



VCube Software Solutions Pvt Ltd

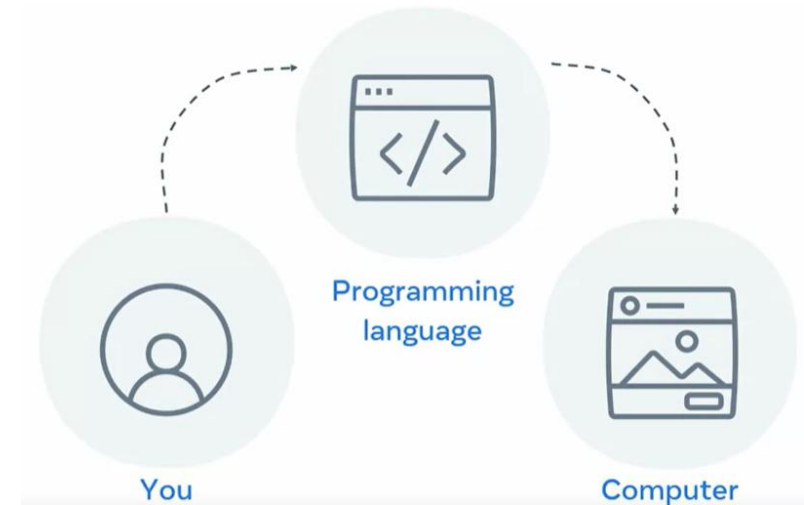
AI & DS using Python



Dr N V Ganapathi Raju

# Programming

- Programming means tells computer what to do.
- A computer program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.
- Rules of Programming languages:
  - Syntax: grammar  
The rules for how each instructions is written .
  - Semantics: logic  
The effect the instructions have (meaning)



# Programming: a way of thinking

---

- The single most important skill for a Software Programmer / Data Scientist / AI Engineer is problem solving.
- Problem solving means
  - ability to formulate problems,
  - think creatively about solutions,
  - express a solution clearly and accurately
- Do practice problems
- Observe how others problem-solve

# Debugging

---

■ Programming errors are called **bugs** and the process of tracking them down and correcting them is called **debugging**.

■ Two kinds of errors can occur in a program:

1. Syntax errors
2. Runtime errors

# Places of Programming puzzles

---

- Hacker rank <https://www.hackerrank.com/domains/python>
- Codechef <https://www.codechef.com/practice/python>

# Classification of Computer Languages

---

- Machine Level Languages
- Assembly Languages
- High Level Languages

- A high-level programming language features a syntax that is **easy for humans to read and understand.**

Examples of high-level languages include Python, C++, C#, and Java

- Low-level languages are those that **can be easily understood by a machine.**
- When you write code in a high-level language, it gets converted into a low-level language, or machine code, that your computer can recognize and run.

# High-Level to Low-Level Languages

- In a high-level language like Python, the addition of two numbers can be expressed more naturally:

`c = a + b`

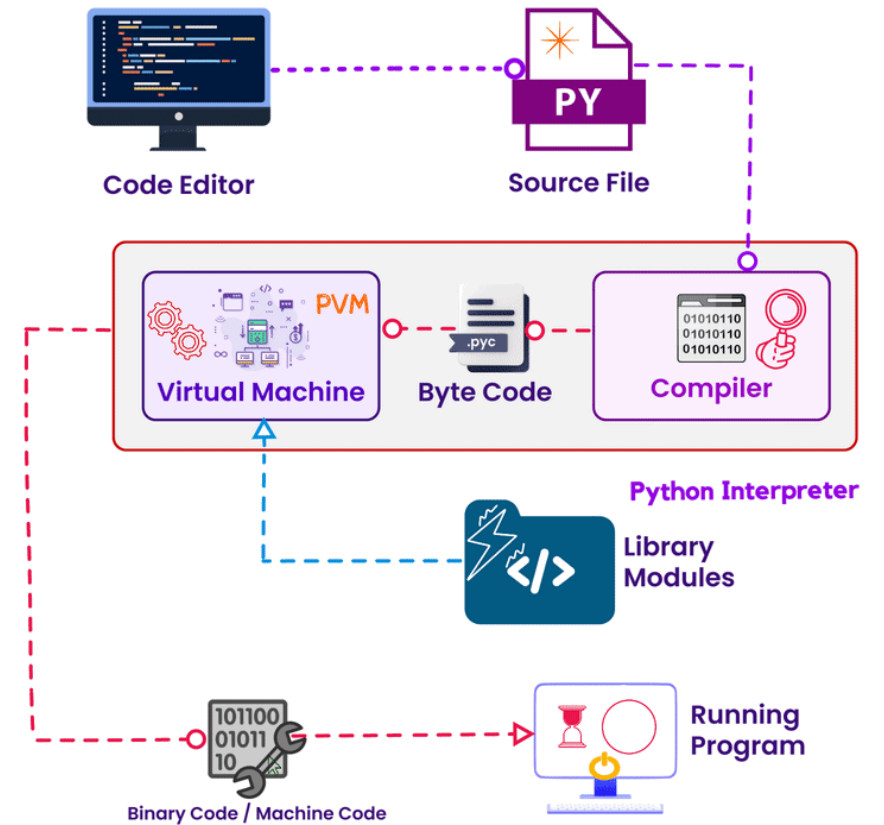
Much Easier!

- But, we need a way to translate the high-level language into a machine language that a computer can execute
  - To this end, high-level language can either be *compiled* or *interpreted*

# How Python Interpreter works ?

- **Writing the Code:** Type Python program in a text editor, saving it as a **'py'** file
- **Python Interpreter:** When run the program, it's sent into the Python Interpreter, which is made up of two parts:
  - ❖ **Compiler:** This byte code is saved in a **'pyc'** file, helping our program run faster the next time.
  - ❖ **Python Virtual Machine (PVM):** The PVM uses the byte code and follows the instructions one by one until the program is finished or runs into an error.
- **Library Modules:** If program uses library modules from Python's standard library or elsewhere, these are also changed into byte code.
- **From Byte Code to Machine Code:** The PVM converts the byte code into machine code, which is a series of 1s and 0s.
- **Running the Program:** computer uses it to run your program.

## How Python Works



# Scripting

---

- Scripts are **for short development cycles**
- A program that is
  - Short
  - Simple
  - Can be written very quickly

# Scripting vs Programming Languages

| Scripting languages             | Programming languages   |
|---------------------------------|-------------------------|
| Platform-specific               | Platform cross-platform |
| (Mostly) interpreted            | Compiled                |
| Slower at runtime               | Faster at runtime       |
| Less code-intensive             | More code-intensive     |
| Creates apps as part of a stack | Creates standalone apps |

# Why the name Python?

- When he began implementing Python, **Guido van Rossum** was also reading the published scripts from “**Monty Python’s Flying Circus**”, a BBC comedy series from the 1970s.
- **Van Rossum** thought he needed a name that was **short, unique, and slightly mysterious**, so he decided to call the language **Python**.



# From Inventors point of view about Python

- Python is an experiment in how much **freedom** programmers need.
- Too much **freedom** and **nobody** can read another's code; too little and expressiveness is **endangered**
- The joy of coding Python should be in seeing **short, concise, readable** classes that express a lot of action in a small amount of clear code -- not in reams of trivial code that bores the reader to death.



Guido van Rossum

# Features of Python

---

- Simple and Easy to learn
- General purpose, High-level Prog. Lang.
- Portable
- Object-Oriented
- Interpreted
- Large Standard Libraries
- GUI programming
- Supports Databases
- Dynamically Typed
- Open Source
- Productivity Increased and Reduced development ti

# Extended Features of Python

---

- **Garbage collected:** python automatically manages memory in a program by dumping objects that are no longer needed
- **Scalable:** Judging from growing list of global companies using python
- Performing complex math operations
- Server-side Web application development
- Rapid Application development
- Maintenance costs are reduced
- Automation

# DEMAND OF PYTHON- CUTTING

## EDGES OF OTHER TECHNOLOGIES

### DATA SCIENCE

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the three virtues, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32.

### JOBS & GROWTH

is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only



### LIBRARIES AND FRAMEWORKS

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### WEB DEVELOPMENT

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### HUGE COMMUNITY

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### SALARY

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source.

### MULTI PURPOSE

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### AUTOMATION

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### MACHINE LEARNING

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

### SIMPLICITY

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of Classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undetectable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet", comes from a line in section 1.10.32."

# Introduction to Python

---

- Uses whitespaces to delimit blocks
- Variables do not declaration
- Uses indentation
- Case sensitive
- Available in 2.x and 3.x versions
- All python programs must have an extension `.py` but jupyter notebooks will have `.ipynb` extension

# Python 3: Fibonacci series up to n

```
>>> def fib(n):  
>>>     a, b = 0, 1  
>>>     while a < n:  
>>>         print(a, end=' ')  
>>>         a, b = b, a+b  
>>>     print()  
>>> fib(1000)
```

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

# print()

- The `print()` function prints the given object to the standard output device (screen) or to the text stream file.
- It doesn't return any value; returns `None`

```
In [8]: print ('Hello World')
```

```
Hello World
```

```
In [9]: print ("Hello World")
```

```
Hello World
```

```
In [10]: str = "Hello, World!"  
print(str)
```

```
Hello, World!
```

```
In [11]: print ('Hello World \n')
```

```
Hello World
```

# input()

- The `input()` function takes input from the user and returns it
- The `input()` function takes a single optional argument:
  - `prompt` (Optional) - a string that is written to standard output (usually screen) without trailing newline
- The `input()` function reads a line from the input (usually from the user), converts the line into a string by removing the trailing newline, and returns it.
  - If EOF is read, it raises an `EOFError` exception.

```
name = input('Enter your name: ')\nprint('Hello, ', name)
```

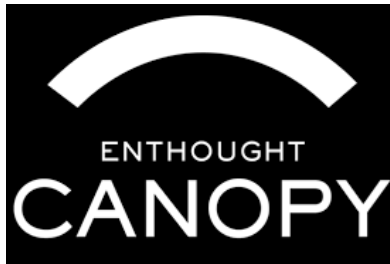
Enter your name:

```
name = input('Enter your name: ')\nprint('Hello, ', name)
```

Enter your name: Rama  
Hello, Rama

# Distributions of Python

---



# Jupyter Notebook

---

- Jupyter Notebooks for
  - writing, running and debugging Python code and Documenting
    - Notebooks are part of an open-source project called Project Jupyter. free to use
- Jupyter Lab is a web-based interface that allows you to use Jupyter Notebooks to write, run and debug Python code.
- Jupyter Lab provides an online environment that allows you to run your code in the cloud.
- It's a great tool for creating and understanding the code you're writing because you can see your input and output all in one spot.
- <https://www.anaconda.com/download>

# Colab

---

- Colab is a web-based platform that allows you to write and run Python code really quickly in Google Drive.
- It is free and ready to use with zero configuration required.
- [https://colab.research.google.com/notebooks/basic\\_features\\_overview.ipynb](https://colab.research.google.com/notebooks/basic_features_overview.ipynb)

# Identifiers

---

- **name used to identify an object such as a variable, module, class, or function**

## **Rules for naming Identifiers are:**

1. First character can be uppercase (A to Z) / lower case (a to z) /, underscore ( \_ ) but **not digit**
2. Long sequence of characters and numbers
3. No special character
4. Keyword should not be used
5. Case sensitive. Using case is significant

# Examples on Identifiers

---

- String `isidentifier()` – to check if the identifier name is valid or not
  - returns boolean
  - Identify **valid/invalid** identifiers
    1. `ab100`
    2. `_` (underscore)
    3. `_abc`
    4. `100`
    5. `a+b`
    6. `for`

# Keywords

- Python keywords are special reserved words that have **specific meanings and purposes** and **can't be used for anything but those specific purposes**.
- These keywords are always available—you'll never have to import them into your code.

|        |          |         |          |        |
|--------|----------|---------|----------|--------|
| False  | await    | else    | import   | pass   |
| None   | break    | except  | in       | raise  |
| True   | class    | finally | is       | return |
| and    | continue | for     | lambda   | try    |
| as     | def      | from    | nonlocal | while  |
| assert | del      | global  | not      | with   |
| async  | elif     | if      | or       | yield  |

# Variables

---

- **Variables are reserved memory locations to store values.**
  - when you create a variable, you reserve some space in memory
- **Python variables do not need explicit declaration to reserve memory space.**
- **The declaration happens automatically when you assign a value to a variable.**
- **The equal sign (=) is used to assign values to variables.**

# Variable assignment

---

```
var1 = 100      # An integer assignment
```

```
print(var1)
```

```
type(var1)
```

```
var2 = 1000.0   # A floating point
```

```
print(var2)
```

```
type(var2)
```

```
var3 = "Python" # A string
```

```
print(var3)
```

```
type(var3)
```

Note:

`type()` returns **type of argument passed as parameter**

# Variable assignment

---

```
x = y = z = 10
```

```
print (x)
```

```
print (y)
```

```
print (z)
```

Note: Python allows you to assign a single value to several variables simultaneously.

# Multiple variable assignment

---

```
x,y,z = 10,25.5,"Python"
```

```
print (x)
```

```
print (y)
```

```
print (z)
```

Note:

Python also allows to assign multiple values to several multiple variables simultaneously

# Variables - Caution

---

```
#n=0  
  
print(n)
```

-----  
**NameError**

Traceback (most recent call last)

Cell In[1], line 3

1 #n=0

----> 3 print(n)

**NameError:** name 'n' is not defined

# Operators

---

- An operator is used to manipulate the values of operands

Python supports seven types of operators:

- Arithmetic operators
- Relational or Comparison operators.
- Assignment operators
- Logical operators
- Bitwise operators
- Membership operators
- Identity operators

# Arithmetic Operators

---

Used to perform basic math operations like

**+     Addition**

**-     Subtraction**

**\*     Multiplication**

**/     Division (quotient)**

**%     Modulus (remainder)**

**\*\*    Exponent (power)**

**//    Floor Division**

# Arithmetic Operators evaluation

```
a,b = 3,4
```

```
print(a + 1, a - 1, a * 3)
```

```
4 2 9
```

```
print(b / 2, b // 2, b % 2, b ** 2)
```

```
2.0 2 0 16
```

```
print(17/3, 17//3, 17 %3)
```

```
5.666666666666667 5 2
```

# Precedence of Arithmetic Operators

Order of precedence : **P E M D A S**



|    | Associativity  |
|----|----------------|
| ** | Exponential    |
| *  | Multiplication |
| /  | Division       |
| // | Floor Div      |
| %  | Modulus        |
| +  | Add            |
| -  | Sub            |

# MCQ on Arithmetic Operators

1 Which is the correct operator for power( $x^y$ )?

- a)  $X^y$
- b)  $X^{**}y$**
- c)  $X^{^^}y$
- d) None

2 Which one of these is floor division?

- a) /
- b) //**
- c) %
- d) None

3 Mathematical operations can be performed on a string.

- a) True
- b) False**

4 Operators with the same precedence are evaluated in which manner?

- a) Left to Right**
- b) Right to Left
- c) Can't say
- d) None of the mentioned

5 What is the output of this expression,  $3*1**3$ ?

- a) 27
- b) 9
- c) 3**
- d) 1

# Problem Solving

```
[ ]: x,y=10,20
      print("input", x, y)

      temp=x
      x=y
      y=temp

      print("\nOutput", x, y)
```

```
[ ]: x,y=10,20
      print("input", x, y)

      x=x+y
      y=x-y
      x=x-y

      print("\nOutput", x, y)
```

```
x,y=10,20

print("input",x,y)

x=x*y
y=x/y
x=x/y

print("\nOutput", x, y)
```

```
x,y=10,20
print("input",x,y)

x,y=y,x

print("\nOutput", x, y)
```

# MCQ on Precedence of operators

1 What is the value of the following expression?

$22//3+3/3$

- a) 8    **b) 8.0**    c) 8.3    d) 8.33

2 What are the values of the following Python expressions?

$2**(3**2)$

$(2**3)**2$

$2**3**2$

- a) 64, 512, 64                      b) 64, 64, 64  
c) 512, 512, 512                    **d) 512, 64, 512**

3 What will be the value of x in the following Python expression?

$x = \text{int}(43.55+2/2)$

- a) 43    **b) 44**    c) 22    d) 23

4 What will be the output of the following Python expression?

$24//6\%3, 24//4//2$

- a) (1,3)**    b) (0,3)    c) (1,0)    d) (3,1)

# Expression Evaluation

The eval() function :

Expression: the string that has been processed and evaluated as a Python expression.

```
] : x = 10  
    y = 20  
    print(eval("x + y"))  
  
30
```

```
] : print(eval("x + 2", {"x": 2}))  
  
4
```

```
] : text = "4 * 6 - 3"  
  
    print("Original string is : " + text)  
  
    result = eval(text)  
    print(result)  
  
Original string is : 4 * 6 - 3  
21
```

```
] : evaluate = 'x * (x+1) * (x+2)'  
    x = 3
```

```
: eval("2 ** 8")
```

```
: x = 100  
  eval("x * 2")
```

# Relational / Comparison operators

- Used to compare the values on either sides of them and decide the relation among them

- return True / False

|          |                          |
|----------|--------------------------|
| ==       | double equal to          |
| !=<br><> | not equal                |
| >        | greater than             |
| <        | less than                |
| >=       | greater than or equal to |
| <=       | less than or equal to    |

# Example on Relational Op

**$1 < 2$**

**$2.0 \geq 1$**

**$2.0 == 2.0$**

**$2.0 != 2.0$**

**$1 == (2 < 3)$**

**$(1 == 2) < 3$**

In [2]: 1 print(1 < 2)

True

In [3]: 1 print(2.0 >= 1)

True

In [4]: 1 print(2.0 == 2.0)

True

In [5]: 1 print(2.0 != 2.0)

False

In [6]: 1 print(1 == (2 < 3))

True

In [7]: 1 print((1 == 2) < 3)

True

# MCQ on Relational Operators

---

1 What is the output of `print 0.1 + 0.2 == 0.3`?

- a) True
- b) False**
- c) Machine dependent
- d) Error

2 What is the output of `print 1 == 2 < 3`?

- a) True
- b) False**
- c) Machine dependent
- d) Error

# MCQ on Relational Operators

```
: print(0.1 + 0.2)
```

0.30000000000000004

```
: print(0.3)
```

0.3

```
: print(0.1 + 0.2 == 0.3)
```

False

```
1 == 2 < 3
```

False

```
1 == 2
```

False

```
2 < 3
```

True

```
(1 == 2 != 3)
```

False

```
1 == 2
```

False

```
2 != 3
```

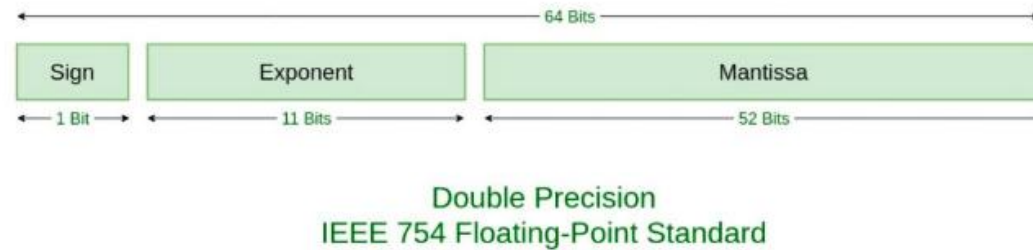
True

```
print 0.1 + 0.2 == 0.3
```

The two types of numbers in python are : *integer* and *float*.

Integer data types store **whole** numbers, while float data types store **fractional** numbers.

Very small and very large numbers are usually stored in scientific notation.



```
1 0.1 + 0.2 == 0.3
```

False

```
1 print(0.1+0.2)
```

0.30000000000000004

```
1 print(0.3)
```

0.3

The representation of the number 0.1 is :

**0 1.10011001100110011001100110011001100110011010 01111111011**

0.2 would be represented as

**0 1.10011001100110011001100110011001100110011010 01111111100**

Adding the two after making the exponents same for both would give us:

**0.1 + 0.2 = 0.30000000000000004.**

Convert 0.1 to binary

$$0.1 \times 2 = 0.2 \quad \text{Keep 0, move 0.2 to next step}$$

$$0.2 \times 2 = 0.4$$

$$0.4 \times 2 = 0.8$$

$$0.8 \times 2 = 1.6$$

$$0.6 \times 2 = 1.2$$

~~1.2~~

$$1.2 \times 2 = 2.4$$

$$0.4 \times 2 = 0.8$$

$$0.8 \times 2 = 1.6$$

$$0.6 \times 2 = 1.2$$

$$0.2 \times 2 = 0.4$$

bin value of 0.1 is 0.000110011...

So  $0.1 + 0.2$  is not exactly 0.3

# MCQ on Relational Operators

---

```
x = 0.1 + 0.1 + 0.1
```

```
y = 0.3
```

```
print(round(x, 3) == round(y, 3))
```

True

**One way to work around this problem is to round both sides of the equality expression to a number of significant digits.**

```
from math import isclose
```

```
x = 0.1 + 0.1 + 0.1 y = 0.3
```

```
print(isclose(x,y))
```

True

**It provides the isclose() function from the math module returns True if two numbers are relatively close to each other.**

# Comparison operator logic

- Comparisons can be chained arbitrarily,
- e.g.,  $x < y \leq z$  is equivalent to  $x < y$  and  $y \leq z$ , except that  $y$  is evaluated only once (but in both cases  $z$  is not evaluated at all when  $x < y$  is found to be false).
- **Why does  $(1 == 2 != 3)$  evaluate to False in Python, while both  $((1 == 2) != 3)$  and  $(1 == (2 != 3))$  evaluate to True?**

```
>>> 1==2
```

```
=> False
```

```
>>> 2!=3
```

```
=> True
```

```
>>> (1==2) and (2!=3)
```

```
# False and True
```

```
=> False
```

# Assignment operators

---

- **Used to assign values to variables.**

|     |                           |
|-----|---------------------------|
| =   | equal to (assignment)     |
| +=  | Addition assignment       |
| -=  | Subtraction assignment    |
| *=  | Multiplication assignment |
| /=  | Division assignment       |
| %=  | Modulus assignment        |
| **= | Exponentiation assignment |
| //= | Floor division assignment |

# Assignment operators

---

## Note :

- In a regular assignment statement, the right-hand side is evaluated first,  
before this assigning it to the left-hand side.

`a = 2 + 3`

- the left side is evaluated first, before evaluating the right side.

`a += 3`

# Assignment op examples

**a,b,c = 21,10,0**

**c = a + b**

**c**

**c += a**  
**print(c)**

**c \*= a**  
**print(c)**

**c /= a**  
**print(c)**

```
In [10]: 1 a,b,c = 21,10,0
```

```
In [11]: 1 c = a + b  
        2 c
```

```
Out[11]: 31
```

```
In [12]: 1 c += a  
        2 print(c)
```

```
52
```

```
In [13]: 1 c *= a  
        2 print(c)
```

```
1092
```

```
In [14]: 1 c /= a  
        2 print(c)
```

```
52.0
```

# Logical Operators

---

- **Used to join two or more conditions/expressions**

---

|     |  |
|-----|--|
| and | Returns True if both the operands are True |
|-----|--|

---

|    |   |
|----|---|
| or | Returns True if either of the operands are True |
|----|---|

---

|     |   |
|-----|---|
| not | Returns True if operand is False and<br>False if the operand is True. |
|-----|---|

---

# Logical Operators

---

**a=50**

**print((a % 4 and a > 0))**

**a=50**

**print(not(a % 4 or a > 0))**

**b1 = True**

**b2 = False**

**print(b1 and b2)**

**print(b1 or b2)**

**print(not b1)**

## MCQ on Logical Operators

---

1 What is the output of print **not (5 < 3 and 5 < 33)** ?

- a) True
- b) False
- c) Machine dependent
- d) Error

2 What is the output of print **20 + 3 >= 23 or 5 != 5** ?

- a) True
- b) False
- c) Machine dependent
- d) Error

# Bitwise Operators

---

- **Act on operands in binary digit form, bit by bit**

|    |                     |
|----|---------------------|
| &  | Bitwise AND         |
|    | Bitwise OR          |
| ~  | Bitwise NOT         |
| ^  | Bitwise XOR         |
| >> | Bitwise right shift |
| << | Bitwise left shift  |

# Bitwise Operators

- **AND** : If both the inputs '1' , then the output is '1' else '0'
- **OR** : If either input is '1' then output is '1' else '0'
- **XOR** : if both the inputs are the same, then the output is '0', else the output is '1'
- **Bitwise NOT**: inverses every bit then make all 1s to 0s and vice versa  $\text{NOT } x = -x - 1$
- **Bitwise left shift ( << )**:  
shifts **all bits left** by the specified position. **Note:  $a \ll b = a * 2^{**}b$**
- **Bitwise Right shift ( >> )**  
shifts **all the bits to right** by specified position. **Note:  $a \gg b = a / 2^{**}b$**

# Examples on bitwise op

---

**a, b = 4, 5**

**print (a & b)**

**a, b = 4, 5**

**print (a | b)**

**a, b = 4, 5**

**print (a ^ b)**

**a = 4**

**print (~a)**

# Examples on bitwise op

---

**a = 7**

**print (a >> 1)**

**a = 7**

**print (a << 1)**

# MCQ on Bitwise operators

1 What will be the value of x in the following expression?

$$x \gg 2 = 2$$

- a) 8      b) 4      c) 2      d) 1

2 What will be the output of the following expression?

$$4 \wedge 12$$

- a) 2      b) 4      c) 8      d) 12

3 What will be the output of the following expression?

$$\sim 100?$$

- a) 101      **b) -101**      c) 100      d) -100

4 What will be the output of the following code if a=10 and b =20?

a,b=10,20

a=a^b

b=a^b

a=a^b

print(a,b)

- a) 10 20      b) 10 10      **c) 20 10**      d) 20 20

## Swap using XOR operation

---

```
]: a,b=10,20  
print(a,b)
```

```
a = a ^ b  
b = a ^ b  
a = a ^ b
```

```
print()  
print(a,b)
```

# Membership Operators ( in , not in )

---

- Used to check if a specific item is present in a sequence (such as a string, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).
  - returns True / False

**s = "Python Programming"**

**" Python " in s**

# Identity Operators

- Used to check if two variables are located on the same memory allocation.

- returns True / False



is

Returns True if the operands are identical.

is not

Returns True if the operands are not identical.

*id()*

*b = 200*

*a = 100*  
*int value*

*Variable*

*Memory loc*

*unprocessed  
15*

# Examples on Identity Op

---

```
x1 = 10
```

```
y1 = 10
```

```
print (x1 is y1)
```

```
x3 = [1, 2, 3]
```

```
y3 = [1, 2, 3]
```

```
print (x3 is y3)
```

# Precedence of Operators

| Precedence         | Operator                         | Description                                      |
|--------------------|----------------------------------|--|
| lowest precedence  | or                               | Boolean OR                                       |
|                    | and                              | Boolean AND                                      |
|                    | not                              | Boolean NOT                                      |
|                    | ==, !=, <, <=, >, >=, is, is not | comparisons, identity                            |
|                    |                                  | bitwise OR                                       |
|                    | ^                                | bitwise XOR                                      |
|                    | &                                | bitwise AND                                      |
|                    | <<, >>                           | bit shifts                                       |
|                    | +, -                             | addition, subtraction                            |
|                    | *, /, //, %                      | multiplication, division, floor division, modulo |
|                    | +X, -X, ~X                       | unary positive, unary negation, bitwise negation |
| highest precedence | **                               | exp  |

# Basic Data Types

---

- Data types are the classification, represent the kind of value that tells what operations can be performed on a particular data
- Everything in Python is an object.
  - data types are considered classes
  - variables are the instances / objects of these classes
- `type()` determine which class a value or variable belongs to
- `isinstance()` is used to check whether an object belongs to a specific class.

## Built-in Python data types are:

---

- Numeric data types: int, float, complex
- String data types: str
- Sequence types: list, tuple, range
- Binary types: bytes, bytearray
- Mapping data type: dict
- Boolean type: bool
- Set data types: set, frozenset

# Numbers

- **A Number as a data type stores numeric values and is immutable,**
  - when the value of the variable is changed, a new object is allocated.
- Types of numbers in Python **are int, float, and complex** respectively
- Creating number objects
  - `x = 1`  
`print(type(x))`
  - `y = 1.0`  
`print(type(y))`
  - `z=1+2j`                      \* where j is an imaginary part  
`print(isinstance(1+2j,complex))`
- Deleting number objects
  - `del x, y, z`

# Type conversion

---

- converting the value of one data type (integer, string, float, etc.) to another data type
- Python has two types of type conversion.
  - Implicit Type Conversion
    - automatically converts one data type to another data type
    - doesn't need any user involvement
  - Explicit Type Conversion
    - users convert the data type of an object to required data type
    - Using predefined functions using `int()`, `float()`, `str()` etc..

# implicit conversion example

---

```
int_n = 123
```

```
flo_n = 1.23
```

```
new_res = int_n + flo_n
```

```
print(type(int_n))
```

```
print(type(flo_n))
```

```
print(new_res)
```

```
print(type(new_res))
```

**Note :**

**Python always converts smaller data type to larger data type to avoid the loss of data**

# Explicit conversion example

---

```
int_n = 123
str_n = "456"
print(type(int_n))
print(type(str_n))
num_str = int(str_n)
print(type(num_str))
num_sum = int_n + num_str
print(num_sum)
print(type(num_sum))
```

## Note :

Explicit type conversion is known as **type casting**

# Booleans

---

- used to represent truth values of two constant objects **False** and **True**.
- In numeric contexts, behave like the integers 0 **and** 1, respectively.
- **function bool()** can be used to cast any **value** to a Boolean
- written **as False and True**, respectively.

# Booleans (rules)

---

- Almost any value is evaluated to **True** except **0**
  - Any string is **True**, except empty strings.
  - Any number is **True**, except **0**.
  - Any list, tuple, set, and dictionary are **True**, except empty ones.
- Many values that evaluates to **False**,  
empty values, such as **()**, **[]**, **{}**, **""**, the number **0**, and the value **None**.

# Boolean examples

---

True + False

False-True

True\*\*False

False<=True

~True

True<<2

False is 0

bool('')

# String

---

**String : as a contiguous set of characters represented in the quotation marks.**

- are immutable.
- enclosing in single as well as double quotes.

```
str = "Python Programming"  
str
```

# List

---

- are the data structure that can hold different type of data.
- composed by storing a sequence of different type of values
- enclosed between square([]) brackets, mutable

```
list1=[100,'python',50.75,'a',True]
```

```
print (list1)
```

# Tuple

---

**Tuple : is a sequence of immutable objects; tuple cannot be changed.,**

- enclosed within parenthesis

```
tup1=(10,'python',56.8,'a')  
print(tup1)
```

# Dictionary

---

- It is a container that contains data, enclosed within curly braces, set of key and value pair
  - an unordered.
  - pair i.e., key and value is known as item.
  - key passed in the item must be unique.

```
dict1={'No':100 , 'Name':'Ram' , 'Salary':125000.00}  
print (dict1['No'])  
print (dict1['Name'])  
print (dict1['Salary'])
```

# Sets

---

- Sets are a **mutable** collection of **unique** values
- Values are **unordered**
- Does **not support indexing**
- Highly useful to efficiently **remove duplicate values from a list or tuple**
- Perform common math operations like **unions and intersections**

# Bytes

---

- The bytes represents a **group of byte numbers**.
- Bytes are immutable (Cannot be changed).
- **Use bytes data type if we want to handle binary data like images, videos, and audio files.**
- In bytes, allowed values are **0 to 256**.
- If we are trying to use any other values, then we will get a ValueError.

# bytearray

---

- The bytearray data type same as the bytes type except bytearray mutable.
- The bytearray() constructor returns a bytearray object.

# None

---

- The None keyword is used to define a null variable or an object.
- In Python, None keyword is an object, and it is a data type of the class NoneType.
- We can assign None to any variable, but you can not create other NoneType objects.
  - None is not the same as False.
  - None is not 0.
  - None is not an empty string.
  - Comparing None to anything will always return False except None itself.

# PIP

---

- pip is the standard package manager for Python.
- We can use pip to install additional packages that are not available in the Python standard library.

`pip install numpy`