# Introduction To Python

Lec-1

# What is Python ??

- Python is a **high-level, interpreted, general-purpose programming language** known for its **simplicity, readability, and versatility**.
- It allows you to write programs that are clear and logical even for complex problems.
- Python's **easy-to-learn syntax** makes it an **ideal choice for beginners** while its **powerful libraries and frameworks** support advanced applications in various domains like **web development, data science, and automation.**
- Python supports multiple programming paradigms, **including procedural, object-oriented, and functional programming**, making it a flexible tool for different programming needs and styles.

# A Brief History of Python

1

**Late 1980s**

Python was conceived by **Guido van Rossum**, a Dutch programmer, at CWI (Centrum Wiskunde & Informatica) in the Netherlands.

2

**1991**

First version **Python 0.9.0** was released. It already had exception handling, functions, and modules.

3

**2000**

**Python 2.0** came with new features like garbage collection and Unicode support.

4

**2008**

**Python 3.0** was released — not backward-compatible with Python 2, but designed for cleaner, modern syntax.

5

**2020**

**Python 2 officially retired**; now only **Python 3** is **maintained and developed.**

As of today, Python is one of the **most popular programming languages** worldwide — used by **Google, Netflix, NASA, YouTube**, and many others.

### 💡 Fun Fact:

Guido van Rossum named it **“Python”** after the British comedy show *“Monty Python’s Flying Circus”*, not after the snake 🐍

# Why to learn Python ??

1. **Easy to Learn:** Python has a simple and straightforward syntax, making it an excellent language for beginners.

2. **Versatility:** Python is a versatile language and is used in a wide range of applications- web development, scientific computing, data analysis, artificial intelligence and more.

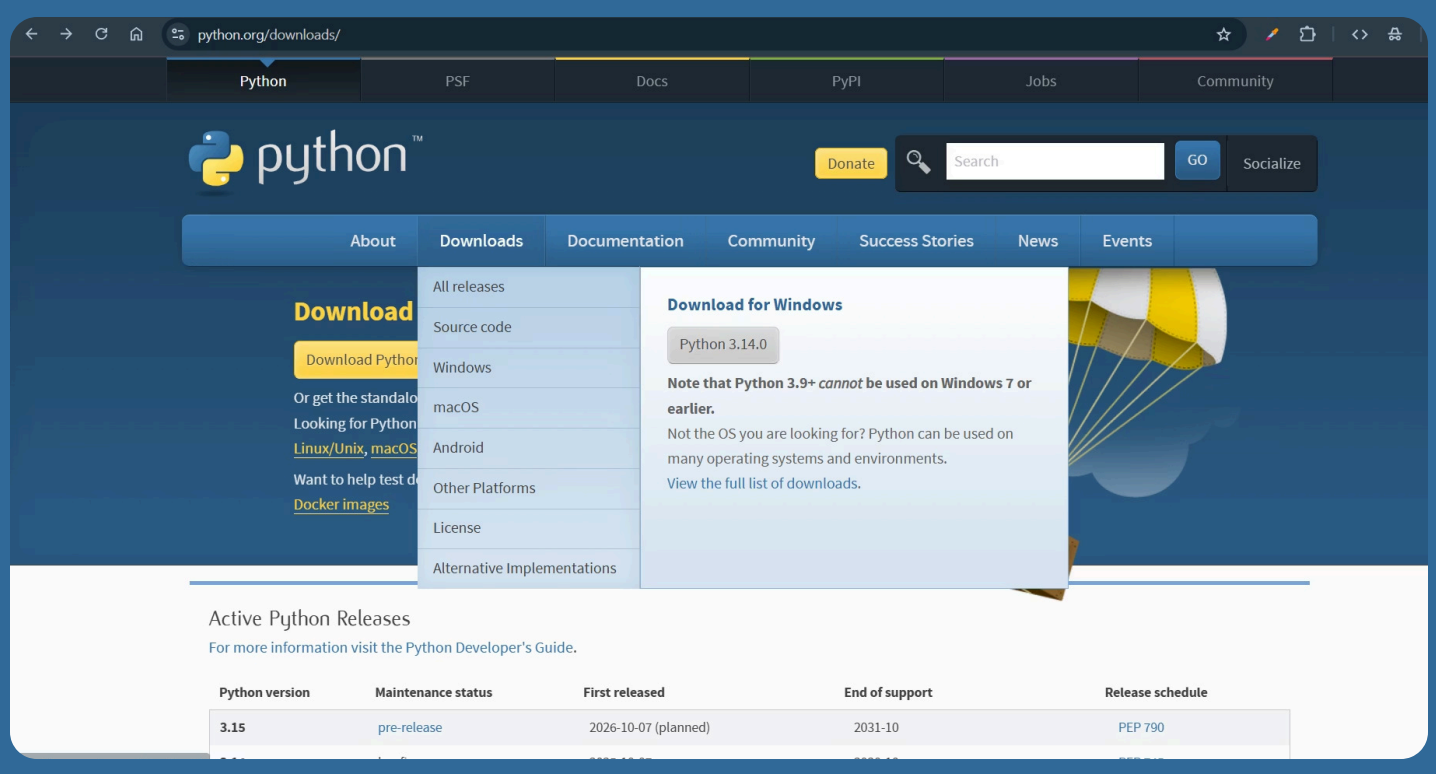
3. **Large and Active Community:** Python has a large and active community of users and developers, which means that there is a wealth of resources and support available.

4. **Good for rapid prototyping:** Python's simplicity makes it possible to quickly test ideas and iterate on them, saving time and resources compared to compiled languages.

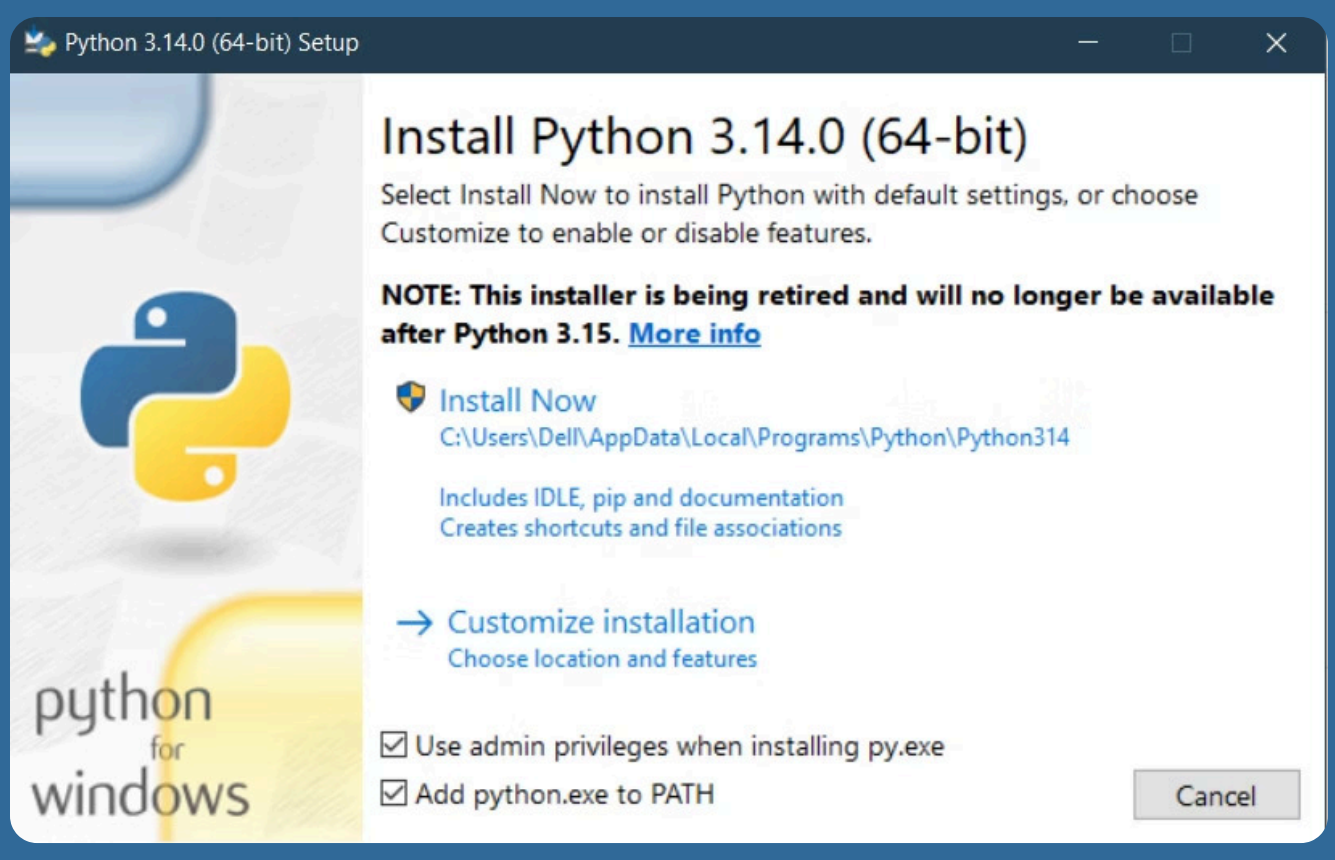
5. **Job Opportunity:** Python has high demand in the job market, particularly in fields of data science, machine learning, web development, genAi and Agentic Ai development.

# Downloading and Installing Python

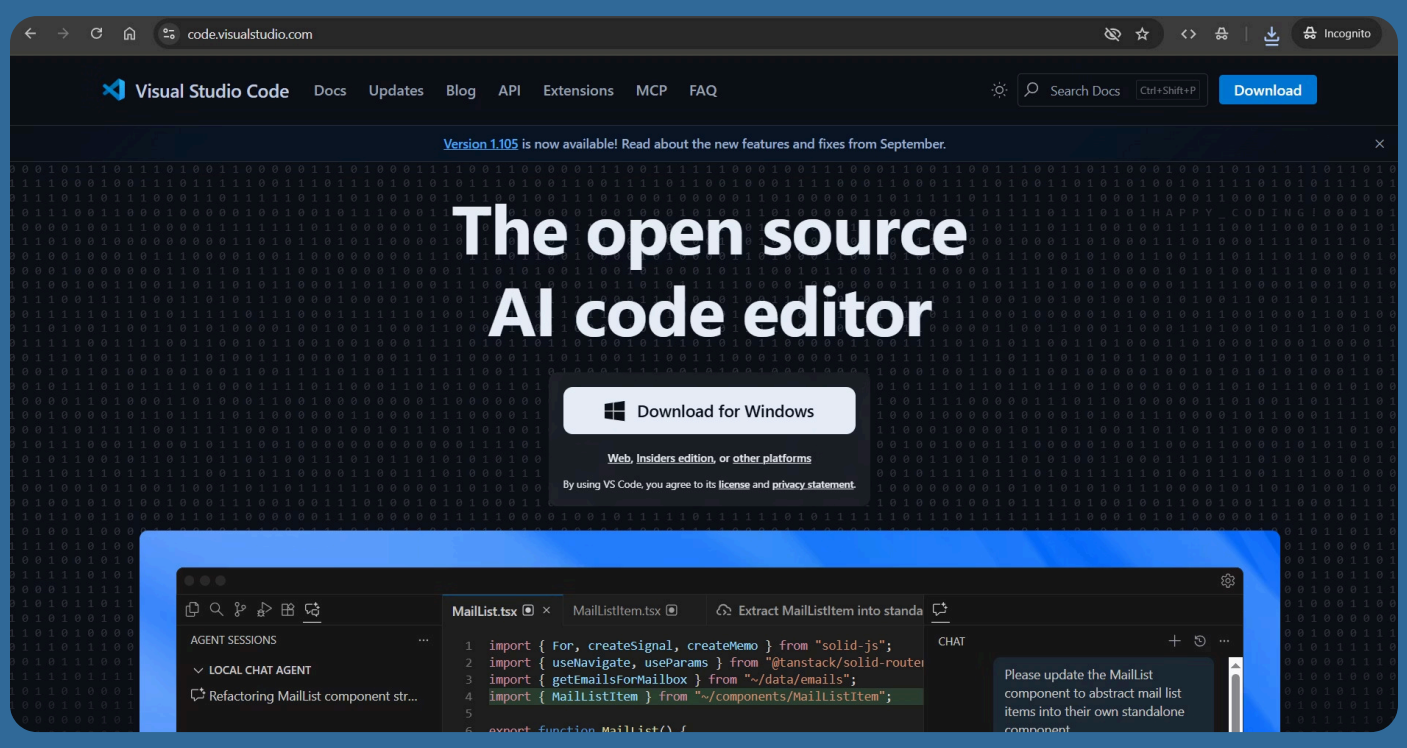
- **Step-1:** Visit- **Download Python**
- **Step-2:** Hover over **Downloads** option and download the latest version according to your OS



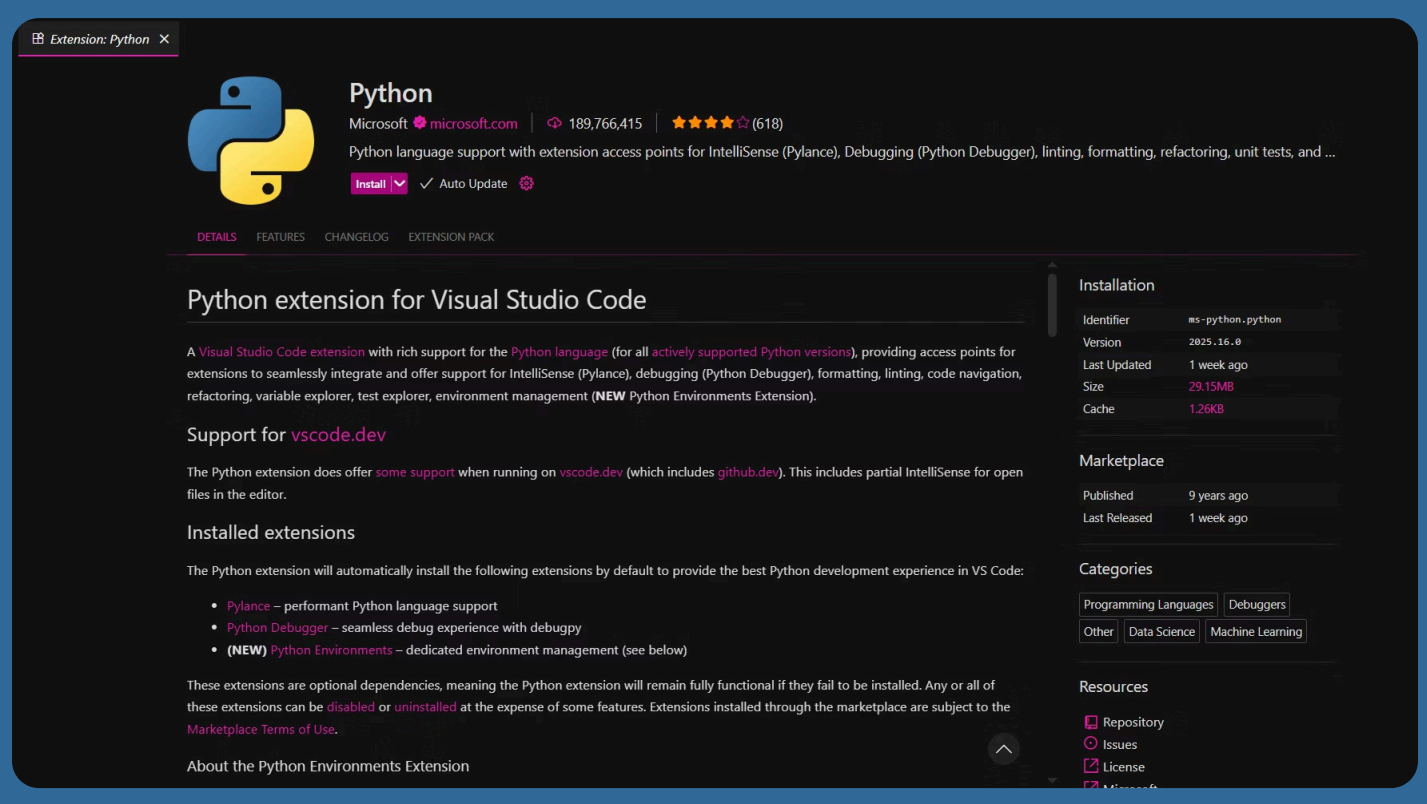

- **Step-3:** Click on **Install Now**




- **Step-4:** Go to **https://code.visualstudio.com/** , download and install it.




- **Step-5:** Download the following VS Code extension for Python in VS Code.

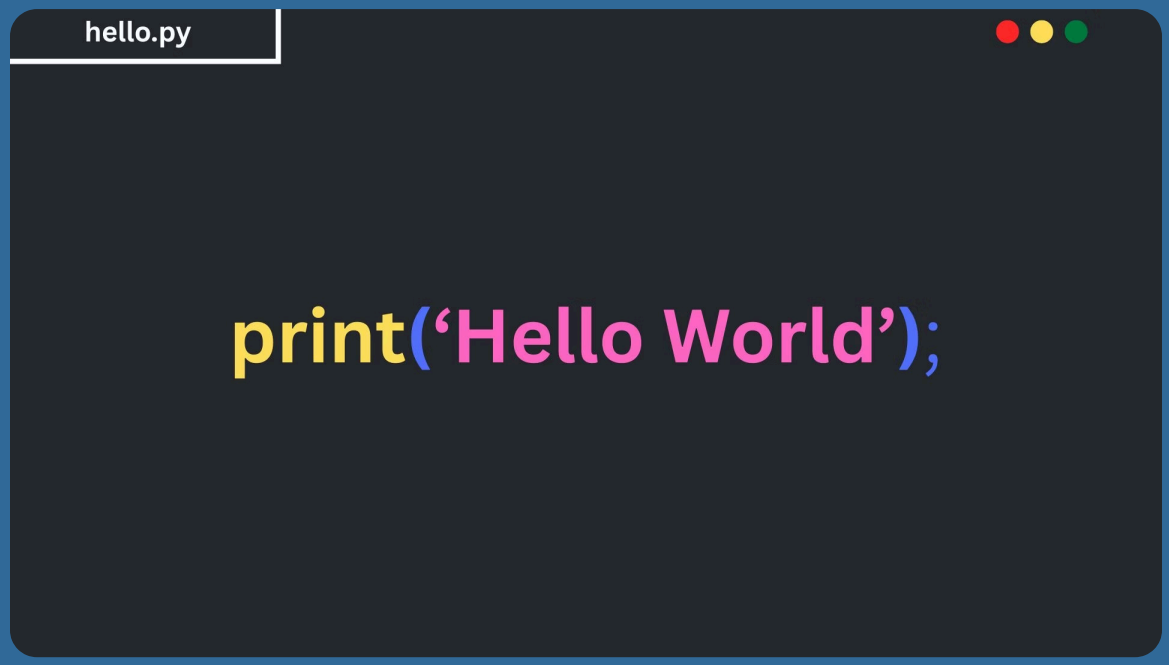



- **Step-6:** Open **Command Prompt** and the run the below command in it, if it results the following way then your python installation is successful.

```
C:\Users\Dell>python --version
Python 3.14.0
```

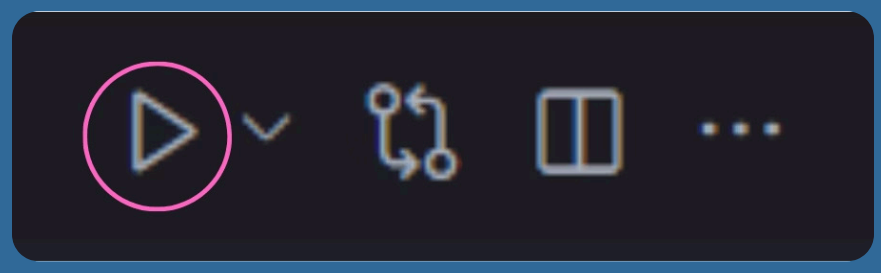
# Hello World in Python

- To print **"Hello World"** in Python, **open VS Code**, create a **new file** with a **.py** extension and type the following line in it.

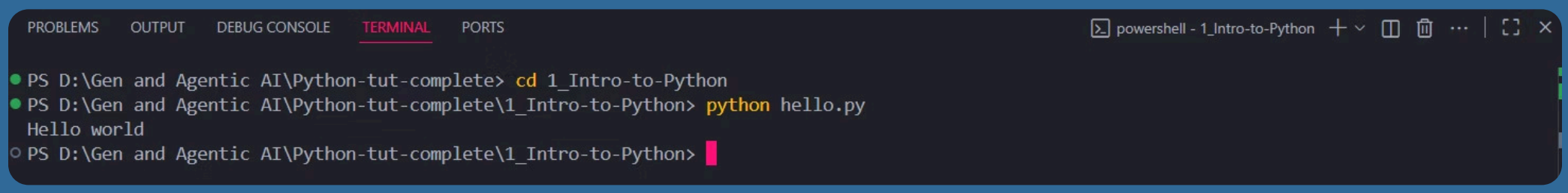



- To run Python program in Vs code there are 3 ways:

## 1. Use Run Button on Right Top Corner of VS Code editor



## 2. Inside terminal by running the command in following way



# Inner Working of Python

- Following is the inner working of Python.

- At very first step our Python code **(inside .py file)** get's compiled into **Byte Code**, which is **low level** and **platform independent.**
- The only reason we can use python in **cloud based platform** is **Byte Code being platform independent**.
- Byte code runs faster.
- Byte code **is not a Machine code.**
- It is a **Python specific interpretation.**
- It has following components:

1. **.PYC: Compiled Python files (frozen binaries)**
2. **__pycache__: Folder used to organize frozen binaries.**

- Frozen Binaries are generated only for imported files, not for top level files.

## Python Virtual Machine (PVM)

- It is a Code loop to iterate byte code.
- It is a Run Time Engine also known as python interpreter.

# Datatypes

- Python has vast collection of datatypes- **Text type, Numeric type, Sequence type, Mapping type, Set type, Boolean type, Binary Type, etc.**

## Numeric Datatype

- It consists of following datatypes:

1. **Integers**
2. **Float**
3. **Complex**

- **Eg:**

```
int1 = 20
flt1 = 12.2

print(int1) # will print 20

print(flt1) # will print 12.2
```

## Arithmetic Operators

| Operator | Name | Example |
|---|---|---|
| + | Addition | x + y |
| - | Subtraction | x - y |
| * | Multiplication | x * y |
| / | Division | x / y |
| % | Modulus | x % y |
| ** | Exponentiation | x ** y |
| // | Floor Division | x // y |

- **Eg:**

```
num1 = 20
num2 = 40

print(num1 + num2) # will print 60

print(num2 - num1) # will print 20

print(num1 * num2) # will print 800

print(num2 / num1) # will print 2.0

print(num2 // num1) # will print 2

num3 = 8
num4 = 3

print(num3 % num4) # will print 2

print(num3 ** num4) # will print 512
```

## Casting in Python

- Casting is basically **specifying a type onto a variable.**
- Casting in python is done by using **Constructor Functions** such as- **int(), float(), bool(), str().**
- **Eg:**

```
var1 = 20

print(var1)   # will print 20

print(type(var1))  # will print <class 'int'>

print(str(var1))   # will print 20

print(type(str(var1))) # will print <class 'str'>
```

# Datatypes

## Boolean Datatype

- Boolean represents one of two values- **True** or **False**.
- **Eg:**

```
isTrue = 0
print(bool(isTrue)) # will print False

isTrue = 1
print(bool(isTrue)) # will print True

isTrue = 'John'
print(bool(isTrue)) # will print True

isTrue = None
print(bool(isTrue)) # will print False
```

## Comparison Operators

| Operator | Name | Example |
|---|---|---|
| == | Equal | x == y |
| != | Not equal to | x ≠ y |
| > | Greater Than | x > y |
| < | Less Than | x < y |
| > = | Greater Than Equal To | x >= y |
| < = | Less Than Equal To | x <= y |

- **Eg:**

```
var1 = 10
var2 = 6

print(var1 == var2) # will print False

print(var1 != var2) # will print True

print(var1 > var2) # will print True

print(var1 < var2) # will print False

print(var1 >= var2) # will print True

print(var1 <= var2) # will print False
```

## Logical Operators

| Operator | Description | Example |
|---|---|---|
| and | Returns True if both statements are true | x < 5 and x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 or x < 4) |

- **Eg:**

```
var1 = 10
var2 = 6

print(var1 > var2 and var1 == var2) # will print False

print(var1 > var2 or var1 == var2) # will print True

print(bool(not(var1))) # will print False
```

# Datatypes

## Strings in Python

- Strings in python are **surrounded by** either **single quotation marks**, or **double quotation marks**.
- **Eg:**

```
str1 = 'Hello'
print(str1) # will print 'Hello'

str2 = "Hello World"
print(str2) # will print 'Hello World'
```

- To get length of an string, we can use **len()** function.
- **Eg:**

```
str1 = "Hello py"

print(len(str1)) # will print 8
```

## Indexing in Strings

```
str1 = "Hello"

print(str1[2]) # will print 'l'

print(str1[4]) # will print 'o'
```

## String Slicing

- **Note: Slicing excludes the end index (slices up to but not including it).**

```
str2 = 'Brother'
print(str2[3:6]) # will print 'the'

# Without Start Index
print(str2[0:3]) # will print 'bro'

# Without Ending Index
print(str2[4:]) # will print 'her'
```

- **Note: During negative indexing slicing excludes the start index (start slicing from it but not including it).**

```
# Negative Indexing
str3 = 'Programmer'

print(str3[-7:-3]) # will print 'gram'

print(str3[:-3]) # will print 'Program'

print(str3[-7:]) # will print 'grammer'
```

# Datatypes

## String Concatenation

- To concatenate, or combine, two strings you can use the **+ operator**.
- **Eg:**

```
str4 = 'Wel'
str5 = 'come'
print (str4 + str5) # will print 'Welcome'
print (str4 + " " + str5) # will print 'Wel come'
```

## Formatted Strings

- In python, we **cannot combine variables(with type other than strings)** with our Strings.
- **Eg:**

```
age = 25
print('My age is' + age) # This results in TypeError
```

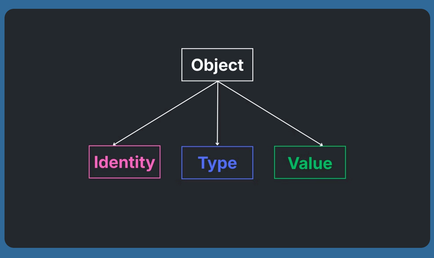
- So in this case **we make use of formatted strings.**
- We initiate formatted string by using a **small "f"** before writing our string.

```
print(f'My age is: {age}') # This will print "My age is 25"
print(f"My age after 5 years will be: {age + 5}) # My age after 5 years will be 30.
```

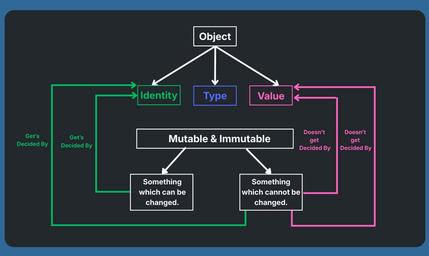
# Mutable and Immutable

- **Everything is Object in Python.**
- **Mutability** means the **ability of an object to get modified inside the memory** after it's creation.
- Every Object has the following properties:

1. **Identity**
2. **Type**
3. **Value**




- Always Remember it is the **Identity of an object** which helps us determine whether the **object is Mutable or Immutable.**




## Immutable

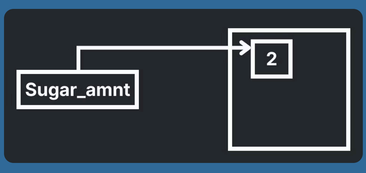
- **Eg:**

python:

```
sugar_amnt = 2
print(f"Initial Sugar: {sugar_amnt}")
```

output:

Initial Sugar: 2




python:

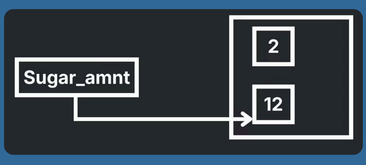
```
sugar_amnt = 2
print(f"Initial Sugar: {sugar_amnt}")

sugar_amnt = 12
print(f"Second Initial Sugar: {sugar_amnt}")
```

output:

Initial Sugar: 2

Second Initial Sugar: 12




- When we say **Mutability doesn't get decided by value**, we mean to say that **when we assign different value to a same variable** the **reference to that variable inside the memory get's changed** and the old value still remains inside the memory as unreferred.

python

```
print(f"Id of 2: {id(2)}")
```

output:

Id of 2: 140716054574**232**

python

```
print(f"Id of 12: {id(12)}")
```

output:

Id of 12: 140716054574**552**

- So the above object was **immutable** i.e. **numbers are immutable.**

## Mutable

- **Eg:**

python:

```
spice_mix = set()
print(spice_mix)
print(f"Initial spice mix id: {id(spice_mix)}")
```

output:

set()

Initial spice mix id: 2139208714**496**


python:

```
spice_mix.add("Cardamom")
print(spice_mix)
print(f"After spice mix id: {id(spice_mix)}")
```

output:

{'Cardamom'}

After spice mix id: 2139208714**496**


python:

```
spice_mix.add("Ginger")
print(spice_mix)
print(f"After spice mix id: {id(spice_mix)}")
```

output:

{'Cardamom', 'Ginger'}

After spice mix id: 2139208714**496**