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.

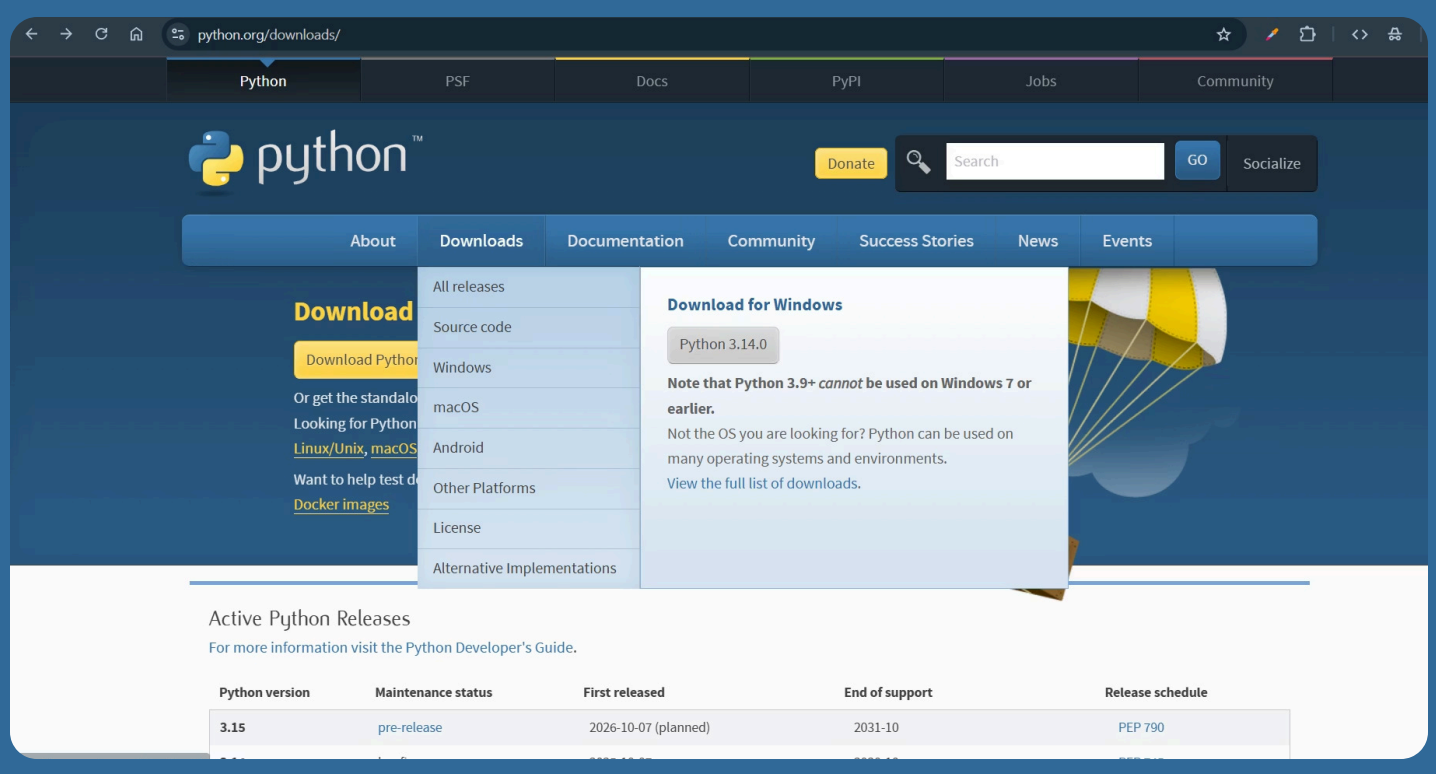
3. **Large and Active Community:** Python has a large and active community of users and developers, which means that there is a wealth of resources and support available.

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.

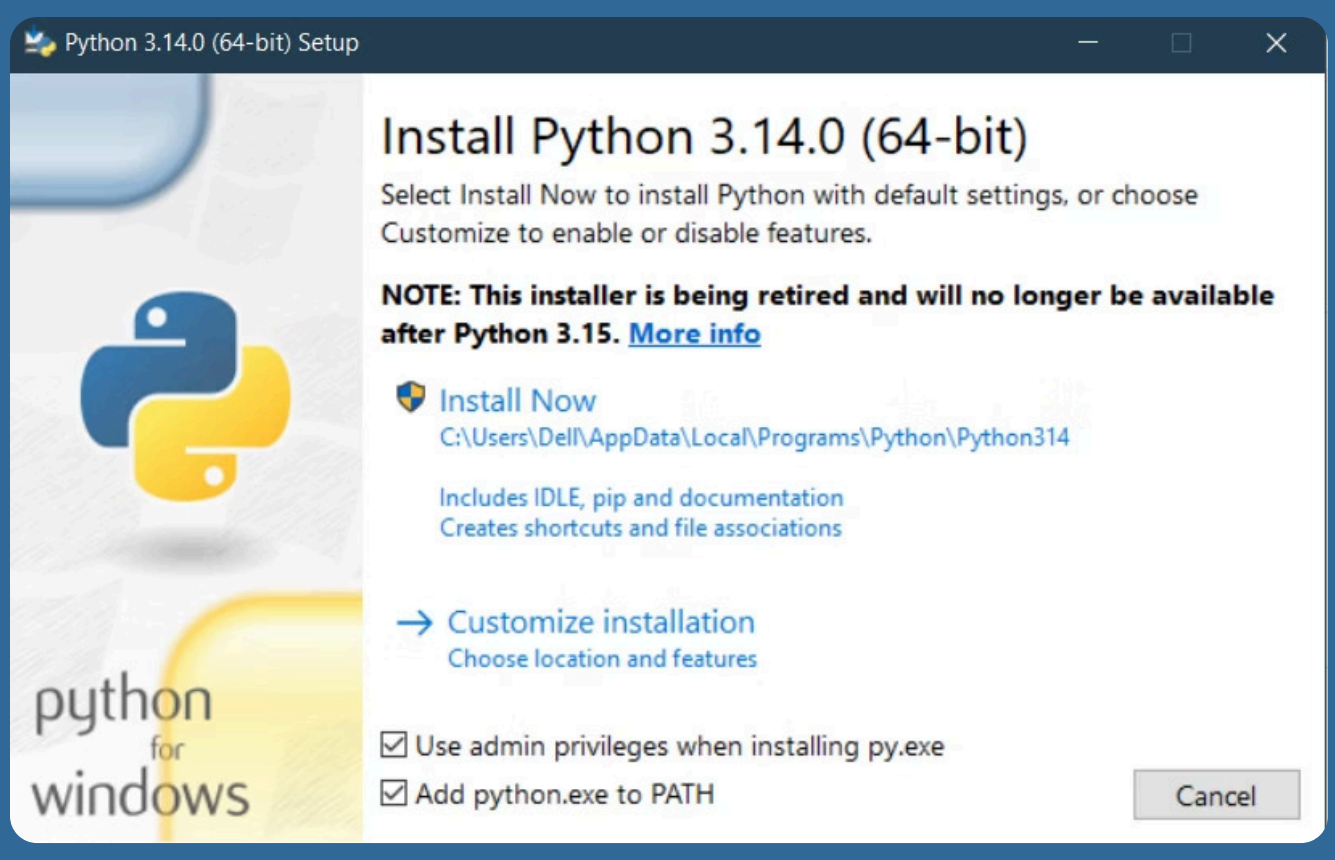
5. **Job Opportunity:** Python has high demand in the job market, particularly in fields of data science, machine learning, web development, genAi and Agentic Ai development.

Downloading and Installing Python

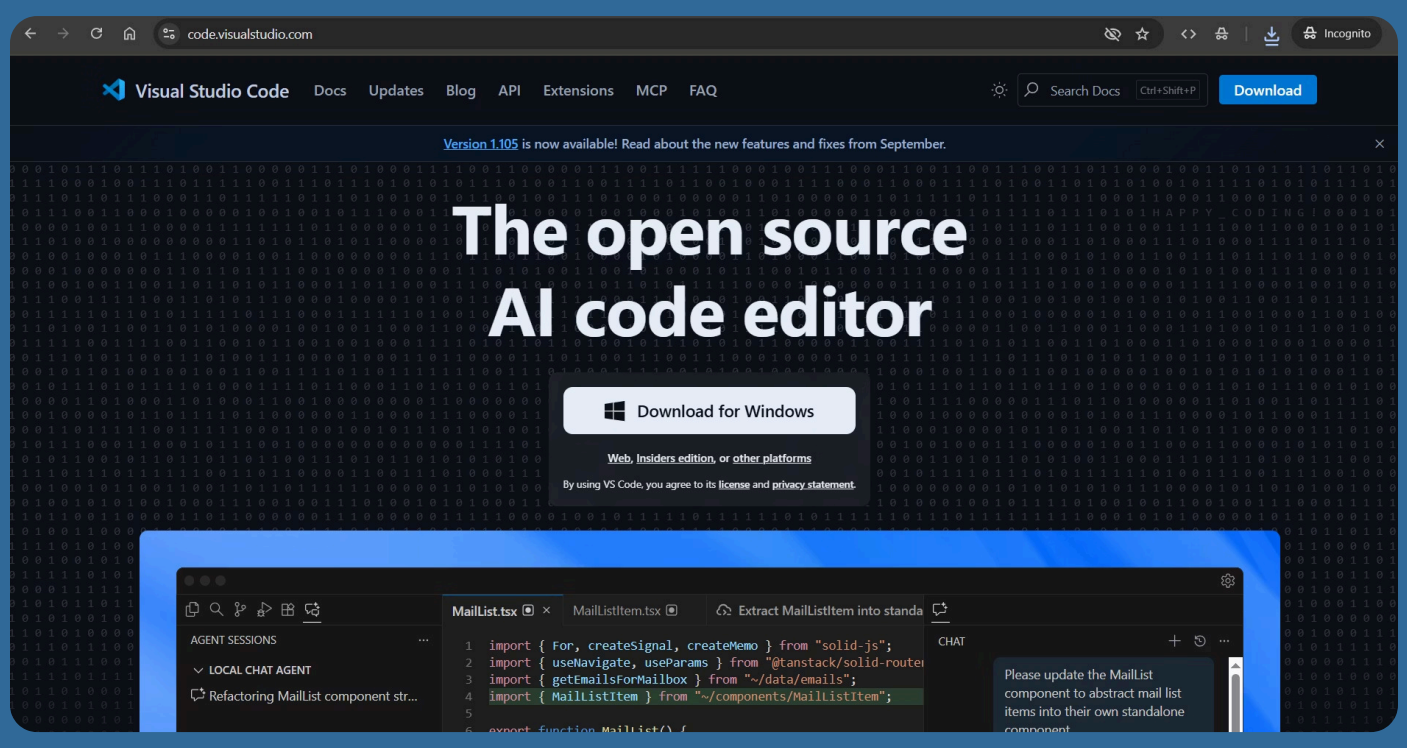
- **Step-1:** Visit- [Download Python]
- **Step-2:** Hover over **Downloads** option and download the latest version according to your OS



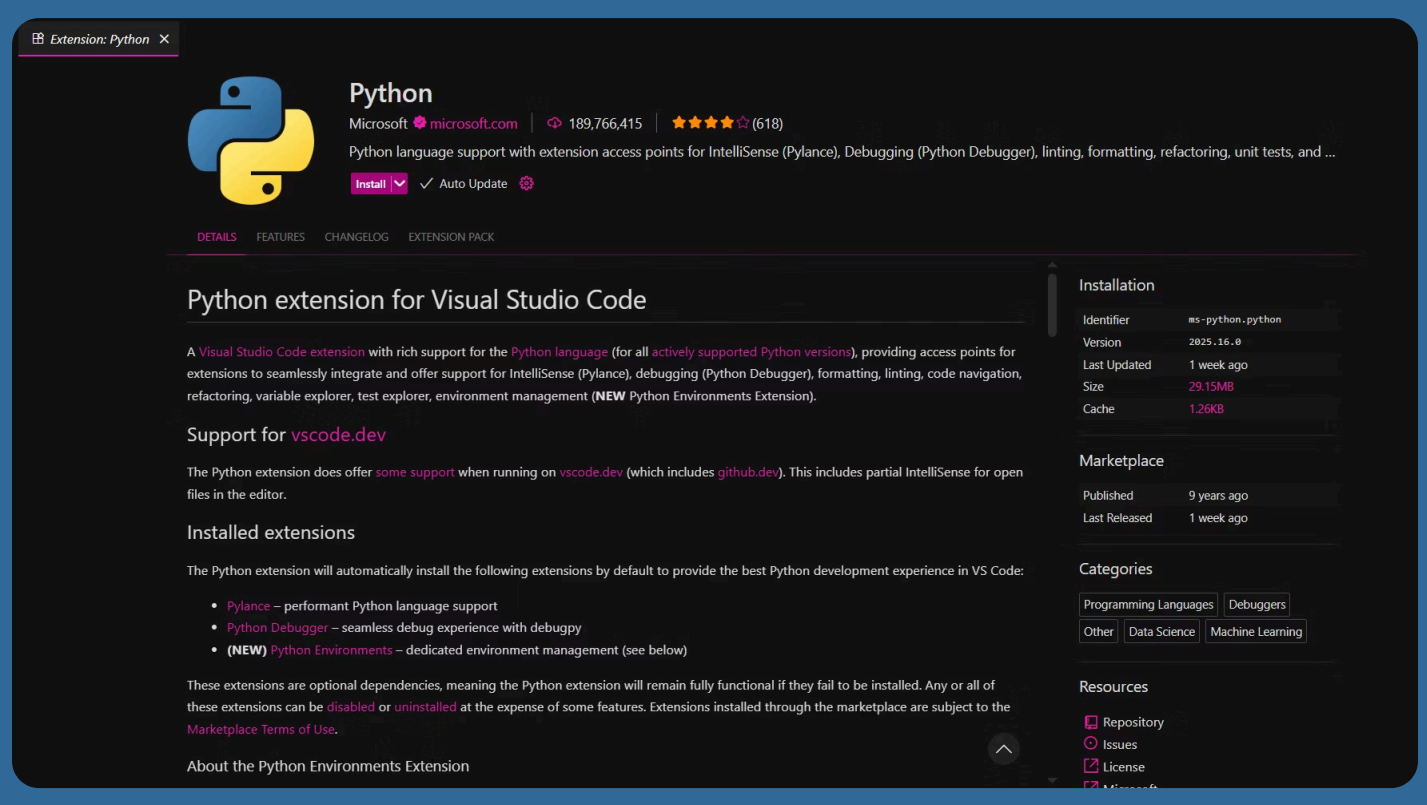
- **Step-3:** Click on **Install Now**



- **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.

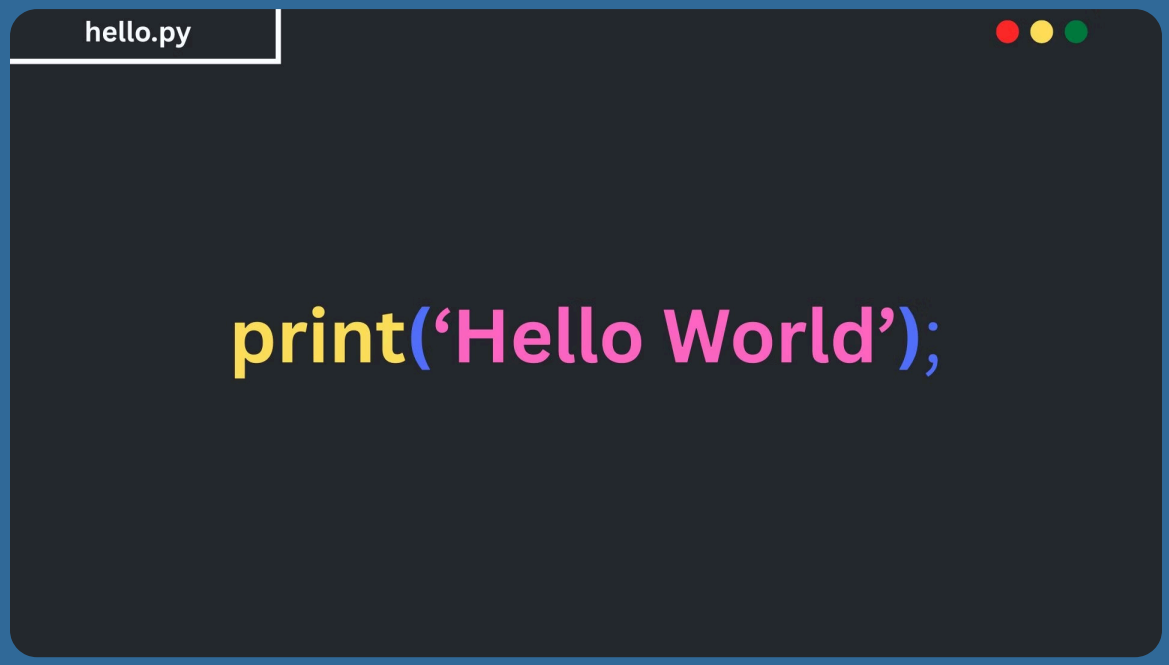


- **Step-6:** Open **Command Prompt** and the run the below command in it, if it results the following way then your python installation is successful.

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

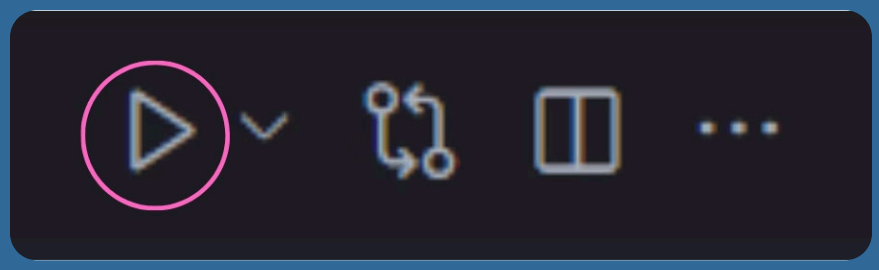
Hello World in Python

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

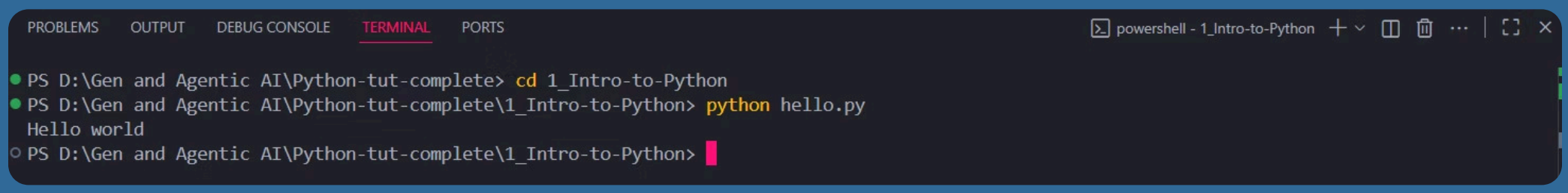


- To run Python program in Vs code there are 3 ways:

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



2. Inside terminal by running the command in following way



Single and Multi Line Comments

- In Python, **comments are used to add explanatory notes within the code**, which are **ignored by the interpreter** during execution.
- Following is the way to give **Single Line Comment**:

```
# Single Line Comment
```

- Following is the way to give **Multi Line Comment**:

```
'''
 Multi Line Comment
'''
```

- Python **does not really have a syntax for multiline comments** but Python will **ignore string literals that are not assigned to a variable**, you can add a multiline string (triple quotes) in your code, and place your comment inside it:
- As long as the string is not assigned to a variable, **Python will read the code, but then ignore it, and you have made a multiline comment.**

Data Types in Python

- Data Types are used to define **Type of a Data** stored inside any variable.
- In Python **built-in data types** are categorized in the following way:

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

Variables in Python

- Variables are **named memory locations** used to **store information within a program**.

Creating Variables

- In Python, you create a variable **just by assigning it something.** No need to declare it beforehand.

```
a = 10
d = "Joe"
print(a) # will print 10
print(d) # will print Joe
```

- Variable names are **Case Sensitive.**

```
a = 4
A = "Borris"
#This will create 2 different variables and a will not override A.
```

- We **don't have to declare a particular datatype while declaring a variable**, variables **automatically adapt to the type of data they hold**, and **can be reassigned to data of any other type.**

```
x = 20 # x is now a variable of type int
x = 'Harry' # x is now a variable of type string
print(x)
```

- String variables can be **declared either by using single or double quotes.**

Casting

- If you want to **specify the data type of a variable**, this can be done with casting using **constructor function.**

```
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

Get the type of variable

- We can get the actual data type of any variable using **type()** function.

```
name = 'Donald'
empid = 100
print(type(name)) # will print <class 'str'>
print(type(empid)) # will print <class'int'>
```

Operators in Python

- Operators allow you to manipulate variables and values.
- In Python, operators are of following types:

1. **Arithmetic Operators**
2. **Assignment Operators**
3. **Comparison Operators**
4. **Logical Operators**
5. **Identity Operators**
6. **Membership Operators**
7. **Bitwise Operators**

Arithmetic Operators

Operator	Name	Use
+	Addition	Returns addition
-	Subtraction	Returns difference
*****	Multiplication	Returns product
/	Division	Returns value of Quotient
%	Modulus	Returns value of Remainder
******	Exponential	Returns Exponential value
//	Floor Division	Returns value of **Remainder** and **rounds down the result to nearest whole number.**

Operators in Python

Assignment Operators

Operator	Example	Use
=	x = 5	x = 5
+=	x+=3	x = x + 3
-=	x-=3	x = x - 3
*=	x*=3	x = x * 3
/=	x/=3	x = x / 3
%=	x%=3	x = x % 3
**	x**=3	x = x ** 3
//	x//=3	x = x // 3

Comparison Operators - Returns boolen result

Operator	Name	Use
==	Equal to	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

Operators in Python

Logical Operators - Returns Boolean Result

Operator	Example	Use
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 and x < 10)

Identity Operators - Returns Boolean Result

- Identity operators are used to **compare the objects**, not if they are equal, but if **they are actually the same object, with the same memory location.**

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Eg:

```
x = ['Apple', 'Banana']
y = ['Apple', 'Banana']
z = x
print(x is z) # returns true
print(x is y) # returns false
print(x == y) # returns true
```

Operators in Python

Membership Operators - Returns Boolean Result

- Membership operators are used to **test if a sequence is presented in an object**.
- It is **Case Sensitive.**

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Eg:

```
fruits = ["apple", "banana", "cherry"]
print("banana" in fruits) # will print true
print("strawberry" in fruits) # will print false
```

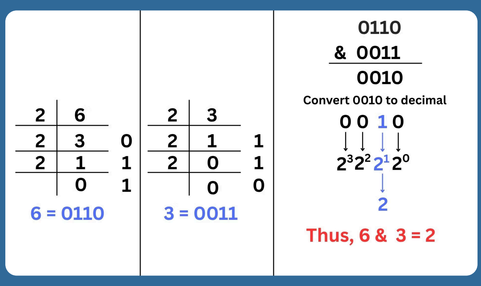
Bitwise Operators -

- Bitwise operators are **used to compare (binary) numbers.**

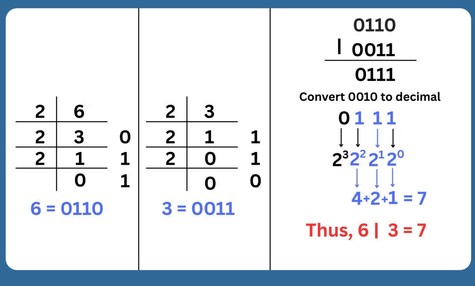
Operator	Name	Description	Example
&	AND	sets each bit to 1 if both bits are 1, else sets 0.	x & y
|	OR	Sets each bit to 1 if one of two bits is 1, else sets 0	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1, else 0	x ^ y
~	NOT	Inverts all the bits	~x

Eg:

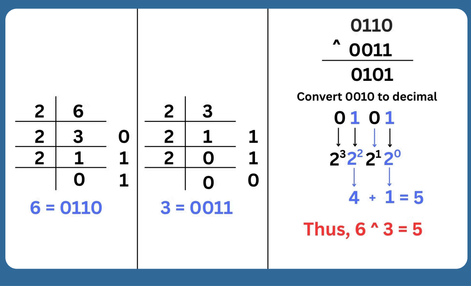
```
a = 6
b = 3
print(a & b) # will return 2
```



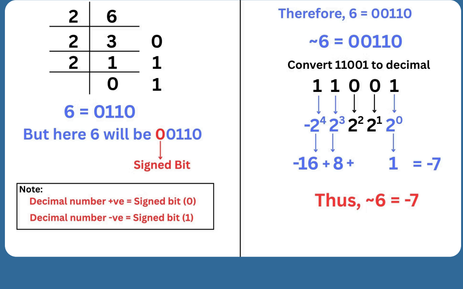

```
a = 6
b = 3
print(a | b) # will return 7
```




```
a = 6
b = 3
print(a ^ b) # will return 5
```




```
a = 6
print (~a) # will return -7
```



Operators in Python

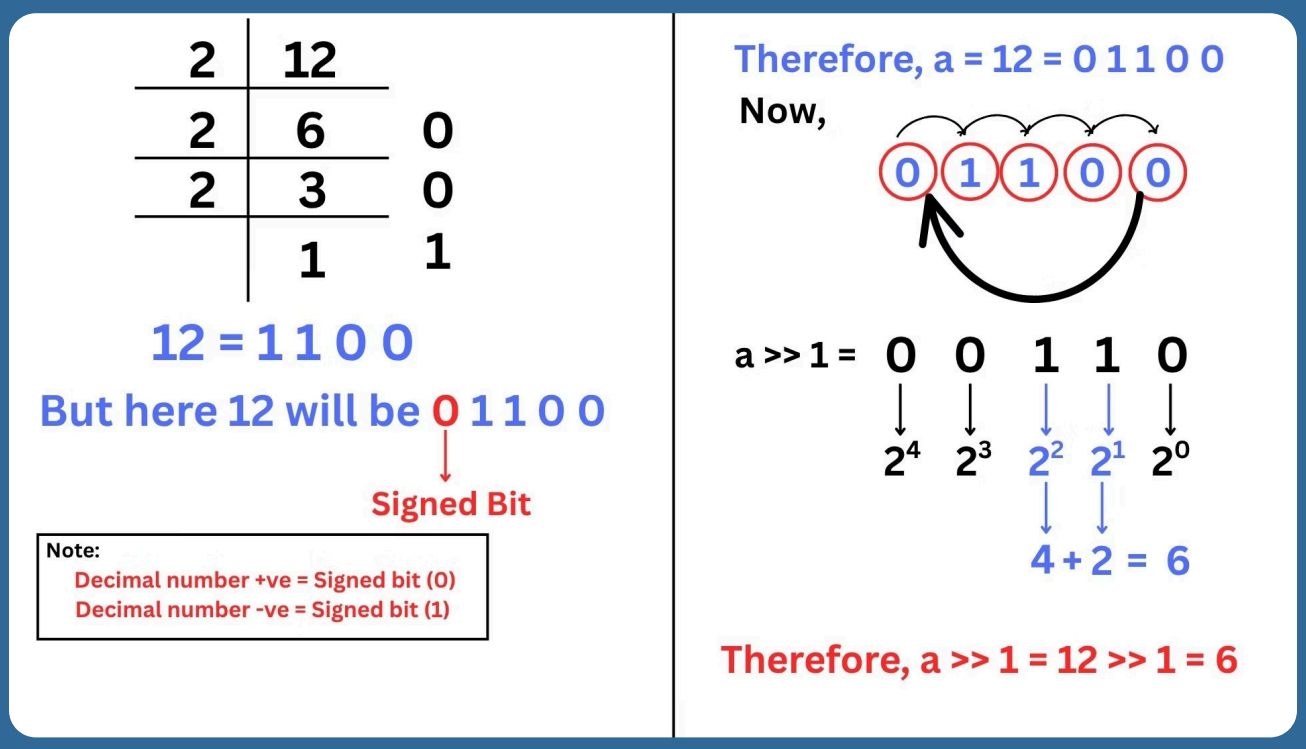
Bitwise Right Shift Operator (>>)

- **Shifts** the **bits** of a binary number towards the **right**.
- As a consequence, it divides the number by 2^n.

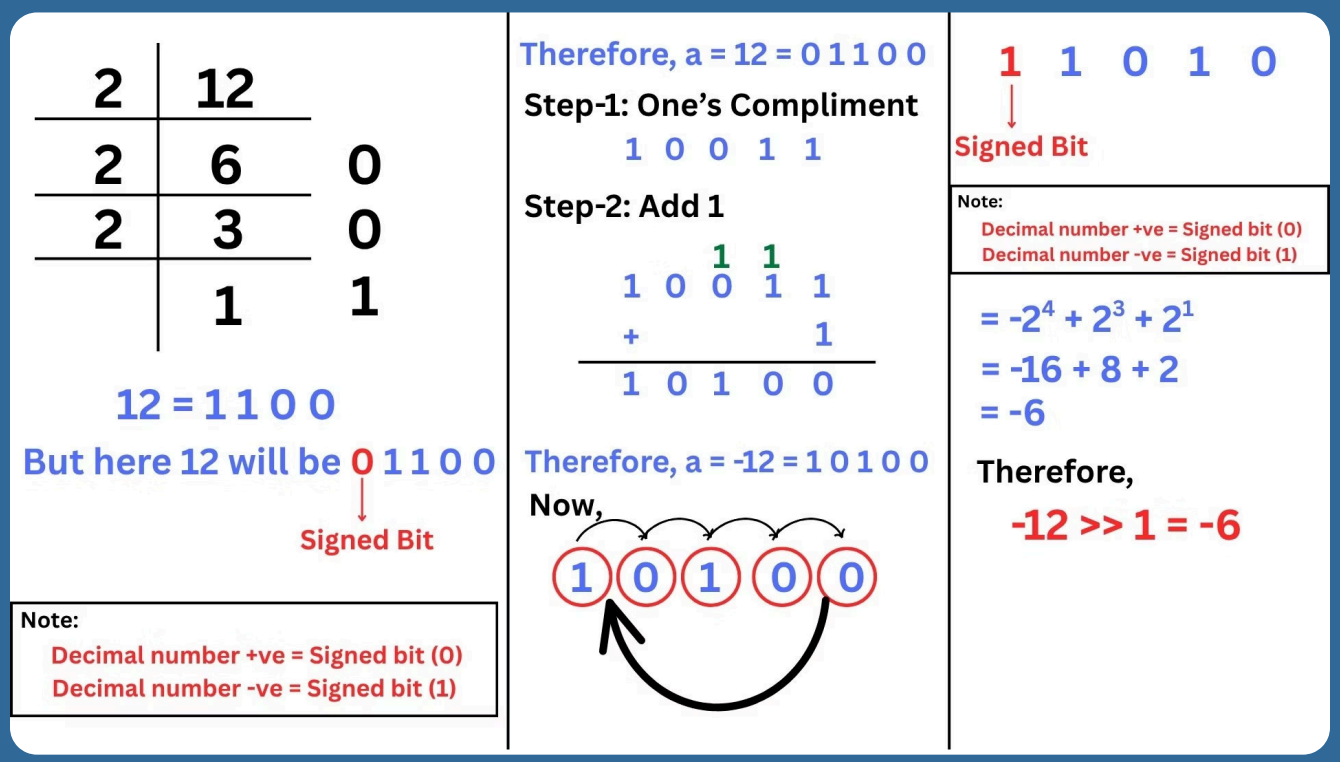
Where **n = magnitude of shift.**

Eg:

```
a = 12
print (a>>1) # will print 6 i.e. 12/2^n = 12/ 2^1 = 12/2 = 6
```




```
a = -12
print (a >> 1) # will print -6
```



Operators in Python

Bitwise Left Shift Operator (<<)

- **Shifts** the **bits** of a binary number towards the **Left**.
- As a consequence, it divides the number by 2^n.

Where **n = magnitude of shift.**

Eg:

```
a = 3
print (a << 1) # will print 6
```

Operator Precedence

- Mnemonic to Remember Python Operator Precedence

“Parents Eat Muffins After Riding Angry Tiger Sleds Calmly”

Breakdown:

- **Parents** → Parentheses `()`
- **Eat** → Exponents `**`
- **Muffins** → Multiplicative `*, /, //, %`
- **After** → Additive `+, -`
- **Riding** → Relational `<, >, <=, >=`
- **Angry** → Boolean `and`
- **Tiger** → Boolean `or`
- **Sleds Calmly** → Assignments `=, +=, -= ...`

Precedence (High → Low)	Operators	Category
1	()	Parentheses / Grouping
2	**	Exponentiation
3	+x, -x, ~x	Unary Operators
4	*, /, //, %	Multiplicative
5	+, -	Additive
6	<<, >>	Bitwise Shift
7	&	Bitwise AND
8	^	Bitwise XOR
9	|	Bitwise OR
10	<, <=, >, >=, !=, ==	Comparison
11	not	Boolean NOT
12	and	Boolean AND
13	or	Boolean OR
14	=, +=, -=, *=, /=, //=, %=, **=, &=,	=, ^=