# **Python Introduction :**

What is Python?

* Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

### **Why Python?**

* Python works on different platforms (Windows, Mac, Linux , etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* IDLE means (“Integrated Development and Learning Environment”).
* It **allows programmers to easily write Python code**. Just like Python Shell, IDLE can be used to execute a single statement and create, modify, and execute Python programms. IDLE provides a fully-featured text editor to create Python programms .
* That include features like syntax highlighting, autocompletion, and smart indent.
* IDLE has two main window types, the Shell window and the Editor window.

**Get Started** with python**:**

## Python Indentation

* Indentation refers to the spaces at the beginning of a code line.
* Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
* Python uses indentation to indicate a block of code.

### **Example:**

if 10 > 5:  
   print("Ten is greater than five!")

* Python will give you an error if you skip the indentation:

### **Example**

Syntax Error:

if 10 > 5:  
 print("Ten is greater than five!")

# **Python Comments:**

* Comments can be used to explain Python code.
* Comments can be used to make the code more readable.
* Comments starts with a #, and Python will ignore them

### **Example**

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

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

## Multi Line Comments:

## Multi line comments starts with “”” \_\_\_\_\_\_\_”””.

### **Example:**

"""  
 This is a comment  
 written in  
 more than just one line  
 """  
 print("Hello, World!")

# **Python Variables:**

# **Variables**

* Variables are containers for storing data values.

**Creating Variables:**

* Python has no command for declaring a variable.

**Example**

x = 10  
 y = "mustafa"  
 print(x)  
 print(y)

* roles in creating variables

### **Example**

* valid variable names:

myvar = "college"  
 my\_var = "college"  
 \_my\_var = "college"  
 myVar = "college"  
 MYVAR = "college

myvar2 = "college"

**Example**

* Invalid variable names, and gives error:

2myvar = "college"  
 my-var = "college"  
 my var = "college"

**Main Topics In Python: -**

* **Operators.**
* **Data Types.**

# Python If ... Else.

## Python Loops.

## While loop.

## For loop.

# Python Functions.

# Python Classes and Objects

## Python Operators:

* Operators are used to perform operations on variables and values.
* In the example below, we use the + operator to add together two values:
* Python divides the operators in the following groups:
* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

## Python Arithmetic Operators:

* Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |  |

## Python Assignment Operators:

* Assignment operators are used to assign values to variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** |  |
| = | 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 |  |
| &= | 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 |  |

## Python Comparison Operators:

* Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Python Logical Operators:

* Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **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 and x < 10) |

## Python Identity Operators:

* Identity operators are used to compare the objects is there are not and return true if it is there.

|  |  |  |
| --- | --- | --- |
| **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 |

## Python Membership Operators:

* Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **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 |

## Python Bitwise Operators:

* Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

# **Python Data Types:**

* Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.
* Example: a=5,a=10.5,a=”student”,…etc



### **Numbers:**

* Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type.

**1**.**Int -** Integer values like-10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer.

Example:

a = 10

print("Type of a: ", type(a)) 🡪Type of a: <class 'int'>

**2.Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

Example:

a = 5.0

print("Type of a: ", type(a)) 🡪Type of a: <class 'float'>

**3.Complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary

Part respectively**,** ,And the complex number like 2+3j ,… etc

Example:

a = 2+3j

print("Type of a: ", type(a)) 🡪Type of a: <class 'complex'>

### **Sequence Type:**

* In Python, sequence is the ordered collection of similar or different data types

1. **String**-The string can be defined as the sequence of characters represented in the quotation marks(“). And

we have single, double, or triple quotes

**Example - 1**

str = "string using double quotes"

**print**(str)

s = '''''A multiline

string'''

**print**(s)

**Output :** s

**String using double quotes**

**A multiline**

**string**

### **2.List :**

* + - * Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are,
      * separated with a comma (,) and enclosed within square brackets [].

**Example:**

list1  = [1, "hello", "student", 2]

**print**(type(list1))

list2  = [1, 3, 4, 2]

**print**(type(list2))

**Output:**

<class 'list'>

<class 'list'>

### **3.Tuple:**

* A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types.
* The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

**Example:**

tup  =  (1,"hello", "student", 2)

print(type(tup))

tup1=(1,2,3,4)

print(type(tup1))

**output :**

**<**class ‘tuple’>

<class ‘tuple’>

### **Dictionary :**

* Dictionary is an unordered set of a key-value pair of items.
* And we can excessed elements in dictionary by using keys.
* The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}

**Example :**

d = {1:'student', 2:'college', 3:'job', 4:'house'}

print(d)

**output:**

{1: 'student', 2: 'college', 3: 'job', 4: 'house'}

### **Boolean :**

* Boolean type provides two built-in values, True and False

**Example :**

**print**(type(True))

**print**(type(False))

  **output:**

<class ‘bool’> >

<class ‘bool’>

* **Set :**
* In Python, [Set](https://www.geeksforgeeks.org/python-sets/) is an unordered collection of data type.
* Set has unique elements init.
* Set no duplicate elements.

**Example :**

Set1 = {'mustafa', 6, 4,'Python'}

Set2 = {1,2,3,2,'python','python'}

print(Set1)

print(Set2)

**output :**

{4, 'Python', 'mustafa', 6}

{1, 2, 3, 'python'}

# **Python If ... Else :**

* If and elif,else statements used to check conditions python.
* **If statement** :
* An "if statement" is written by using the if keyword.

**Example :**

name=input("enter the name :")

if name == "Sathyabama":

print("True")

**output :**

enter the name: Sathyabama

True

## Elif statement :

## The elif keyword in python saying that "if the previous if conditions were not true, then try this elif condition".

## Example:

## name=input("enter the name :")

## if name == "Sathyabama":

## print("True ")

## elif name !=”Sathyabama”:

## print("False")

## output :

## enter the name : Sathyabama

## True

## Enter the name : Student

## False

## Else statement :

## The else keyword catches anything which isn't caught by the preceding conditions.

## Example:

## name=input("enter the name :")

## if name == "Sathyabama":

## print("True from Sthyabama")

## elif name =="Student"

## print("True from Student")

## else:

## print("False")

## output :

## enter the name : Student

## True from Student

## enter the name : college

## False

## Python Loops:

## While Loop :

## while loop in python can execute a set of statements as long as a condition is true.

## Example :

## i = 1 while i < 6:    print(i)    i += 1

## output:

## 1

## 2

## 3

## 4

## 5

## For Loop:

## A for loop is used for iterating over a sequence.

## With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

## Example :

## Languages= ["c++", "python", "java"]

## for i in Languages:   print(i)

## output :

## c++

## python

## java

## 

## Python Functions:

## A function is a block of code which only runs when it is called.

## You can pass data, known as parameters, into a function.

## A function can return data as a result.

## A function is created by “def” keyword 🡪def myfunction():

## A function is called by using the function name followed by parenthesis🡪myfunction()

## Example :

## def evenOdd(x):

## if (x % 2 == 0):

## print(“even”)

## else:

## print(“odd”)

## 

## #calling the function

## evenOdd(4)

## evenOdd(11)

## Output :

## even

## odd

# **Python Classes and Objects :**

* class is a virtual entity and can be seen as a blueprint of an object
* On the other hand, the object is the instance of a class.
* Almost everything in Python is an object, with its properties and methods.

**Simple Example :**

Calss Human:

def \_\_init\_\_(self,n,o):

self.name =n

self.occupation =o

def do\_work(self):

if self.occupation == “cricket player”:

print(self.name,”plays cricket”)

if self.occupation ==”Student”:

print(self.name,”used to study”)

def speaks(self):

print(self.name,”says hi and hello how are you all”)

dhoni=Human(“MS dhoni”,cricket player”)

dhoni.do\_work()

dohni.speaks()

Output:

MS dhoni plays cricket

MS dhoni says hi and hello how are you all

**AND now we are compare python and java with some examples :**

* **Triangle Number Pattern 🡪 Python**

n=int(input())

currRow=1

while currRow <=n:

currCol =1

while currCol <=currRow:

print(currRow,end=” “)

currCol +=1

print()

currRow +=1

**output:**

1

22

333

4444

* **Triangle Number Pattern 🡪 java**

import java.util.Scanner;

public class Pattern {

public static void main(String[] args){

Scanner s =new Scanner(System.int);

int n=s.nextInt();

int currRow=1;

while(currRow <=n){

int currCol =1;

while(currCol <= currRow){

System.out.print(currRow);

currCol +=1;

}

System.out.println();

currRow +=1;

}

}

}

**output:**

1

22

333

4444

* **Alphabet Pattern** 🡪**Python**

n=int(input())

currRow =1

while currRow <=n:

currCol =1

ch=ord(‘A’)+currRow -1

while currCol <=currRow:

print(chr(chr),end=” “)

currCol +=1

print()

currRow +=1

output:

A

BB

CCC

* **Alphabet Pattern** 🡪**java**

Import java.util.Scanner;

Public class Pattern {

Public static void main(String[] args){

Scanner s= new Scanner(System.in);

int n=s.nextInt();

int currRow =1;

while(currRow <=n){

int currCol =1;

char ch =(char)(‘A’+currRow-1);

while(currCol <= currRow) {

System.out.print(ch);

currCol +=1;

}

System.out.println();

currRow +=1;

}

}

}

**Output :**

A

BB

CCC

## 

## 

## 

## 