    if num == 5:
        print("Breaking at number:", num)
        break
    print("Number:", num)
```

# ACTIVITIES

- Write a program that takes two numbers as input and prints the sum of all even numbers in the range between the two numbers (inclusive).
- Write a program to print the following pattern using nested loops.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

## Pattern Solution

```
# Outer loop for rows
for i in range(1, 6): # Rows from 1 to 5
    for j in range(1, i + 1): # Columns up to the current row number
        print(j, end=" ") # Print number with space
    print() # Move to the next line
```





# FUNCTIONS

- A **function** is a block of reusable code that performs a specific task. Functions in Python are defined using the `def` keyword.
- Functions can take input arguments, perform operations, and return results.
- Key points:
  - Functions reduce code duplication and improve readability.
  - Parameters can be passed to functions to customize their behavior.
  - Return values allow functions to send back results to the caller.
  - Default and keyword arguments provide flexibility when calling functions.
  - Recursive functions call themselves for repetitive operations.

## Simple Function

```
def greet(): # Function without parameters
    print("Hello, welcome to Python functions!")

# Function call
greet()
```

## Functions with Parameter

```
def greet_user(name): # Function with one parameter
    print(f"Hello, {name}! Welcome to Python functions.")

# Function call
greet_user("Alice")
```

## Multiple Parameters

```
def add_numbers(a, b): # Function with two parameters
    result = a + b
    print(f"The sum of {a} and {b} is {result}")

# Function call
add_numbers(5, 3)
```

## Default Parameter

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

# Function calls
greet("Alice") # Passing argument
greet()        # Using default value
```

## With Return value

```
def multiply_numbers(a, b):
    return a * b

# Storing the return value in a variable
result = multiply_numbers(4, 5)
print("The product is:", result)
```

## Passing arguments using keywords

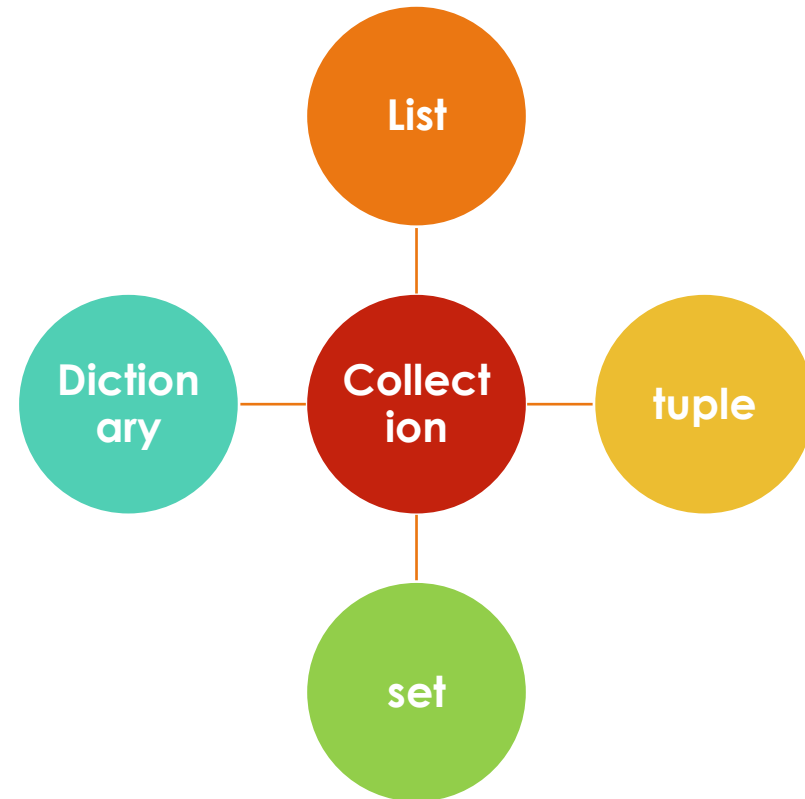
```
def describe_person(name, age, city):  
    print(f"{name} is {age} years old and lives in {city}.")  
  
# Calling using keyword arguments  
describe_person(age=25, city="New York", name="Bob")
```

## Recursive Function

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n - 1)  
  
# Function call  
result = factorial(5)  
print("Factorial of 5 is:", result)
```

# COLLECTIONS

- A group of items called collection.





# LIST

- Python lists are **ordered, mutable, and allow duplicate values**.
- The List is like a dynamic array, along with the linear sequence,
- It provides you with several built-in functions to perform various operations on data stored in it.
- We can store any number of items/elements in a list
  - homogeneous
  - heterogeneous

# CHARACTERISTICS OF LIST





# CREATING A LIST

```
# Creating a list of numbers
numbers = [10, 20, 30, 40, 50]
print(numbers)

# Creating a list with mixed data types
mixed_list = [10, "Python", 3.14, True]
print(mixed_list)

# Creating an empty list
empty_list = []
print(empty_list)
```

# ACCESSING A LIST

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

# Access first element
print(fruits[0])

# Access last element (negative index)
print(fruits[-1])

# Access a range of elements (Slicing)
print(fruits[1:3])
```

# MODIFYING LIST

```
## Adds an element at the end
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)

## Adds an element at a specific index
numbers.insert(1, 10)
print(numbers)

## Adds multiple elements to the list
numbers.extend([5, 6, 7])
print(numbers)
```

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

# Change an element
fruits[1] = "Blueberry"
print(fruits)
```

# REMOVE ELEMENTS

```
## Removes a specific element
fruits = ["Apple", "Banana", "Cherry"]
fruits.remove("Banana")
print(fruits)

## Removes an element by index
fruits.pop(0)
print(fruits)

## Deletes an element or the entire list
numbers = [10, 20, 30, 40]
del numbers[1]
print(numbers)
```

# ITERATION

```
## Using for loop
for num in numbers:
    print(num)

## Using while Loop
i = 0
while i < len(numbers):
    print(numbers[i])
    i += 1
```

# SOME MORE OPERATIONS

```
numbers = [10, 20, 30, 40]
## Checking if an Element Exists in a List
if 20 in numbers:
    print("20 is in the list!")

## Sort
numbers.sort()
print(numbers)

## Reverse
numbers.sort(reverse=True)
print(numbers)
```

```
## Copy List
copy_numbers = numbers.copy()
print(copy_numbers)

## Sort way to Create List
squares = [x ** 2 for x in range(1, 6)]
print(squares)
```



# TUPLE

- A **tuple** is an **ordered, immutable** collection in Python.
- It allows duplicate values.
- Unlike lists, tuples **cannot be modified** (no addition, removal, or update of elements after creation).
- Tuples are created using **parentheses ()**.

# CREATING TUPLE

```
# Creating a tuple
fruits = ("Apple", "Banana", "Cherry")
print(fruits)

# Tuple with different data types
mixed_tuple = (10, "Python", 3.14, True)
print(mixed_tuple)

# Creating a tuple with a single element (comma is required)
single_element = ("Hello",)
print(single_element)

# Empty tuple
empty_tuple = ()
print(empty_tuple)
```



# TYPE CASTING

```
# Convert tuple to list
numbers_tuple = (10, 20, 30)
numbers_list = list(numbers_tuple)

# Modify the list
numbers_list.append(40)

# Convert back to tuple
numbers_tuple = tuple(numbers_list)
print(numbers_tuple)
```

# SOME MORE OPERATIONS

```
# Concatenation
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = tuple1 + tuple2
print(result)
```

```
# Repetition
repeated_tuple = ("Python",) * 3
print(repeated_tuple)
```

```
print('length: ',len(result)) ## Finding length
print('Maximum: ',max(result)) ## Finding Maximum
print('Minimum: ',min(result)) ## Finding Minimum
print('Sum: ',sum(result)) ## Finding Sum
```

```
# Count occurrences of an element
print('2 Repeated: ',result.count(2)) # Output: 3
```

```
# Find index of the first occurrence
print('3 available at index',result.index(3))
```

# ADVANTAGES OF TUPLE OVER LIST



We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.

Since tuples are immutable, iterating through a tuple is faster than with a list. So there is a slight performance boost.

Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.

If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

# DICTIONARY

- A **dictionary** in Python is a **collection of key-value pairs** where each key is unique.
- It is **mutable**, meaning values can be updated, but keys must be **immutable (strings, numbers, or tuples)**.
- Dictionaries are defined using **curly braces {}** or the `dict()` function.
- **Fast lookups** compared to lists.
- Useful for **real-world applications** like **storing API responses, tracking data, and user information**.

## Working With Dictionary Collection

```
# Creating a dictionary with key-value pairs
student = {
    "name": "Alice",
    "age": 21,
    "course": "Computer Science",
    "grades": [85, 90, 78]
}

print(student)

## Accessing Dictionary
print(student["name"])
print(student.get("course"))

## Using get() is safer because it doesn't raise
## an error if the key doesn't exist.
print(student.get("address", "Not Available"))
```

## Working With Dictionary Collection

```
student["age"] = 22  # Updating an existing value
student["city"] = "New York"  # Adding a new key-value pair

print(student)

# removes a key and returns its value
removed_value = student.pop("city")
print(removed_value)

# Using del keyword
del student["age"]

# removes and returns the last inserted key-value pair
last_item = student.popitem()
print(last_item)

# removes all items
student.clear()
print(student)
```

## Iterate dictionary

```
employee = {"name": "John", "role": "Developer", "salary": 60000}

# for in loop
for key in employee:
    print(key, "->", employee[key])

# Using .items() method
for key, value in employee.items():
    print(f"{key}: {value}")
```

```
## Checking If a Key Exists
if "salary" in employee:
    print("Salary information is available.")
```





# SET

- A **set** in Python is an **unordered collection** of **unique elements**.
- Sets are **mutable**, but they **cannot contain duplicate values**.
- **Key Features:**
  - **Unordered:** The items have no fixed position.
  - **Unique Elements:** Duplicate values are automatically removed.
  - **Mutable:** We can add or remove elements.
  - **Supports Set Operations:** Union, Intersection, Difference, etc.

```
# Creating a set
names = {"alex", "bob", "catty", "devid", "john"}
print(names)
```

```
## Access
for name in names:
    print(name)
```

```
## Adding Single Element
names.add("jack")
print(names)
```

```
## Add multiple element
names.update(["gracy", "tracy"])
print(names)
```

```
## Element Remove
names.remove("sonam") # raises an error if not found
names.discard("sonam") # does nothing if not found
```

```
names.pop() # Removes a random element
print(names)
```

```
names.clear() # Removes all elements
print(names)
```

```
# Using set() function
numbers = set([1, 2, 3, 4, 4, 2, 5])
print(numbers)
```

# Set operations

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

```
B = {4, 5, 6, 7, 8}
```

```
# Union - Combines elements from both sets
```

```
print(A | B)
```

```
print(A.union(B))
```

```
# Intersection - Common elements
```

```
print(A & B)
```

```
print(A.intersection(B))
```

```
# Difference - Elements in A but not in B
```

```
print(A - B)
```

```
print(A.difference(B))
```

```
# Symmetric Difference
```

```
# Elements in either set, but not both
```

```
print(A ^ B) |
```

```
print(A.symmetric_difference(B))
```

# EMPLOYEE SALARY TRACKER

- **Scenario:** A company maintains employee names and their **salaries** in a dictionary.
  - Create a **dictionary** with 3 employees and their salaries.
  - Add a **new employee** with their salary.
  - Update an existing employee's salary.
  - Remove an employee from the dictionary.
  - Print all employee names and their salaries.

# MOVIE TICKET BOOKING SYSTEM

- You are building a **Movie Ticket Booking System** that helps track movies, theaters, available seats, and customer bookings.
- Store available movies in a list, where each movie is a dictionary {title, genre, showtime}.
- Maintain a set of unique theaters (to avoid duplicates).
- store ticket pricing details as a tuple (since prices are fixed).
- create a dictionary of customers, where the key is the customer's name and the value is a list of booked movies.



# TASK TO PERFORM ON DATA

- Add a new movie to the movie list.
- Remove a movie from the list.
- Add a new theater to the set.
- Add a new customer to the dictionary.
- Allow a customer to book a movie ticket (update their list of booked movies).
- Print all movies, theaters, ticket pricing, and customer bookings.