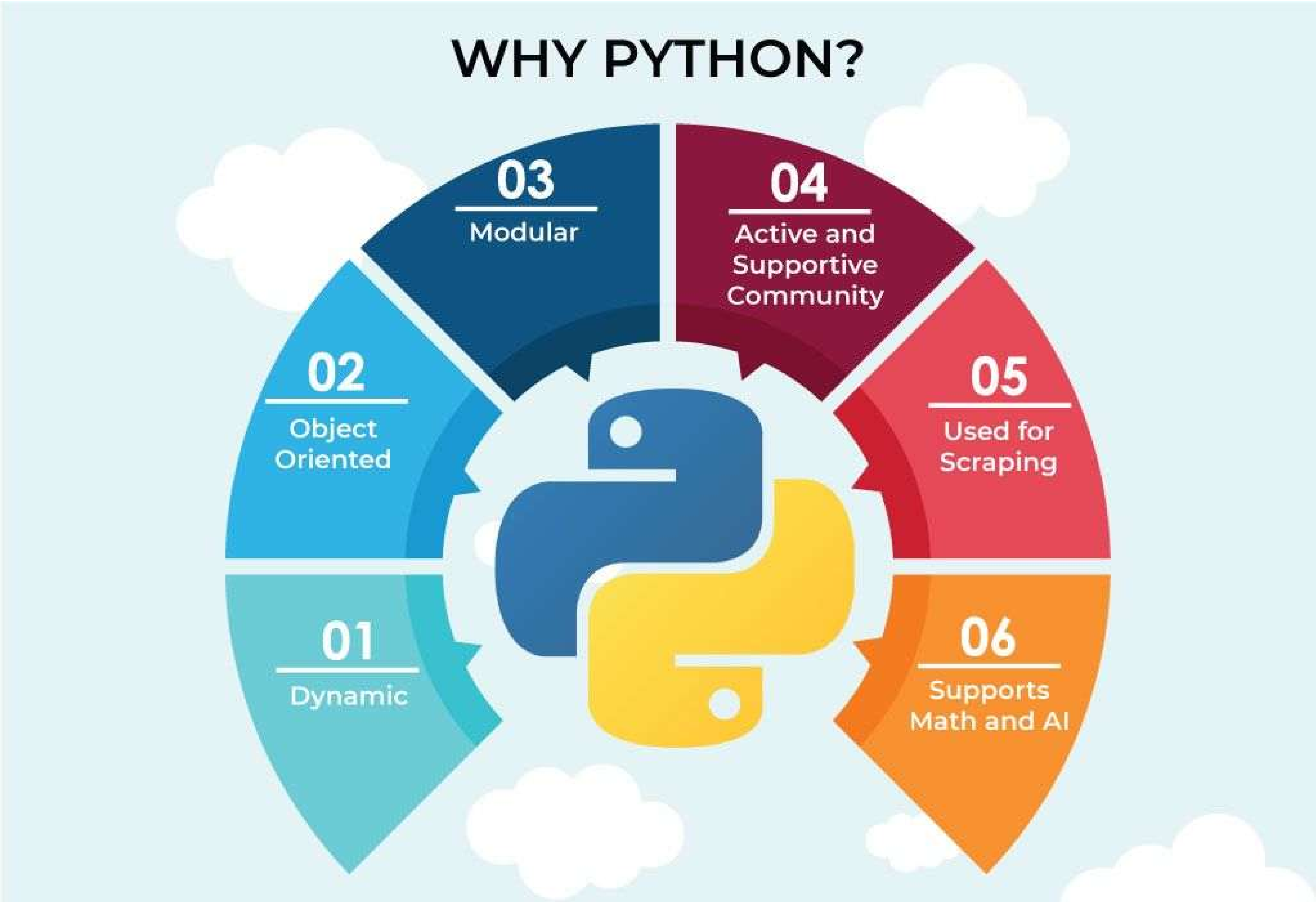
Python Basics

Sanjiva Bennur

EXCELSOFT



WHY PYTHON?

01 Dynamic

02 Object Oriented

03 Modular

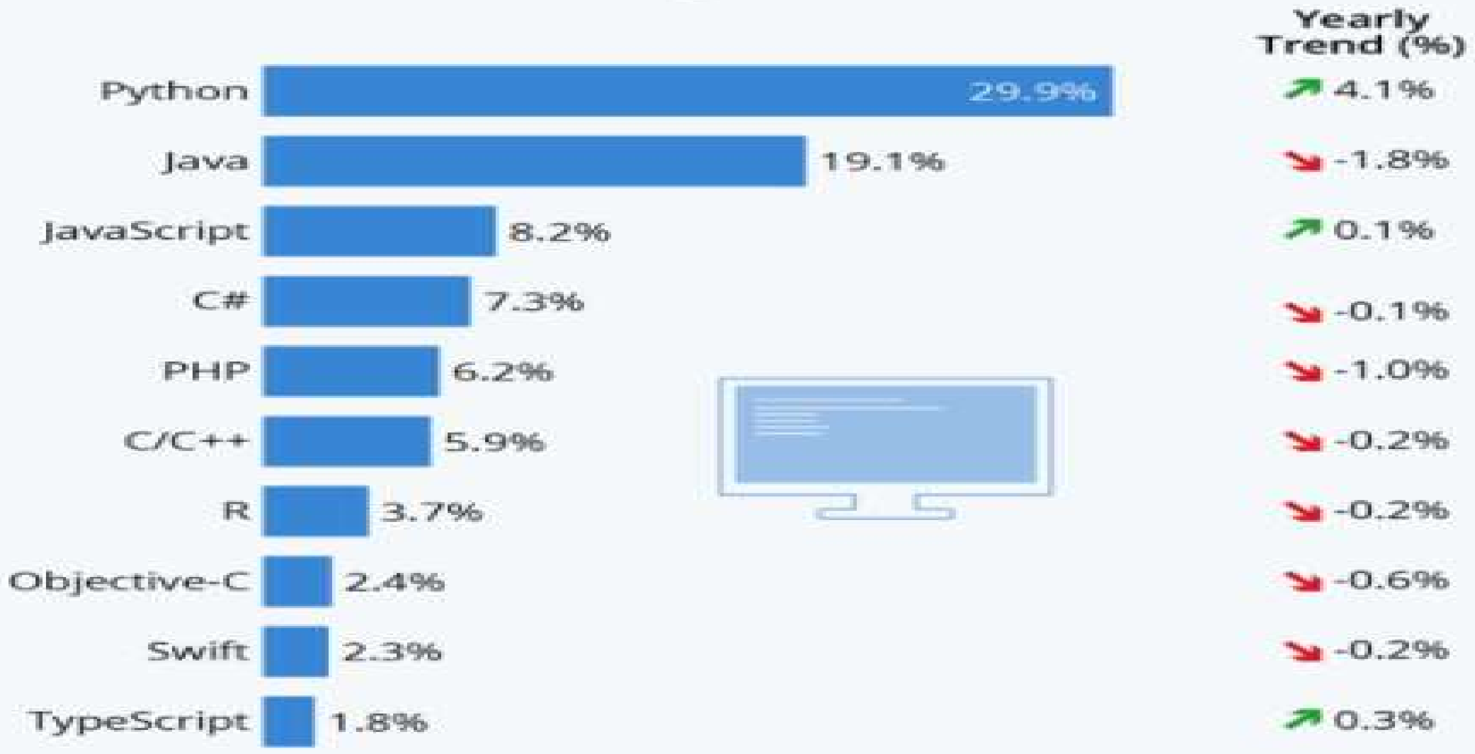
04 Active and Supportive Community

05 Used for Scraping

06 Supports Math and AI

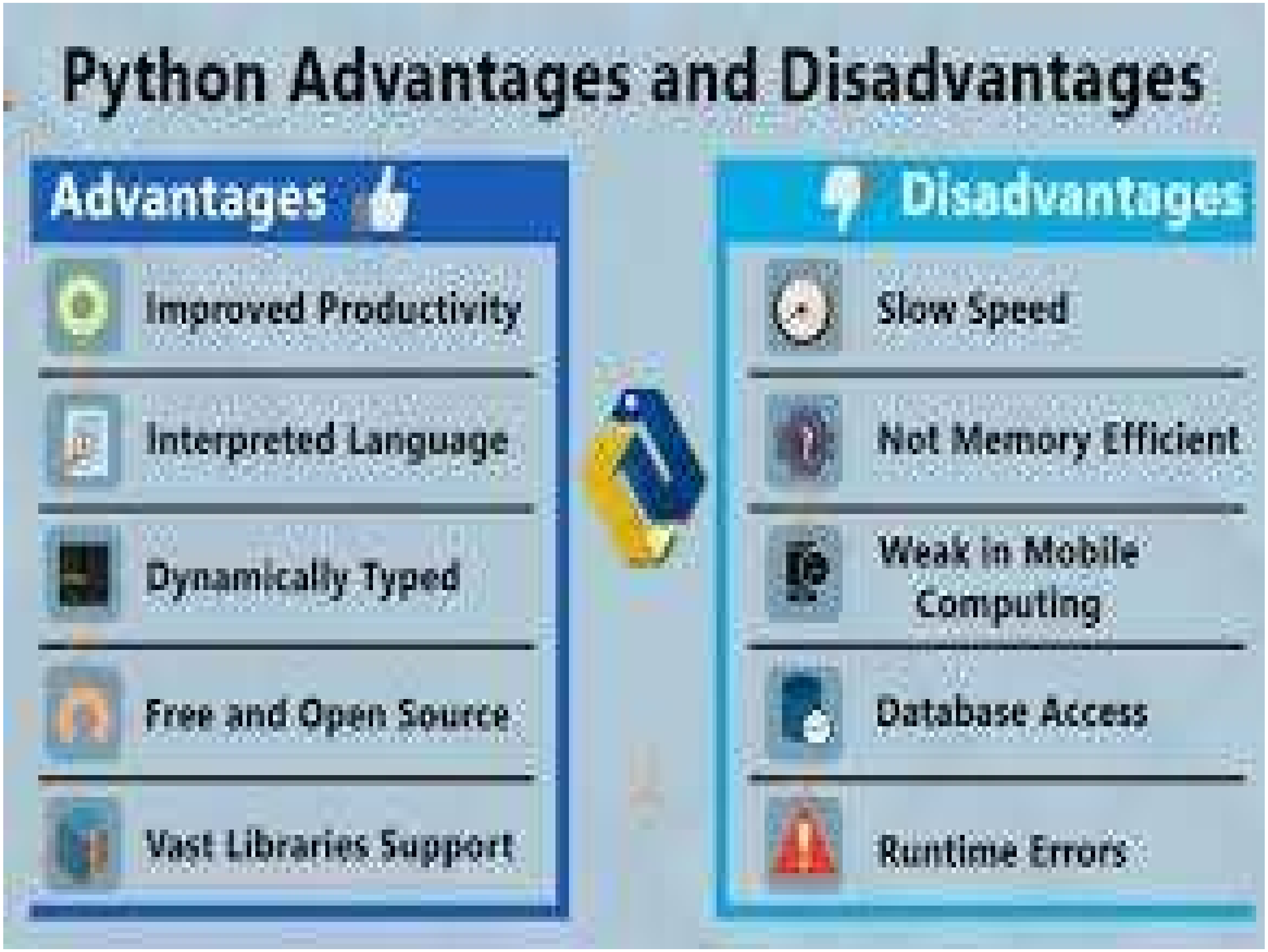
Python Remains Most Popular Programming Language

Popularity of each programming language based on share of tutorial searches in Google



Yearly trend compares percent change from Feb 2019 to Feb 2020
Sources: GitHub, Google Trends

statista



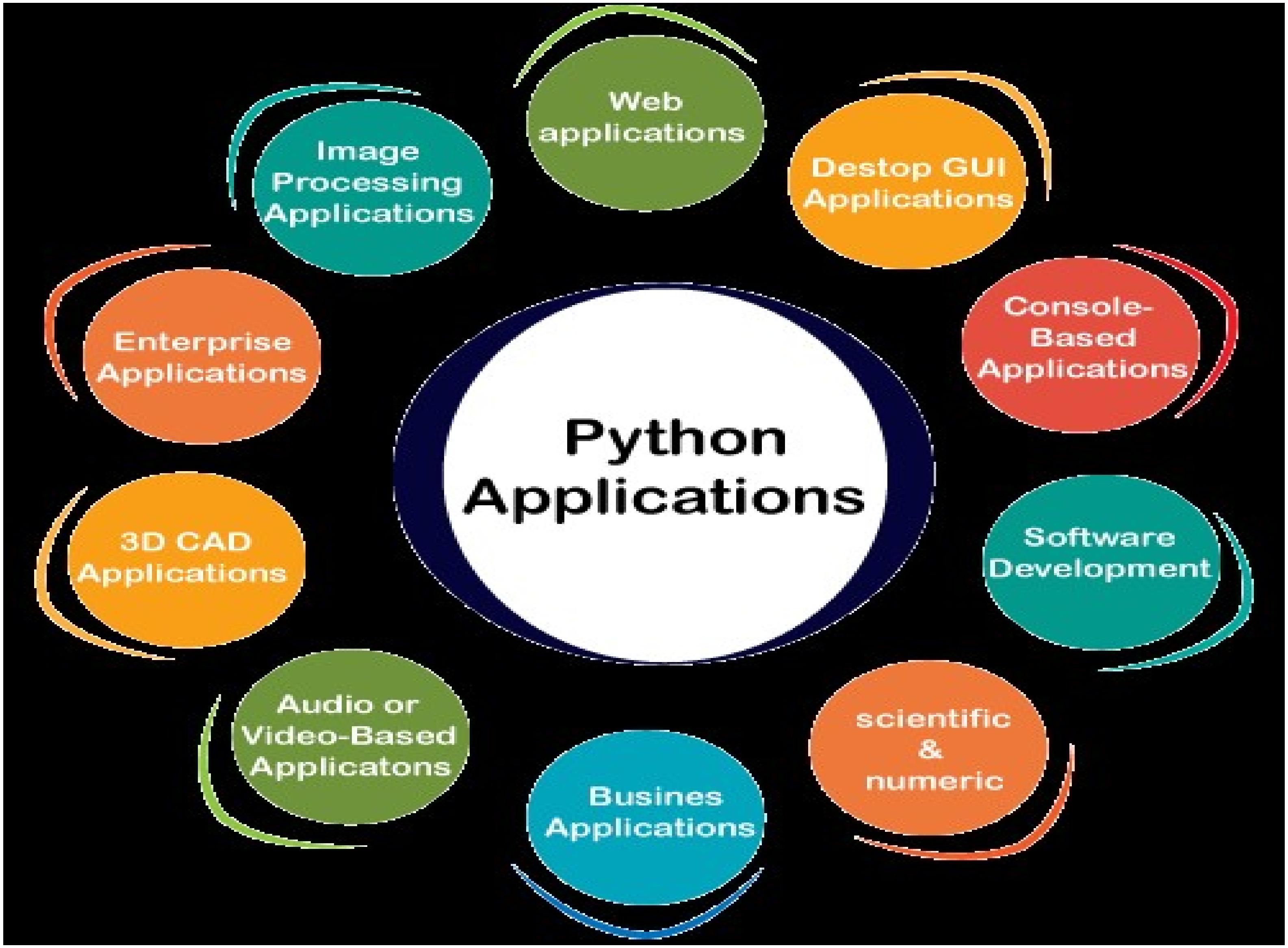
Python Advantages and Disadvantages

Advantages

- Improved Productivity
- Interpreted Language
- Dynamically Typed
- Free and Open Source
- Vast Libraries Support

Disadvantages

- Slow Speed
- Not Memory Efficient
- Weak in Mobile Computing
- Database Access
- Runtime Errors



Python Applications

Web applications

Destop GUI Applications

Console-Based Applications

Software Development

scientific & numeric

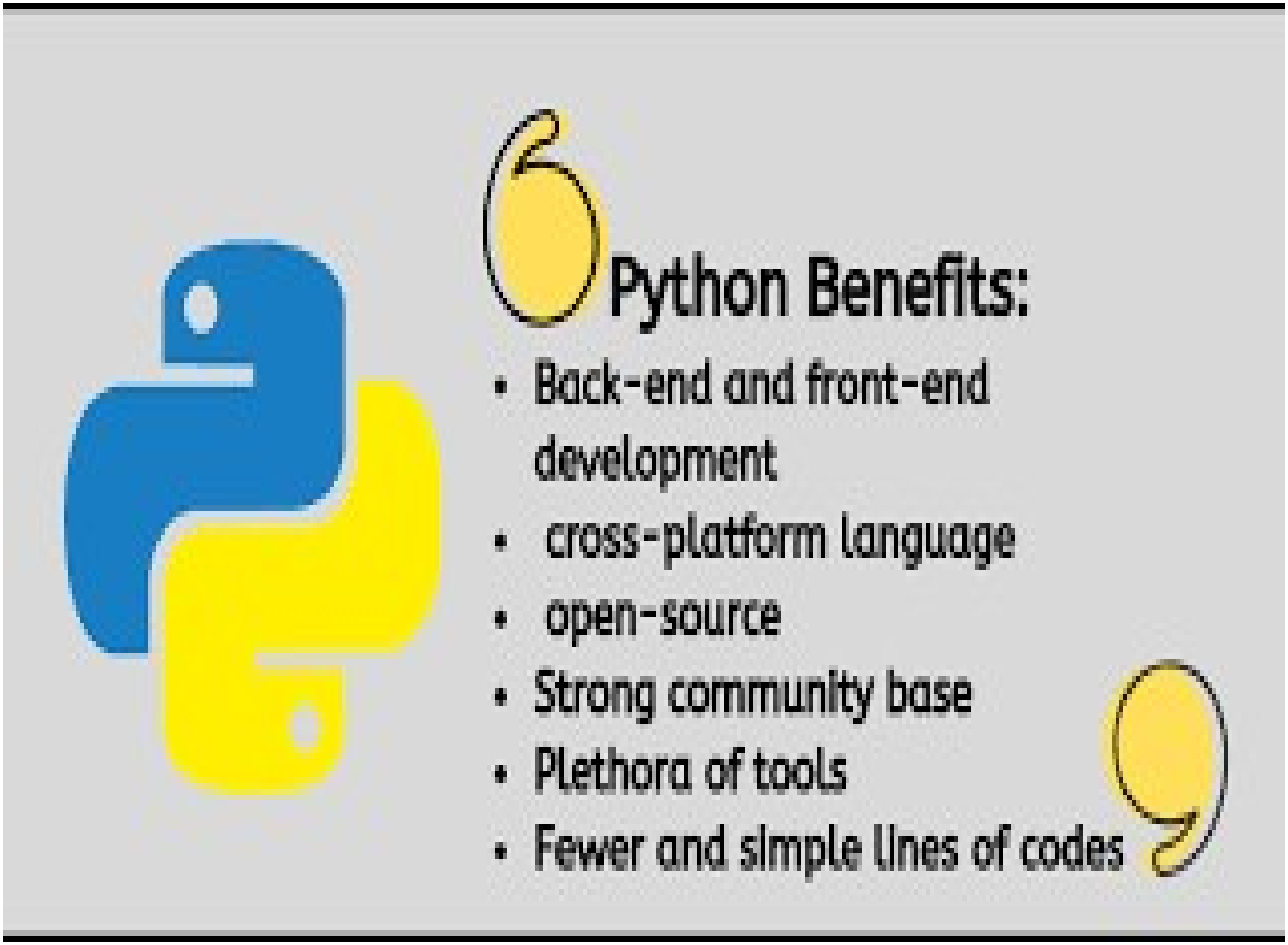
Busines Applications

Audio or Video-Based Applicatons

3D CAD Applications

Enterprise Applications

Image Processing Applications



Python Benefits:

- Back-end and front-end development
- cross-platform language
- open-source
- Strong community base
- Plethora of tools
- Fewer and simple lines of codes

Who found?

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991-92.
- Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a and then put those files into the python interpreter to be executed.
- text editor

Installation of Python

https://www.python.org/

Install 3.8.3

The Python Command Line

For Ubuntu – Python3 comes with package – No need to install again
Verify on terminal python3 - - version
Should show 3.10.3

Create file – welcome.py using gedit texteditior
Write one line program
Print(“Hello World”)
Save the file
For run python program – python3 welcome.py

Python Indentation

Indentation refers to the spaces at the beginning of a code line. **the indentation in Python is very important.**

With Proper Indentation Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Without Proper Indentation Example

```
if 5 > 2:
print("Five is greater than two!")
```

The first line with *less* indentation is outside of the block.

The first line with *more* indentation starts a nested block

To understand code

A : Assignment uses = and comparison uses ==,

B: Numbers + - * / %

C: Use of ***+*** for string concatenation

D: use of ***%*** for string formatting

E: Logical operators (and, or, not)

F: The basic printing command is print

G: Variable types don’t need to be declared, Python figures out the variable types on its own.
a = 5 (Integer). Int = 5, float = 5.5
b = 5.5 (float)
C = “mumbai”

Comments

A : Explain Python code
B : Code more readable
C : Prevent execution when testing code

Comments starts with a #, and Python will ignore them

Comments can be placed at the end of a line, and Python will ignore the rest of the line

To add a multiline comment you could insert a # for each line

```
# this code explains addition of 2 numbers
#  this is written by Ramesh dated 21-june 2020
#print("Hello, World!")

print(“Hello, World!”) # this line explains printing hello world
```

Variables

A : Storing data values

B: A variable is created the moment you first assign a value to it.

F = 5 c = 7

Variable Naming Convention

A variable name must start with a letter or the underscore character

A, city=7, = 8_city

A variable name cannot start with a number 1a = 5

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (city, City and CITY are three different variables)

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Variables

```
x = “Best"

def myfunc():
  print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local.

```
x = “The Best“  - Global  variable

def myfunc():
  x = “Easy“  - Local variable
  print("Python is " + x)  = python is easy

myfunc()

print("Python is " + x)  = Python is best
```

Global Keyword

```
def myfunc():
  global x  - global Keyword
  x = "fantastic"

myfunc()

print("Python is " + x)
```

Keywords

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers

All the keywords except True, False and None are in lower case

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class

continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator

or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

Python Identifiers

An identifier is a name given to entities like class, functions, variables.

Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
- An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
- Keywords cannot be used as identifiers

Literals/Values

A **literal** is a visible way to write a value

choices of **types** of literals are often integers, floating point, Booleans and character strings.

- String literals :: "halo" , '12345'
- Int literals :: 0,1,2,-1,-2
- Long literals :: 89675L
- Float literals :: 3.14
- Complex literals :: 12j
- Boolean literals :: True or False
- Special literals :: None
- Unicode literals :: u"hello"
- List literals :: [], [5,6,7]
- Tuple literals :: (), (9,),(8,9,0)
- Dict literals :: {}, {'x':1}
- Set literals :: {8,9,10}

Punctuators

Symbols that are used in python to organise sentence structures used in evaluation of expressions , statements and Program structure

Most common examples

‘ “ # \ () [] {} @ : =

A = [1,2,3,4]

# Comemnts

\n – new line in program

A = {1,2,3,4}

If (condition)

Type Casting

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.

- Implicit Type Conversion
- Explicit Type Conversion

Implicit type conversion

Python automatically converts one data type to another data type. This process doesn't need any user involvement.

```
num_int = 123
num_flo = 1.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Explicit Type Conversion

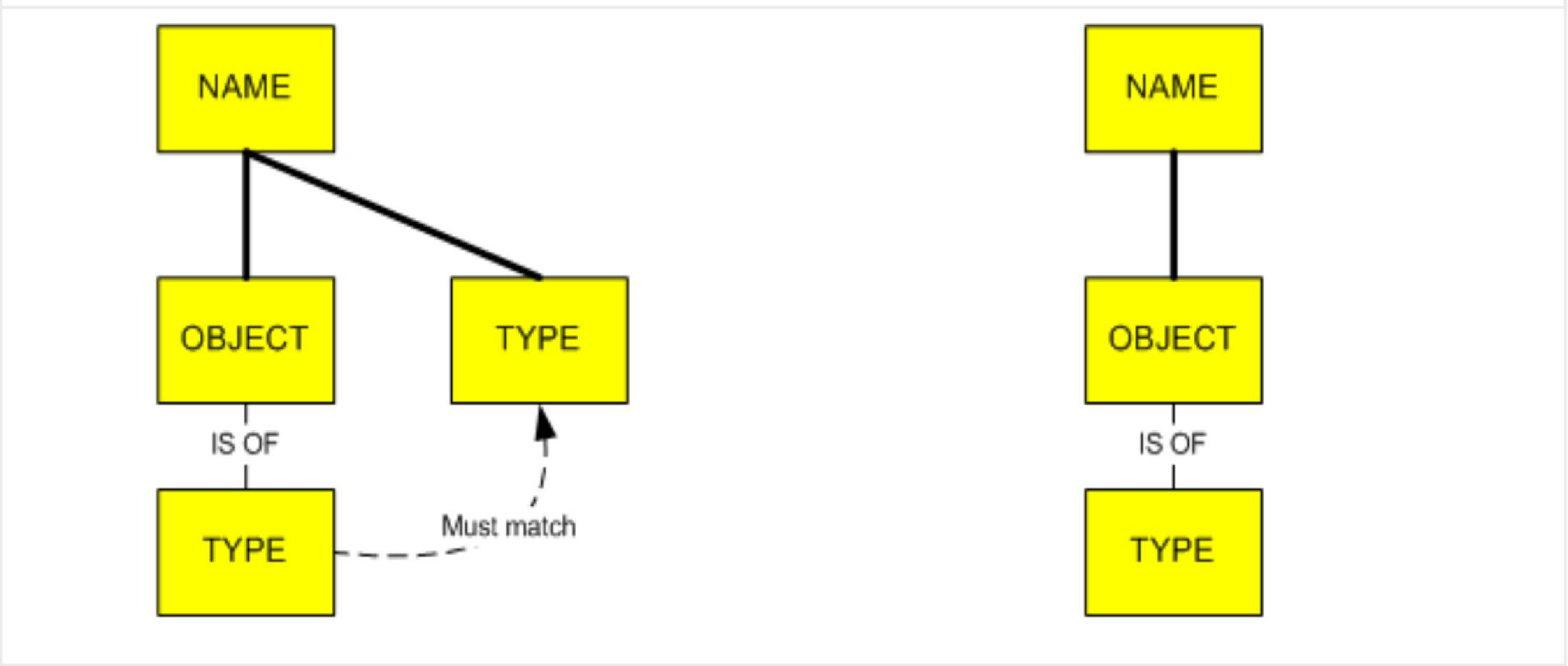
Convert the data type of an object to required data type.

This type of conversion is also called typecasting because the user casts (changes) the data type of the objects

Example

```
num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Dynamic Typing



```
I = 5
I = "Bangalore"
```

Math Library

The Python Math Library provides us access to some common math functions and constants in Python, which we can use throughout our code for more complex mathematical computations.

Special Constants

A : Pie

B : Euler's Number - base of natural logarithm

```
import math
radius = 4
print('The area of a circle with a radius of 4 is:', math.pi * (radius ** 4))
```

```
import math
print((math.e + 6 / 2) * 4.32)
```

Exponents and Logarithms

A : The exp() Function

calculate the power of e. For example, e^x, which means the exponential of x. The value of e is 2.718281828459045.

```
import math
# Initializing values
an_int = 2
a_neg_int = -3
a_float = 5.00
# Pass the values to exp() method and print
print(math.exp(an_int))
print(math.exp(a_neg_int))
print(math.exp(a_float))
```

Exponents and Algorithms

B : **The log() Function**

This function returns the logarithm of the specified number. The natural logarithm is computed with respect to the base e.

```
import math
print("math.log(9.43):", math.log(8.43))
print("math.log(10):", math.log(10))
print("math.log(math.pi):", math.log(math.pi))
```

Exponents and Algorithms

B : **The log() Function**

This function returns the logarithm of the specified number. The natural logarithm is computed with respect to the base e.

```
import math
print("math.log(9.43):", math.log(8.43))
print("math.log(10):", math.log(10))
print("math.log(math.pi):", math.log(math.pi))
```

Exponents and Algorithms

C: **The log10() Function**

returns the base-10 logarithm of the specified number

```
import math # Returns the log10 of 50
print("The log10 of 50 is:", math.log10(50))
```

Exponents and Algorithms

D: **The log2() Function**

logarithm of a number to base 2

```
import math
# Returns the log2 of 16

print("The log2 of 16 is:", math.log2(16))
```

Exponents and Algorithms

E: **log(x, y) Function**

```
import math
# Returns the log of 3,4
print("The log 3 with base 4 is:", math.log(3, 4))
```

Exponents and Algorithms

F: **log1p(x) Function**

calculates the logarithm(1+x)

```
import math
print("Logarithm(1+x) value of 10 is:", math.log1p(10))
```

Arithmetic Functions

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns x * (2**i)
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1
log(x[, base])	Returns the logarithm of x to the base (defaults to e)
log1p(x)	Returns the natural logarithm of 1+x
log2(x)	Returns the base-2 logarithm of x

log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
acos(x)	Returns the arc cosine of x
asin(x)	Returns the arc sine of x
atan(x)	Returns the arc tangent of x
atan2(y, x)	Returns atan(y / x)
cos(x)	Returns the cosine of x
hypot(x, y)	Returns the Euclidean norm, sqrt(x*x + y*y)
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
degrees(x)	Converts angle x from radians to degrees
radians(x)	Converts angle x from degrees to radians
acosh(x)	Returns the inverse hyperbolic cosine of x
asinh(x)	Returns the inverse hyperbolic sine of x
atanh(x)	Returns the inverse hyperbolic tangent of x
cosh(x)	Returns the hyperbolic cosine of x
sinh(x)	Returns the hyperbolic cosine of x
tanh(x)	Returns the hyperbolic tangent of x
erf(x)	Returns the error function at x
erfc(x)	Returns the complementary error function at x
gamma(x)	Returns the Gamma function at x
lgamma(x)	Returns the natural logarithm of the absolute value of the Gamma function at x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

Python – Operators

EXCELSOFT

L Value and R value

- Age (L Value) = 25 Years (R Value)
- L Value – Assigned Object Ex Variable name
- R value – expression that has value.

Python Arithmetic Operators

+ Addition	Adds values on either side of the operator. x = 10,y = 22, print(x+y)
- Subtraction	Subtracts right hand operand from left hand operand. x = 10,y = 22, print(x-y)
* Multiplication	Multiplies values on either side of the operator x = 10,y = 22, print(x*y)
/ Division	Divides left hand operand by right hand operand x = 10,y = 22, print(x/y)
% Modulus	Divides left hand operand by right hand operand and returns remainder x= 5, y= 2, print(x%y)
** Exponent	Performs exponential (power) calculation on operators
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. x = 5, y=2, print(x//y)

Practice in Class

Write a program to calculate total price of an item with following fields from user

A : Enter item name

B : Enter Item Price

C : Qty

D : GST 18%

E : Total Price to pay

Program

```
# this program calculates the total price inclusive of GST for an Item#
x = float(input(“enter price for TV:”))
Y = float(input(“enter quantity to buy:”))
#Total Price before tax
p= x * y
#Total price after tax
Z = p + p*18/100
Print(z)
```

Comparison Operators

Operator	Description
==	If the values of two operands are equal, then the condition becomes true. X = 10, y= 20 print (x == y) returns False
!=	If values of two operands are not equal, then condition becomes true. X = 10, y= 20 print (x!=y) returns True
>	If the value of left operand is greater than the value of right operand, then condition becomes true. X = 10, y= 20 print (x>y) returns False
<	If the value of left operand is less than the value of right operand, then condition becomes true. X = 10, y= 20 print (x<y) returns True
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. X = 10, y= 20 print (x>=y) returns False
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true. X = 10, y= 20 print (x<=y) returns True

Logical Operators

Operator	Description
and Logical AND	If both the operands are true then condition becomes true. X = 8 Print (x> 5 and x < 10)
or Logical OR	If any of the two operands are non-zero then condition becomes true. X = 8 Print (x> 5 or x < 10)
not Logical NOT	Used to reverse the logical state of its operand. X = 8 Print (not(x> 5 and x < 10))

Logical Operators

AND (0 – False, 1 – True) - &

True True = True

True False = False

False True = False

False False = False

OR

Any one condition is true then true otherwise false.

NOT

Reverse of &/OR result

Assignment Operators

Operator	Description
=	Assigns values from right side operands to left side operand x = 50, print(x)
+= Add AND	It adds right operand to the left operand and assign the result to left operand x= 50, x += 10 print(x)
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand x= 50, x -= 10 print(x)
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand x= 5, x *= 10 print(x)
/= Divide AND	It divides left operand with the right operand and assign the result to left operand x= 5, x /= 10 print(x)
%= Modulus AND	It takes modulus using two operands and assign the result to left operand x= 5, x %= 10 print(x)
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand x= 5, x **= 10 print(x)
//= Floor Division	It performs floor division on operators and assign value to the left operand x= 5, x //= 10 print(x)

Membership Operators

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise x = [“Cat”,”Dog”,”Mouse”] Print(“Dog” in x)
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. x = [“Cat”,”Dog”,”Mouse”] Print(“snake”not in x)

Identity Operators

Operator	Description

is	Returns True if both variables are the same object
	x = [“cat”,”Dog”]
	Y = [“cat”,”dog”]
	Z =x
	Print (x is z), print(x is y)

is not	Returns True if both variables are not the same object
	x = [“cat”,”Dog”]
	Y = [“cat”,”dog”]
	Z =x
	Print (x is not z), print(x is not y)

Bitwise Operators

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	Bitwise Ones' Complement Operator
<<	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 Functions

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

function is defined using the def keyword

```
def my_function():
print("Hello from a function")
```

Arguments

Arguments are specified after the function name, inside the parentheses.

You can add as many arguments as you want, just separate them with a comma,

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called

```
def can1(“john”)
def can1(*name)
def my_function(fname):
  print(fname + "Message")
my_function("Gmail")
my_function("Yahoo")
my_function("Outlook")
```

Keyword arguments

arguments with the *key* = *value*

```
def func(a, b=5, c=10):
        print ('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Default Parameter Value

```
def my_function(country="India"):
print("I am from " + country)
my_function("Australia")
my_function("USA")
my_function( )
my_function("Russia")
```

Passing a List as an Argument

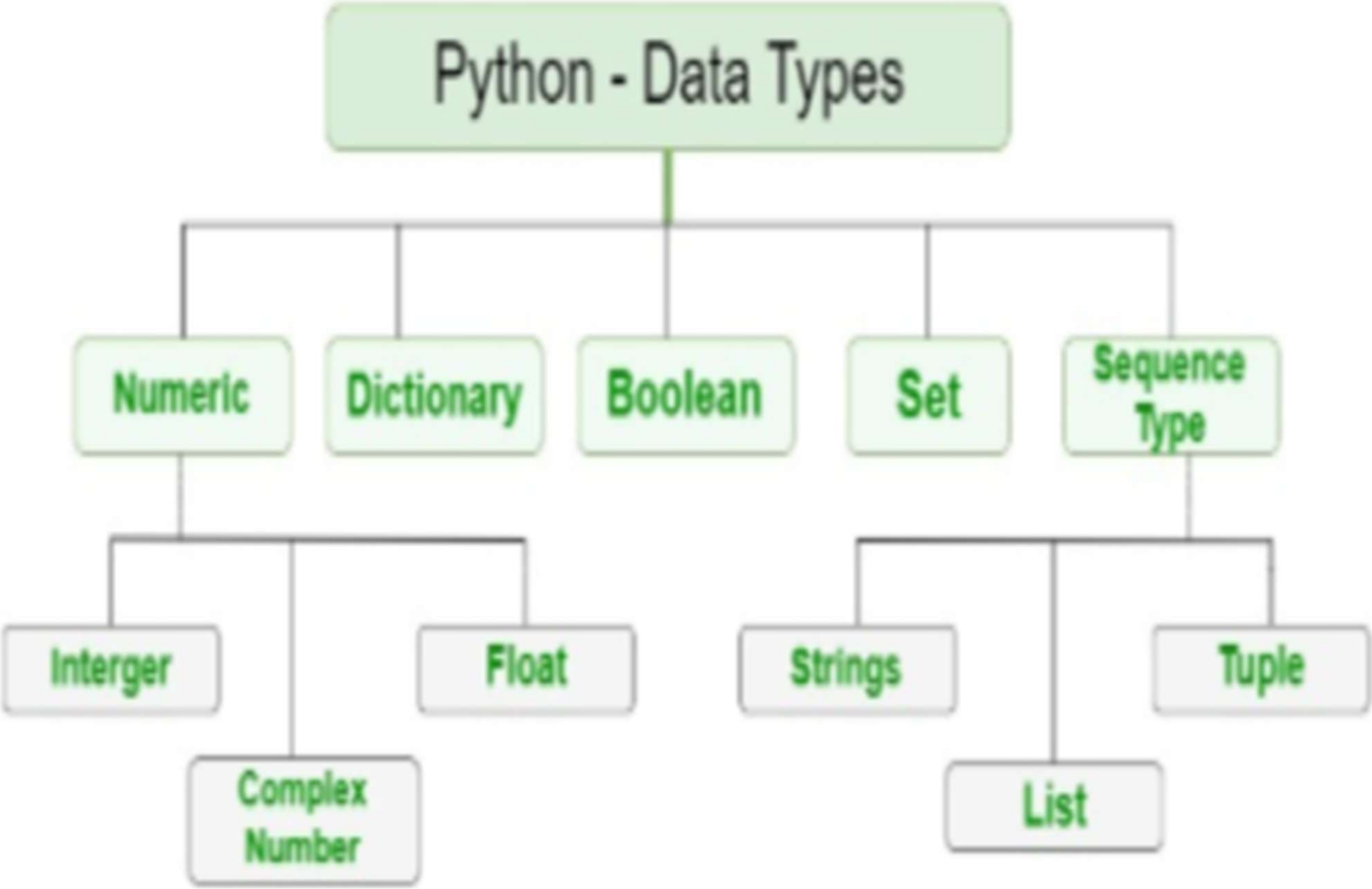
any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function

```
fruits = ["apple", "banana", "cherry"]

def my_function(fruits):
  for x in fruits:
    print(x)

my_function(fruits)
```

Return Values

function return a value, use the return statement

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```



Python - Data Types

Numeric

Dictionary

Boolean

Set

Sequence Type

Interger

Complex Number

Float

Strings

List

Tuple

Python Numeric

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal).
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*.

Get Type of Data

A :

```
a = 4
Print(“Type of a:”, type(a))


B = 5.0


C = 1+2j
```

Sequence - String

A string is a collection of one or more characters put in a single quote, double-quote or triple quote

‘bangalore’

“bangalore”

“”””bangalore””””

String Position

B	A	N	G	A	L	O	R	E
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

```
City = “BANGALORE”
Print(City[0])
Print(City[-1])
```

String – Slicing

return a range of characters by using the slice syntax

```
a = “BANGALORE”
Print(a[2:5])
Print(a[1:7])
Print(a[3:6])
```

Negative Indexing

Use negative indexes to start from end of string

```
a = “BANGALORE”
print(a[-5:-2])
Print(a[-1:-3])
```

String Length

To get the length of a string, use the len() function.

```
a = “BANGALORE”
Print(len(a))
```

Output = 9

String Methods

A: strip() method removes any whitespace from the beginning or the end

```
a = “     Bangalore  City        ”
Print(a.strip())
```

B : lower() method returns the string in lower case

```
a = “Bangalore  City”
Print(a.lower())
```

C: upper() method returns the string in upper case

```
a = “Bangalore  City”
Print(a.upper())
```

String Methods

D: replace() method replaces a string with another string:

```
a = “Bangalore  City”
Print(a.replace(“B”,”T”))
```

E: split() method splits the string into substrings if it finds instances of the separator

```
a = “Bangalore  City”
Print(a.split(“,”))
```

F: Check string

```
A = (“ Bangalore city opens”)
X = “lore” in a
Print(x)
```

String Methods

G: A = (“ Bangalore city opens”)
X = “lore” not in a
Print(x)

H: String Concatenation
a = “Bangalore”
b = “city”
c = a+b

I : String Concatenation with space
C = a + “” + b

String Format

To combine string and number by using the format() method.

format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Bangalore has 8 gardens and 100 lakes

```
a = 8
b = 100
Sen = “ bangalore has {} gardens and {} lakes”
Print(sen.format(a,b))
```

```
a = 8
b = 100
Sen = “ bangalore has {0} gardens and {1} lakes”
Print(sen.format(a,b))
```

Escape Character

To insert characters that are illegal in a string, use an escape character.

```
A = “ Bangalore is \”the best\” city “
Print(a)
```

Conditional Statements

EXCELSOFT

Definition

- Conditional Statement in Python perform different computations or actions.
- Evaluates to true or false
- Conditional statements are handled by IF statements in Python

IF Statement

If Statement is used for decision making. It will run the body of code only when IF statement is true.

```
if [boolean expression]:
statement1
statement2
...
Statement N
```

Any Boolean expression evaluating to True or False appears after the if keyword. Use the : symbol and press Enter after the expression to start a block with increased indent. One or more statements written with the same level of indent will be executed if the Boolean expression evaluates to True

If-Example

Assume that we have to write a Python program that calculates the amount payable from price and quantity inputs by the user and applies a 10% discount if the amount exceeds 1000.

```
price=int(input("Enter Price: "))
qty=int(input("Enter Quantity: "))
amt=price*qty
if amt>1000:
print ("10% discount is applicable")
 discount=amt*10/100
amt=amt-discount
print ("Amount payable: ",amt)
```

If-Else

When you want to justify one condition while the other condition is not true, then you use "if else statement”

if [boolean expression]:

statement1 statement2 ... statementN

else:

statement1 statement2 ... statementN

If else example

Assume that we have to write a Python program that calculates the amount payable from price and quantity inputs by the user and applies a 10% discount if the amount exceeds 1000.

```
price=int(input("Enter Price: "))
qty=int(input("Enter Quantity: "))
amt=price*qty
if amt>1000:
print ("10% discount is applicable")
Else:
Print(“10% discount not applicable”)
```

elif Condition

Use the elif condition is used to include multiple conditional expressions between if and else.

```
if [boolean expression]:
[statements]
elif [boolean expresion]:
[statements]
elif [boolean expresion]:
[statements]
elif [boolean expresion]:
[statements]
else:
[statements]
```

EIF Example

```
X = input(“Enter a number”)
if x==10:
print('X is 10')
elif x==20:
print('X is 20')
elif x==30:
print('X is 30')
else:
print('X is something else')
```

While loop

A program, by default, follows a sequential execution of statements.

Python uses the while and for keywords to constitute a conditional loop, by which repeated execution of a block of statements is done until a Boolean expression is true.

```
while [boolean expression]:
statement1
statement2
...
Statement N
```

While Example

```
num =0
while num< 50:  (Max value – 5) Min value = 0
        num = num + 1 # Incrementing 1,2,3,4,5
                = 0 +1 = 1 # print Value
                = 1 +1 =  2 # print Value
                = 2 + 1 = 3 # print Value
                = 3 + 1 = 4 # print Value
                = 4 + 1 = 5 # print Value
        print("num =", num)
```

For loop

The for loop is used with sequence types such as list, tuple and set. The body of the for loop is executed for each member element in the sequence.

doesn't require explicit verification of Boolean expression controlling the loop

```
for x in sequence:
statement1
statement2
...
Statement N
```

For loop Example

```
A = [100,200,300,400,500]
for i in A:
  Print(i)
```

Break::

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
   break
```

Apple and banana

Continue

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Apple and Cherry

The range() Function

loop through a set of code a specified number of times, we can use the range() function

```
for x in range(6):
  print(x)
```

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

```
city = [“Bangalore", “Indore", “Delhi"]
fruits = ["apple", "banana", "cherry"]

for x in city:
  for y in fruits:
    print(x, y)
```

Thank you

Question and Answer

EXCELSOFT