



Introduction to Python





What is Programming



- ✓ Programming is the mental process of thinking up instructions to give to a machine (like a computer).
- ✓ Programming is a collaboration between humans and computers.
- ✓ Programming, also known as coding, refers to the process of writing instructions for computing devices and systems. A computer program translates those instructions into a language that computers can understand.





Python Programming Language



- ✓ *Python is a general purpose high level programming language.*
- ✓ *Python was developed by Guido Van Rossum in 1989 while working at National Research Institute at Netherlands.*
- ✓ *But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.*
- ✓ *Python is recommended as first programming language for beginners.*





One and Only



GUIDO VAN ROSSUM
Creator of Python



Python Programming Language



- ✓ *Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.*



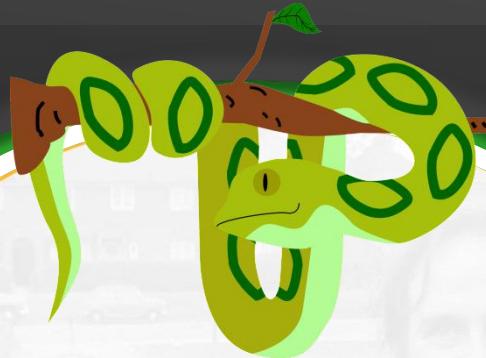
Why is Python Called Python ?



- While *Guido van Rossum* was implementing Python, he was also reading the published scripts from Monty Python's Flying Circus.
- Monty Python's Flying Circus is a BBC Comedy TV series from the year 1969+. It is a highly viewed TV series and is rated 8.8 in IMDB.
- Python programming language is highly rated too. According to a recent Stackoverflow survey, Python has overtaken Java in popularity.



Why is Python Called Python ?

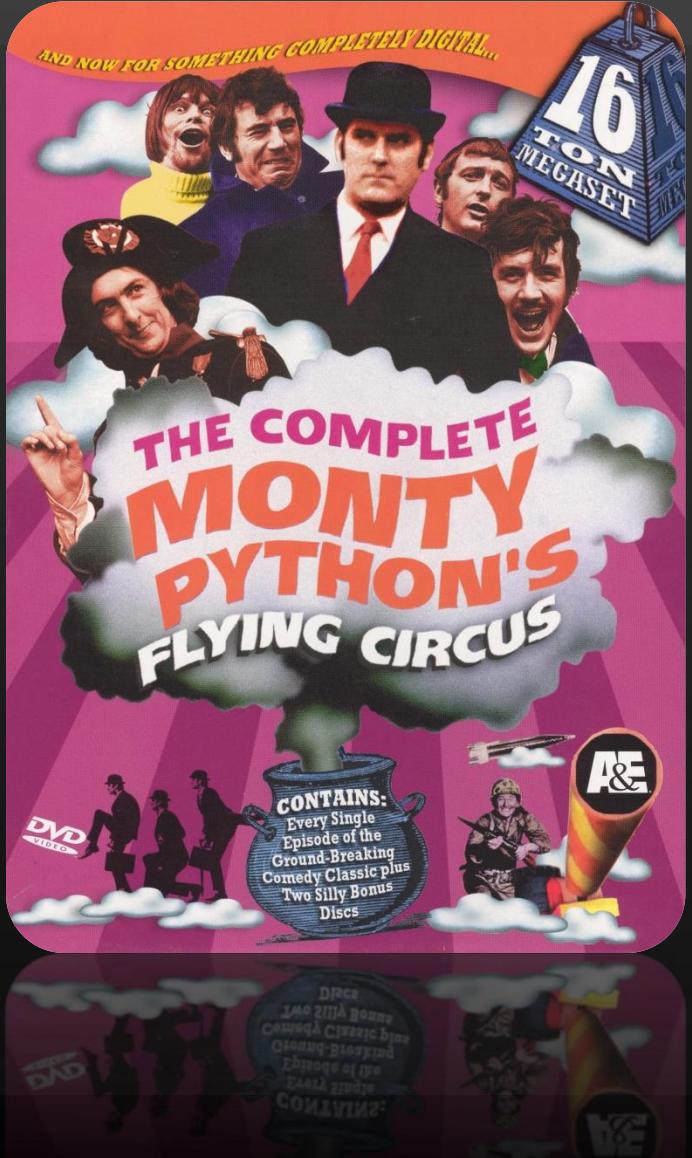


DID YOU KNOW ?

The Python programming language isn't named after snakes but after the popular British comedy troupe Monty Python



Why is Python Called Python ?





Why Learn Python ?



- ✓ *Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, 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 language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc.*





Why Learn Python ?



*Internally Google and You tube use Python coding
NASA and New York Stock Exchange Applications developed
by Python.*

*Top Software companies like Google, Microsoft, IBM, Yahoo
using Python.*

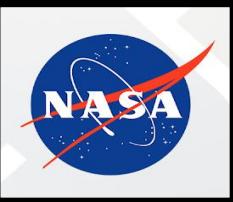




The Top Giants



16 Famous Companies that uses PYTHON





Where We Can use Python ?

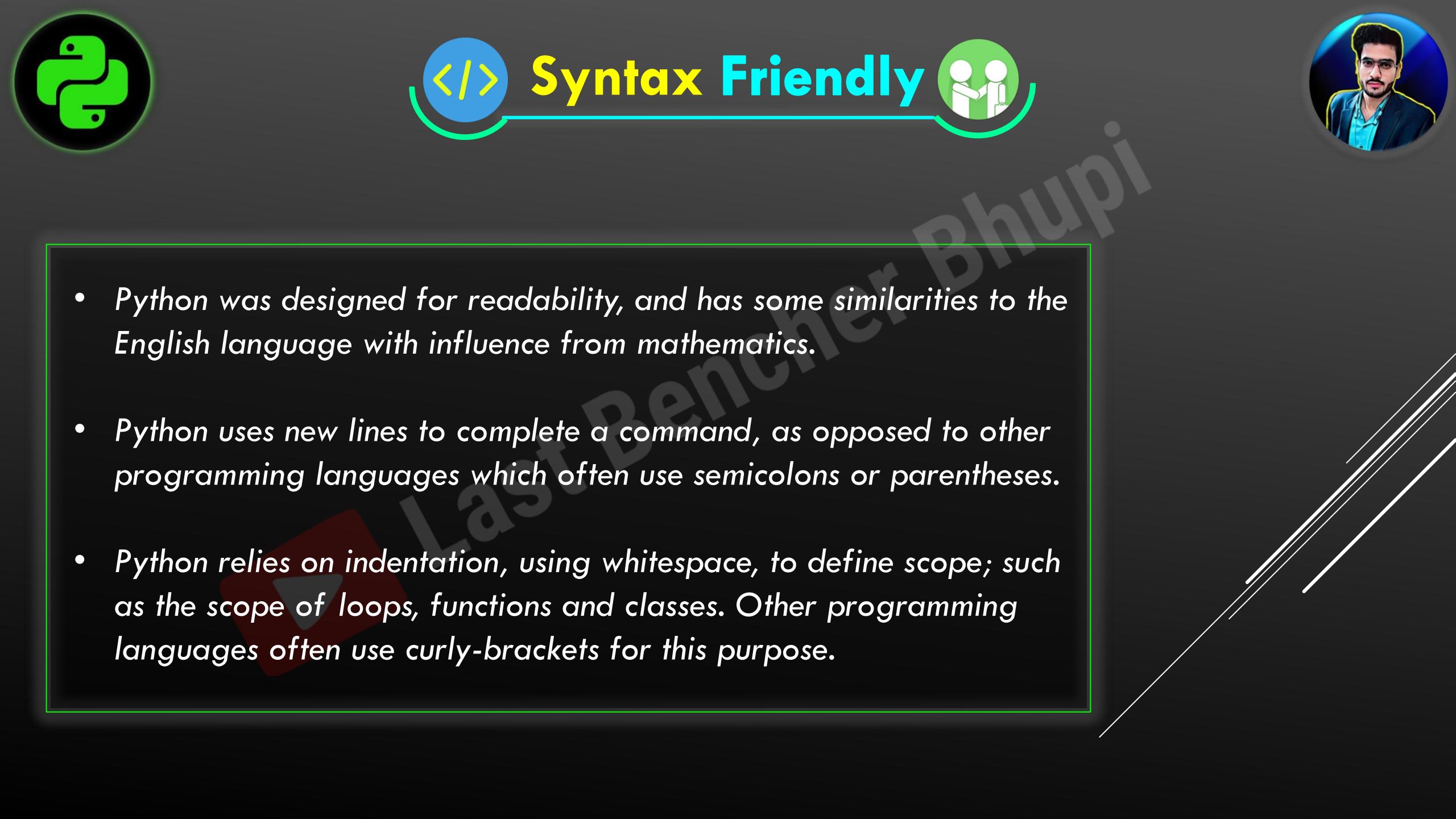




Where We Can use Python ?

- 1. For developing Desktop Applications**
- 2. For developing web Applications**
- 3. For developing database Applications**
- 4. For Network Programming**
- 5. For developing games**
- 6. For Data Analysis Applications**
- 7. For Machine Learning**
- 8. For developing Artificial Intelligence Applications**
- 9. For IOT**





- *Python was designed for readability, and has some similarities to the English language with influence from mathematics.*
- *Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.*
- *Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.*



Syntax Friendly



JAVA

C

PYTHON

```
class HelloWorld
{
    public static void main(String []args)
    {
        System.out.println("Hello, World!");
    }
}
```

```
int main()
{
    printf("Hello, World!");
    return 0;
}
```

```
print("Hello, World!")
```



Syntax Friendly



JAVA

```
class Add
{
    public static void main(String[] args)
    {
        int a,b;
        a = 10;
        b = 20;
        System.out.println("The Sum:"+ (a+b));
    }
}
```

C

```
void main()
{
    int a,b;
    a = 10;
    b = 20;
    printf("The Sum:%d", (a+b));
}
```

PYTHON

```
a = 10
b = 20
print("The Sum:",(a+b))
```



Pre Requisites of Learning Python



- There are **no prerequisites** to learn Python but a little bit of knowledge of any programming language like what is a loop, what if and else does, how operators are used. If you know the basics of any programming language, it would be easy to learn Python.
- You should have a **basic understanding** of Computer Programming terminologies. A basic understanding of any of the programming languages is a plus.





Features of Python

1

EASY

2

EXPRESSIVE

3

OPEN SOURCE

7

PLATFORM INDEPENDENT

4

HIGH LEVEL

5

EXTENSIVE LIBRARY

6

DYNAMICALLY TYPED



Features of Python



Simple and easy to learn:

Python is a simple programming language. When we read Python program, we can feel like reading English statements. The syntaxes are very simple and only 30+ keywords are available. When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity. We can reduce development and cost of the project.





Features of Python

Expressive - Python is an expressive language. Expressive in this context means that a single line of Python code can do more than a single line of code in most other languages. The advantages of a more expressive language are obvious – the fewer lines of code you write, the faster you can complete the project

Freeware and open source - We can use Python software without any license and it is freeware. Its source code is open, so that we can customize based on our requirement.





Features of Python

High Level Programming language - Python is high level programming language and hence it is programmer friendly language. Being a programmer we are not required to concentrate low level activities like memory management and security etc..

Platform Independent - Once we write a Python program, it can run on any platform without rewriting once again. Internally PVM is responsible to convert into machine understandable form.





Features of Python



Dynamically Typed - In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language.

But Java, C etc are *Statically Typed Languages* b'z we have to provide type at the beginning only.

Extensive Library - Python has a rich inbuilt library.

Being a programmer we can use this library directly and we are not responsible to implement the functionality.

Huge
Libraries



Limitations of Python



We discussed above that Python is an interpreted language and dynamically-typed language. The line by line execution of code often leads to **slow execution**.

It is very rarely used for mobile development. Python is not a very good language which means it is a **weak language for mobile development**.





High Level V/s Low Level Language



High Level Language	Low Level Language
Easy for humans to read , write and modify	Hard for Humans to read, write and modify
Program run slower as they make worse use of the CPU and are not very memory efficient	Direct control over the CPU means memory use is more efficient and programs run faster
No understanding of how hardware components work is needed	Good understanding of how the different hardware components work is needed
Examples are python, java, C# , C++ etc.	Examples are machine code and assembly Language



Python is a Case Sensitive Language



Python is a Case Sensitive Language that means Uppercase and Lowercase letters would be treated differently.

Anything that is case sensitive discriminates between uppercase and lowercase letters. In other words, it means that two words that appear or sound identical but are using different letter cases, are not considered equal.

Example – Your password is case-sensitive, it must be written in a particular form using all capital letters or all small letters.

A screenshot of a login interface with a blue header. It features fields for 'Username' (containing 'username') and 'Password' (containing '*****'). There is a 'Remember Me' checkbox, a yellow padlock icon, and buttons for 'Login' and 'Register'.



Python Versions

Python 1.0V introduced in Jan 1994

Python 2.0V introduced in October 2000

Python 3.0V introduced in December 2008

*Note: Python 3 won't provide backward compatibility to Python2
i.e. there is no guarantee that Python2 programs will run in Python3.*

Current versions

Python 3.10.4 : Latest





Installation of Python



Python is a widely used high-level programming language. To write and execute code in python, we first need to install Python on our system.

Installing Python on Windows takes a Bunch of Seconds

Step 1 : - Go to <https://www.python.org/downloads/>

**Step 2 : - There you will see the Latest version of Python as of now it is
3.10.4**

Step 3 : - Click on Download Now

Now the Python Executable Controller is downloaded simply install it

Download the latest version for Windows

[Download Python 3.10.4](#)

Looking for Python with a different OS? Python for [Windows](#),
[Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#),
[Docker images](#)

Looking for Python 2.7? See below for specific releases



Intro To IDE



An *integrated development environment (IDE)* is a software application that provides comprehensive facilities to computer programmers for software development. An *IDE* normally consists of at least a source code editor, build automation tools and a debugger.



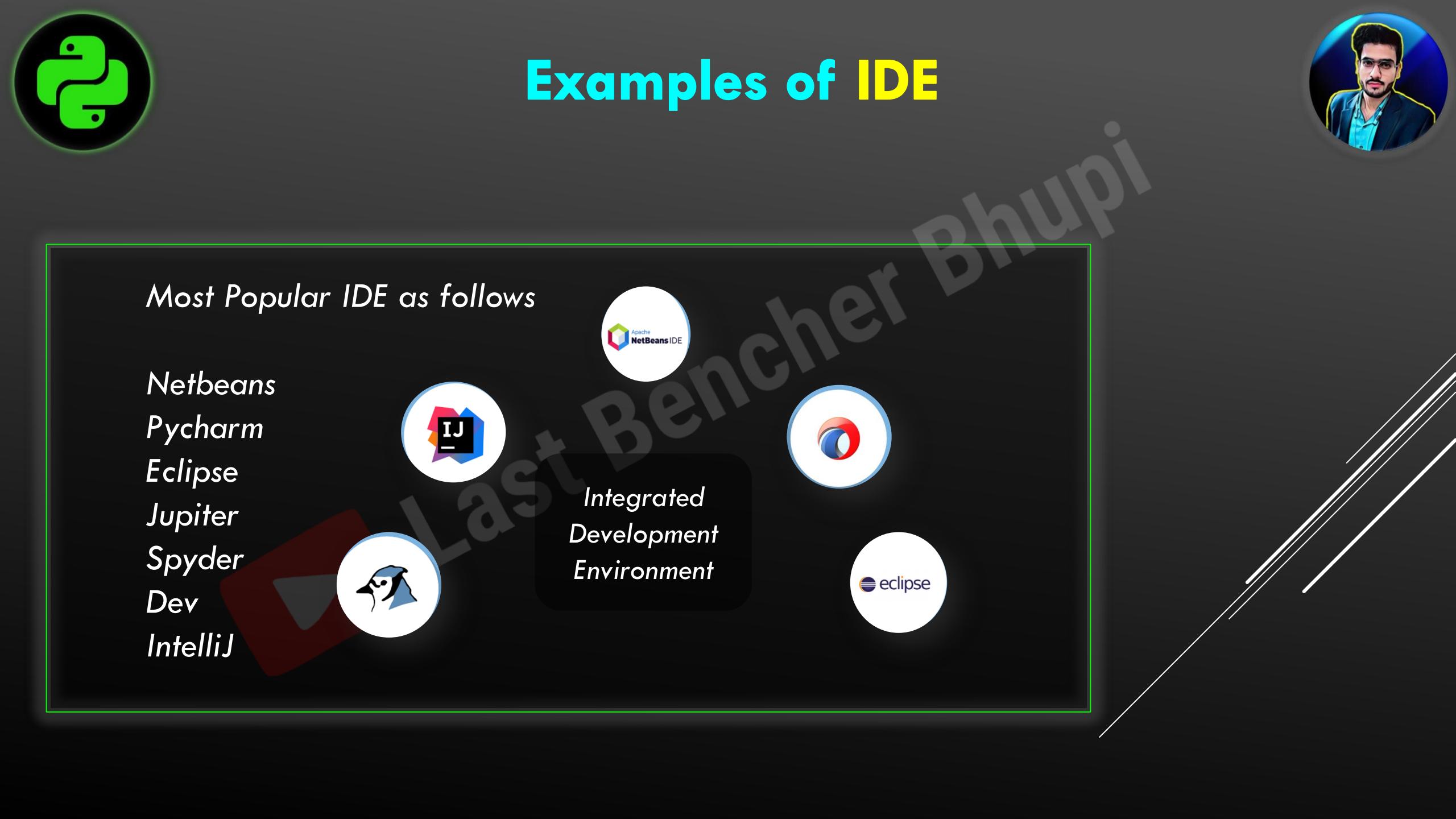


Intro To IDE



An *IDE* (or *Integrated Development Environment*) is a program dedicated to software development. As the name implies, IDEs integrate several tools specifically designed for software development. These tools usually include:

- An editor designed to handle code (with, for example, syntax highlighting and auto-completion)
- Build, execution, and debugging tools
- Some form of source control



Most Popular IDE as follows

Netbeans

Pycharm

Eclipse

Jupiter

Spyder

Dev

IntelliJ



*Integrated
Development
Environment*





Installation of IDE

Here, We are using Pycharm and inbuilt IDLE throughout our course but you can choose any IDE of your choice.

Steps to install -

1. Google it and download it from a trusted website
2. Then setup will download and install it in your PC/Laptop
3. Here is the icon automatically created in your Desktop Screen.





Setting Path In System



Before starting working with Python, a specific path is to set.

- 1** Your Python program and executable code can reside in any directory of your system, therefore Operating System provides a specific search path that index the directories Operating System should search for executable code.

- 2** The Path is set in the Environment Variable of My Computer properties:





Reserved Keywords

PYTHON



The keywords are some predefined and reserved words in python that have special meanings. The keyword cannot be used as an identifier, function, and variable name. All the keywords in python are written in lower case except

- True
- False
- None



There are **35 keywords** in Python as of Now



Reserved Keywords

PYTHON



You can get a *list* of available keywords by using **help()**:

```
Python
>>> help("keywords")

Here is a list of the Python keywords. Enter any keyword to get more help.

False      class      from       or
None       continue   global     pass
True       def        if         raise
and        del        import    return
as         elif       in        try
assert    else       is         while
async     except    lambda   with
await     finally   nonlocal yield
break
```



Reserved Keywords

PYTHON



you can use `help()` again by passing in the specific keyword that you need more information about. You can do this, for example, with the `pass` keyword:

```
Python >>>

>>> help("pass")
The "pass" statement
*****
pass_stmt ::= "pass"

"pass" is a null operation – when it is executed, nothing happens. It
is useful as a placeholder when a statement is required syntactically,
but no code needs to be executed, for example:

def f(arg): pass    # a function that does nothing (yet)

class C: pass      # a class with no methods (yet)
```



Keywords in a Programmatic Way



Python also provides a **keyword module** for working with Python keywords in a programmatic way. The keyword module in Python provides two helpful members for dealing with keywords:

- **kwlist** provides a list of all the Python keywords for the version of Python you're running.
- **iskeyword()** provides a handy way to determine if a string is also a keyword.



Keywords in a Programmatic Way



```
IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>> len(keyword.kwlist)
35
>>>
```



IDENTIFIERS

A name in Python program is called **identifier**.

It can be **class name** or **function name** or **module name** or **variable name**.

$$a = 10$$

Identifiers are case sensitive. of course Python language is case sensitive language.

So we know what a Python Identifier is. But can we name it anything?

Or do certain rules apply?

Well, we do have Some rules to follow when naming identifiers in Python:





IDENTIFIERS



Rule No 1

A Python identifier can be a combination of lowercase/ uppercase letters, digits, or an underscore. The following characters are valid:

Lowercase letters (a to z)

Uppercase letters (A to Z)

Digits (0 to 9)

Underscore (_)



IMPORTANT



IDENTIFIERS



Rule No 2

An identifier cannot begin with a digit.

_9python

__9python

python9

python_9

9python



IMPORTANT



IDENTIFIERS



Rule No 3

We cannot use special symbols in the identifier name. Some of these are :-

!

@

#

\$

%



IDENTIFIERS

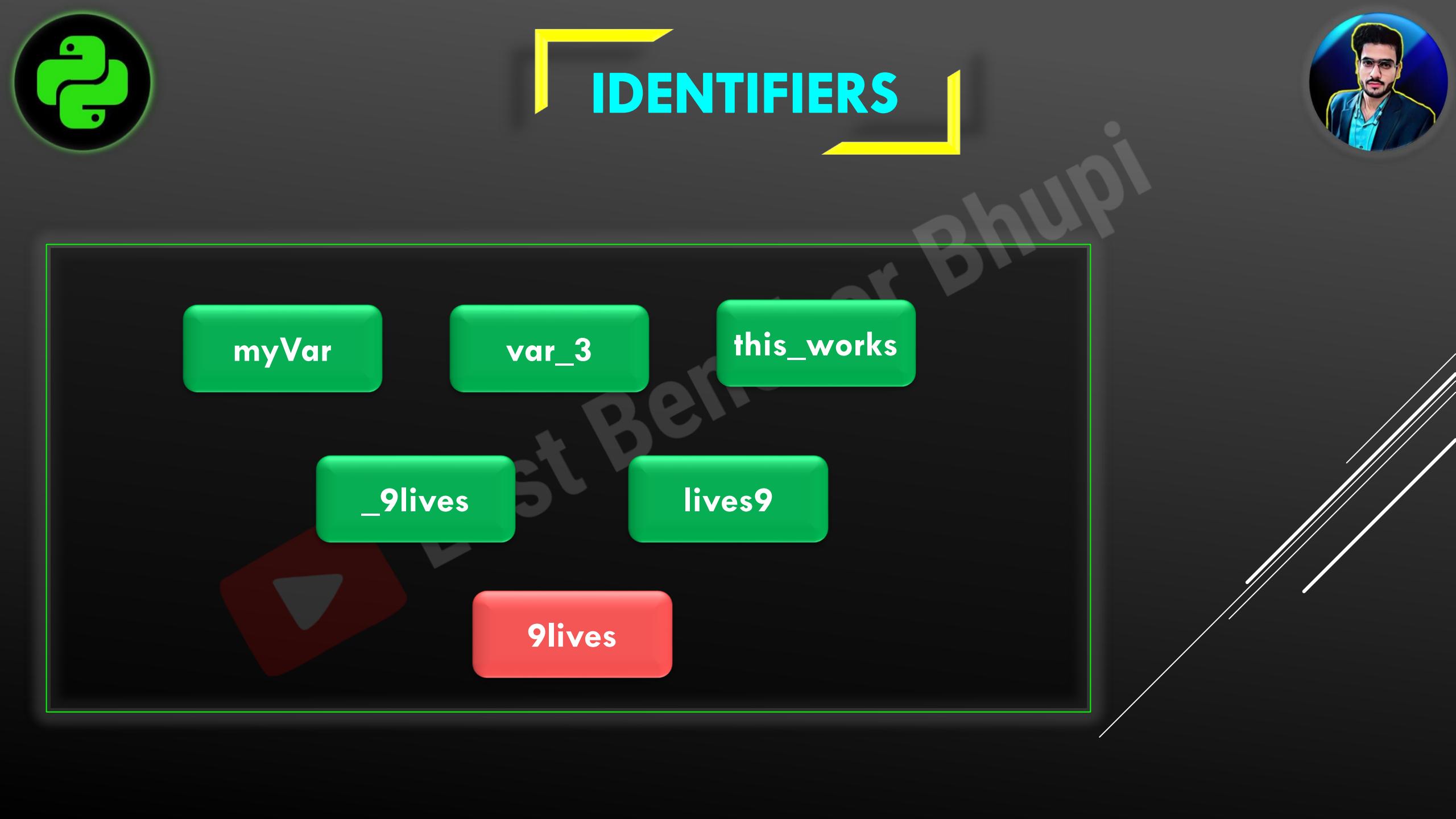


Rule No 4

We cannot use a keyword as an identifier.

Keywords are reserved names in Python and using one of those as a name for an identifier will result in a SyntaxError.

```
>>> for=50  
      SyntaxError: invalid syntax
```



IDENTIFIERS

myVar

var_3

this_works

_9lives

lives9

9lives



IDENTIFIERS



AbCd
aBcDa
abc123
ABC123
abc_123
ABC_123



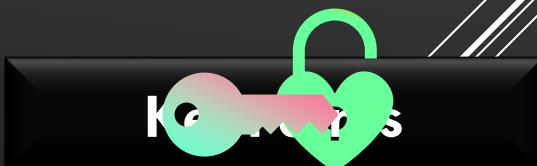
123abc
123_abc
while
1A2B
abc@123
\$abc

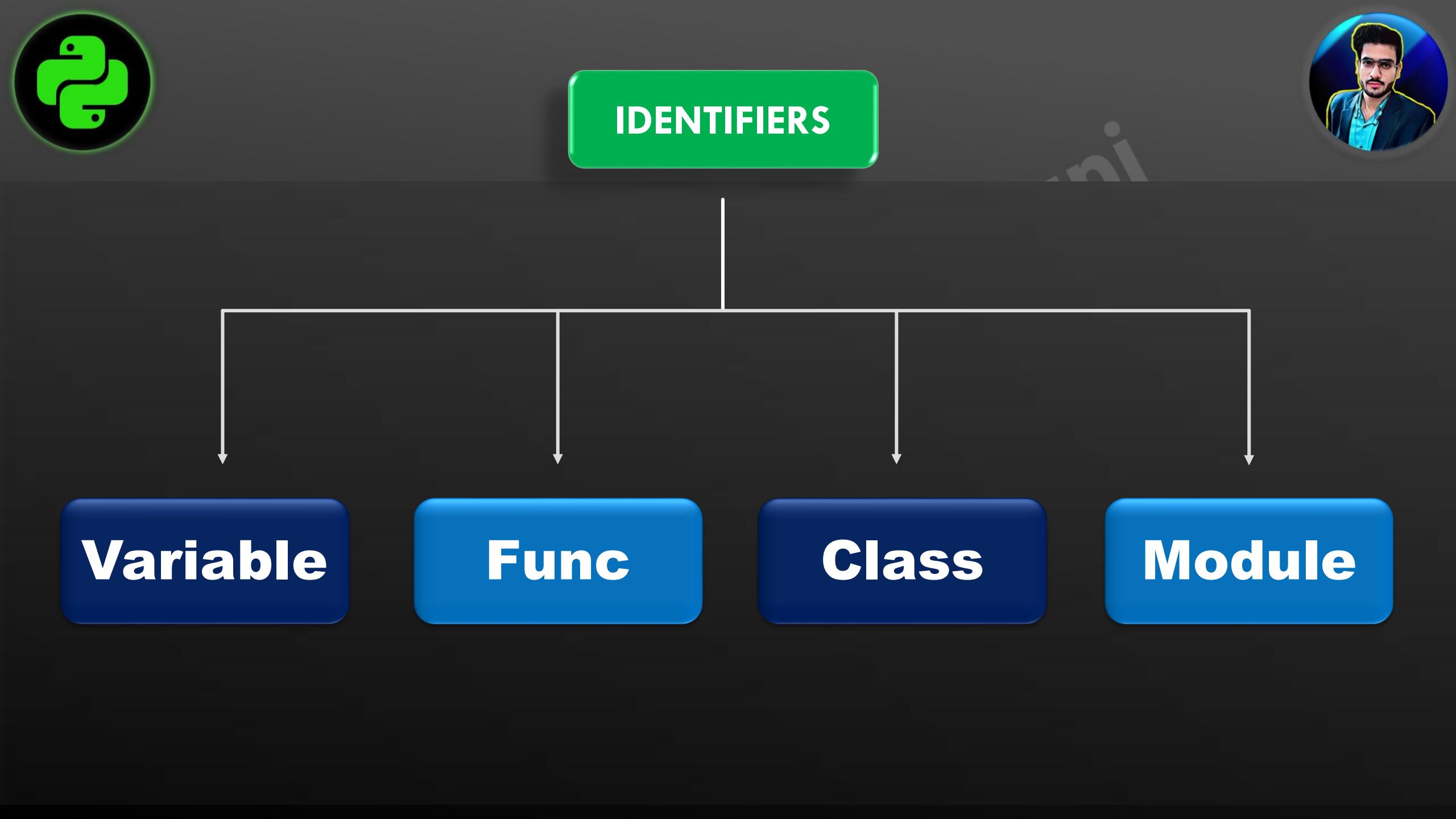


IDENTIFIERS Rules



- 1 *Alphabet Symbols (Either Upper case OR Lower case)*
- 2 *If Identifier is start with Underscore (_) is considered poor programming style*
- 3 *Identifier should not start with Digits and are case sensitive.*
- 4 *We cannot use reserved words as identifiers. Eg: def=10*
- 5 *There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.*
- 6 *Dollar (\$) Symbol is not allowed in Python.*





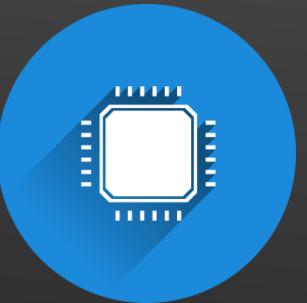
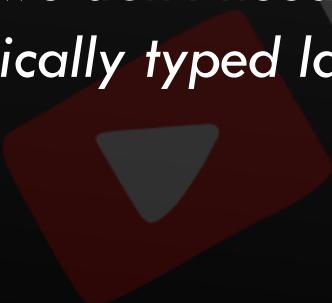


Variables



Variable is a name that is used to refer to memory location. Python variable is also known as an **identifier** and used to hold value.

In Python, we don't need to specify the type of variable because Python is a dynamically typed language and smart enough to get variable type.





Several Inbuilt Functions

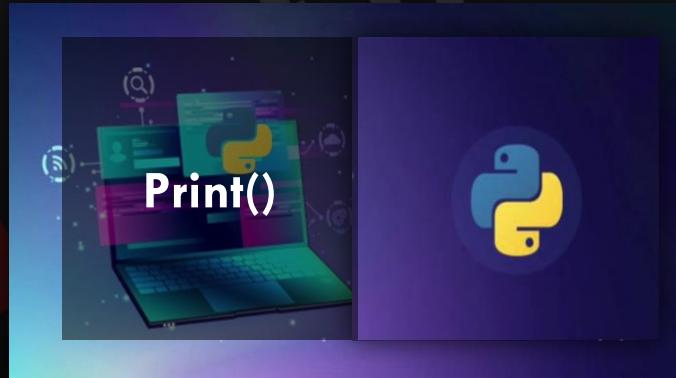


type()

To check the type of variable

For ex

```
A = 10  
type(A)
```





Several Inbuilt Functions



id()

to get address of object

For ex

A = 10

id(A)



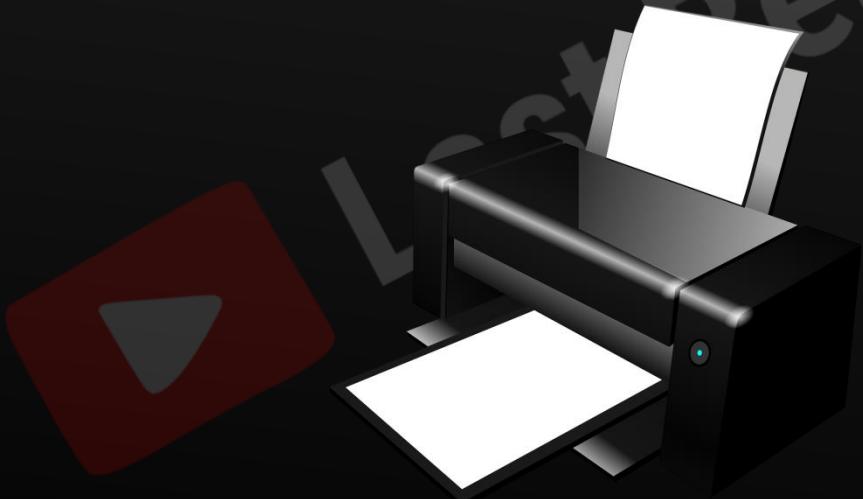


Several Inbuilt Functions



print()

print in Python is the standard function used to print the output to the console. Simply to print the value





ACTIVITY

TIME



Write your first python Program

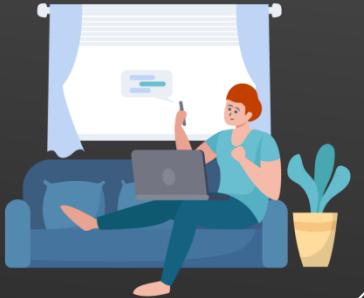


```
print("Hello World")
```



Attributes of print()

One of the most basic and popular functions in Python – the print() statement. We all might have started learning Python programming with the evergreen example of printing ‘Hello World’. The Python print() method is used to obtain output i.e. display anything on the screen as well as to debug code. This function outputs the supplied message or a value to the console





Attributes of print()



Suppose I Need to print this Statement in a single line with same Scenario

```
print("Hello Alia!")
```

```
print("How Are you")
```

```
$python main.py
```

```
Hello Alia!
```

```
How Are you
```

What should I do know



Attributes of print()



```
print("Hello Alia!")    print("How Are you")
```

```
$python main.py  
Hello Alia! How Are you
```



CONCLUSION



*If you are thinking that
Then python interpreter to you be like*





Attributes of print()



sep (Optional) – this specifies how objects should be separated if multiple objects are passed. Default value: ‘ ’

end (Optional) – specifies what is to be printed at last.
Default value: \n



End in print()



```
print("Hello Alia!",end=" ")  
print("How Are you")
```

```
$python main.py  
Hello Alia! How Are you
```



Sep in print()



```
print("19","Apr","2025",sep="-")
```

\$python main.py

19-Apr-2020





Comments in Python



Python Comment is a programmer's tool. We use them to explain the code, and the interpreter ignores them.

You never know when a programmer may have to understand code written by another and make changes to it.

Sometimes your brain will not remember what you have written and it might forget it.

For these purposes, good code will hold comments in the right places.





Comments in Python



Run this Program and analyze

```
# Single Line Comment  
  
''' This is a  
    Multi Line Comment '''  
  
print("Hello World")
```



Comments in Python

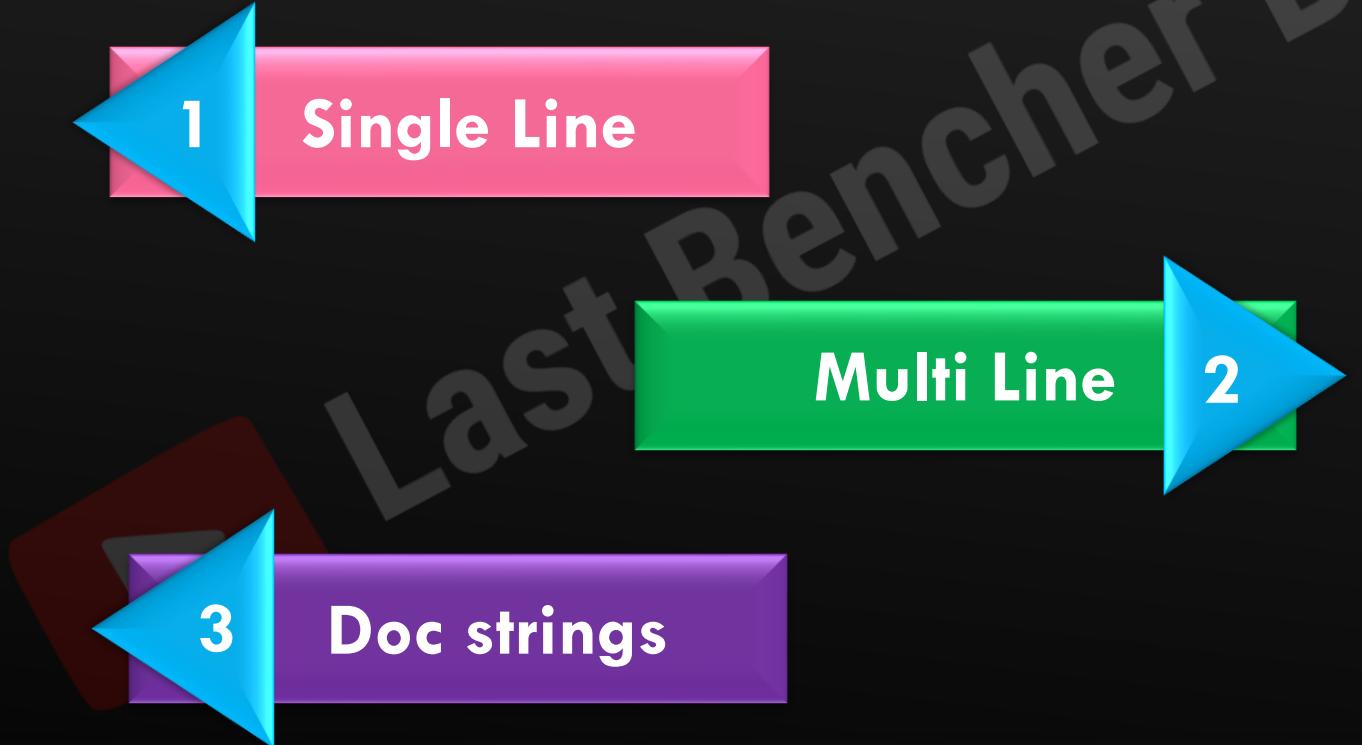


If you are thinking like that, this meme is perfectly match for you 😊





Types of Comments





Comments in Python



To Provide a Single Line comment we use (#)

If you want to provide multi line comments use (#) again in every line there is no other way to declare multi line string

```
# This is Single Line Comment  
  
# This is Multi Line Comment  
# This is Multi Line Comment  
# This is Multi Line Comment
```

1 Single Line



Comments in Python



Multiline Python Comment

To have *multiline python comment* in your code, you must use a hash at the beginning of every line of your comment in python.

```
# Comment 1  
# Comment 2  
# Comment 3  
# Comment 4
```



Multi Line 2



Comments in Python



Docstrings Python Comment

A *docstring* is a documentation string in Python. It is the first statement in a module, function, class, or method in Python.

Don't worry we will see function, methods, and classes later
It is just to aware

```
def square(a):
    '''Returned argument a is squared.'''
    return a**a
```

3 Doc strings



ESCAPE SEQUENCE



Escape sequences allow you to include special characters in strings. To do this, simply add a backslash (\) before the character you want to escape.

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

An escape character is a backslash \ followed by the character you want to insert.





ESCAPE SEQUENCE



```
str ="Katrina Kaif is "married" to Vicky Kaushal"  
  
# This statement throw syntax error  
# ("") not allowed inside string  
  
# To use this we need to use Escape Sequence  
  
str ="Katrina Kaif is \"married\" to Vicky Kaushal"
```



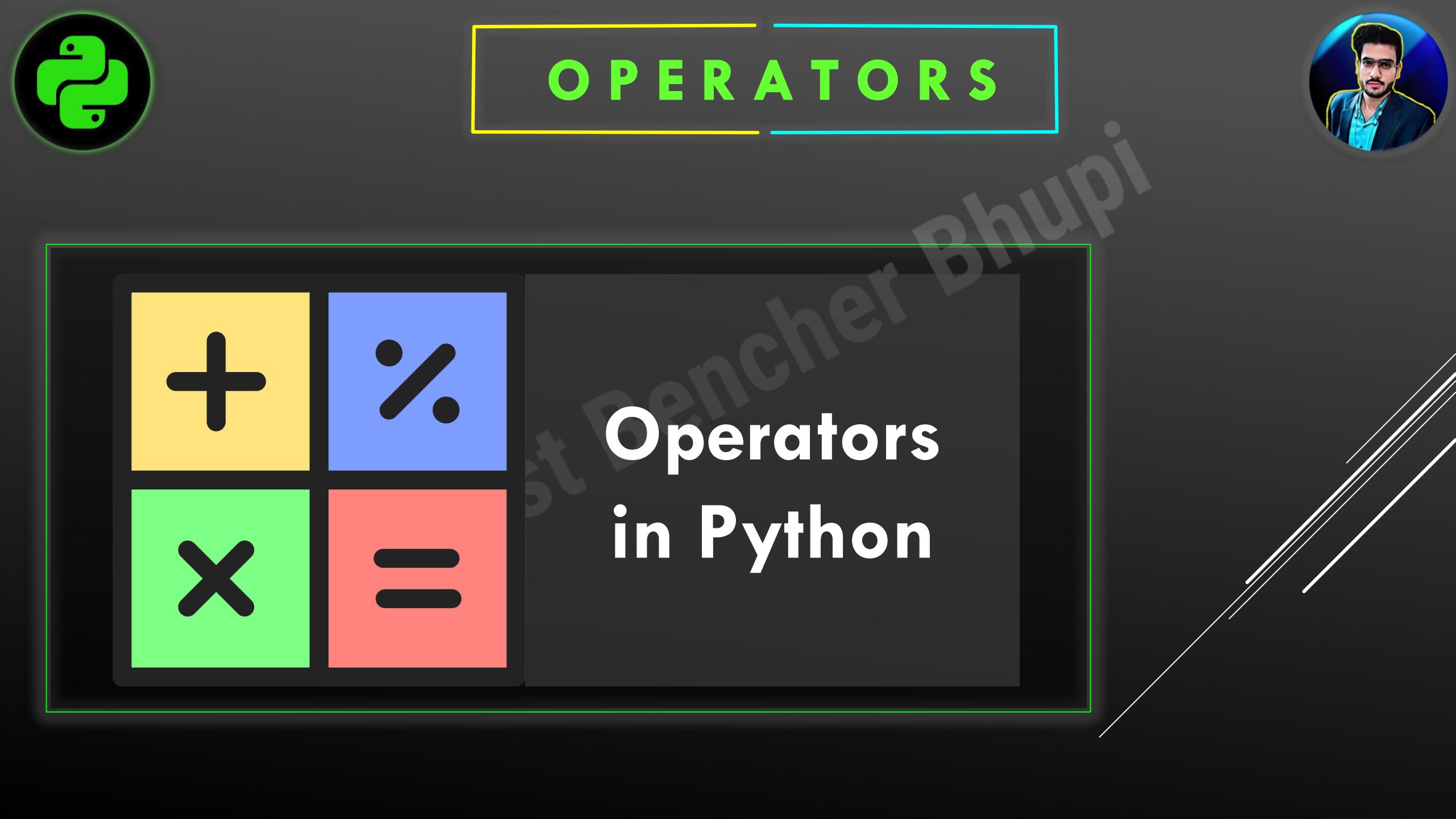
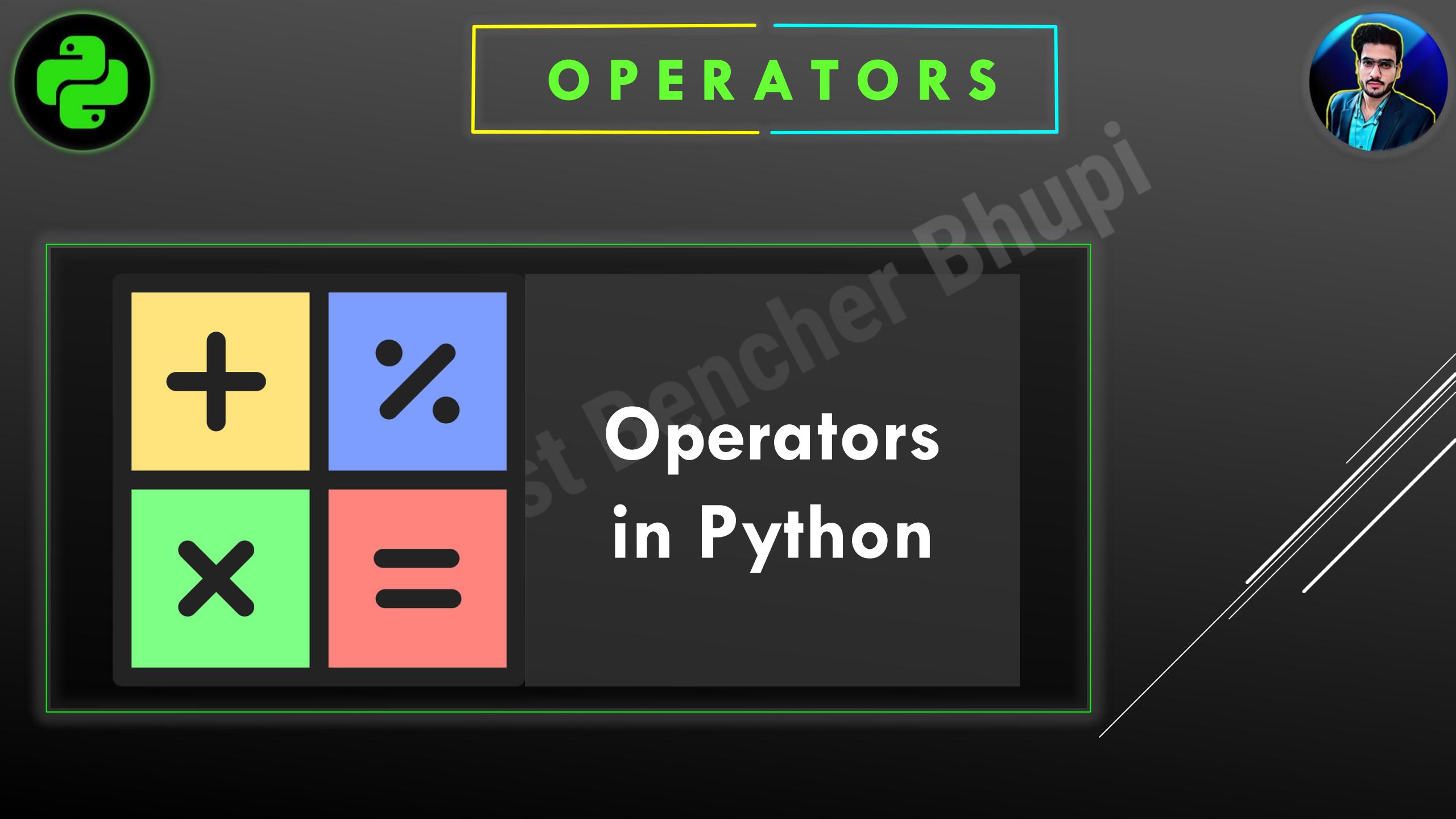


ESCAPE SEQUENCE

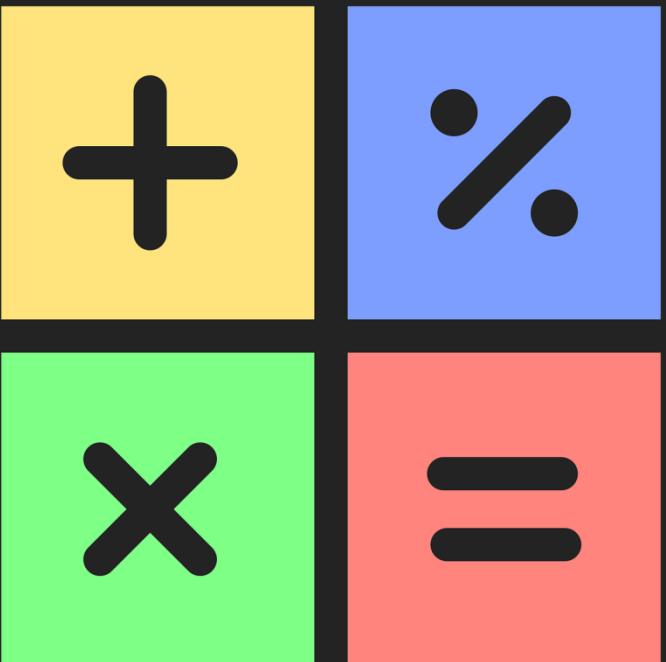


Other escape characters used in Python

- \' Single Quote
- \\" Backslash
- \n New Line
- \t Tab
- \b Backspace



OPERATORS



Operators in Python

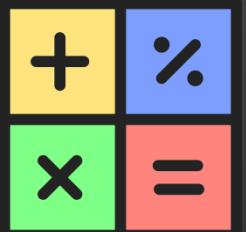


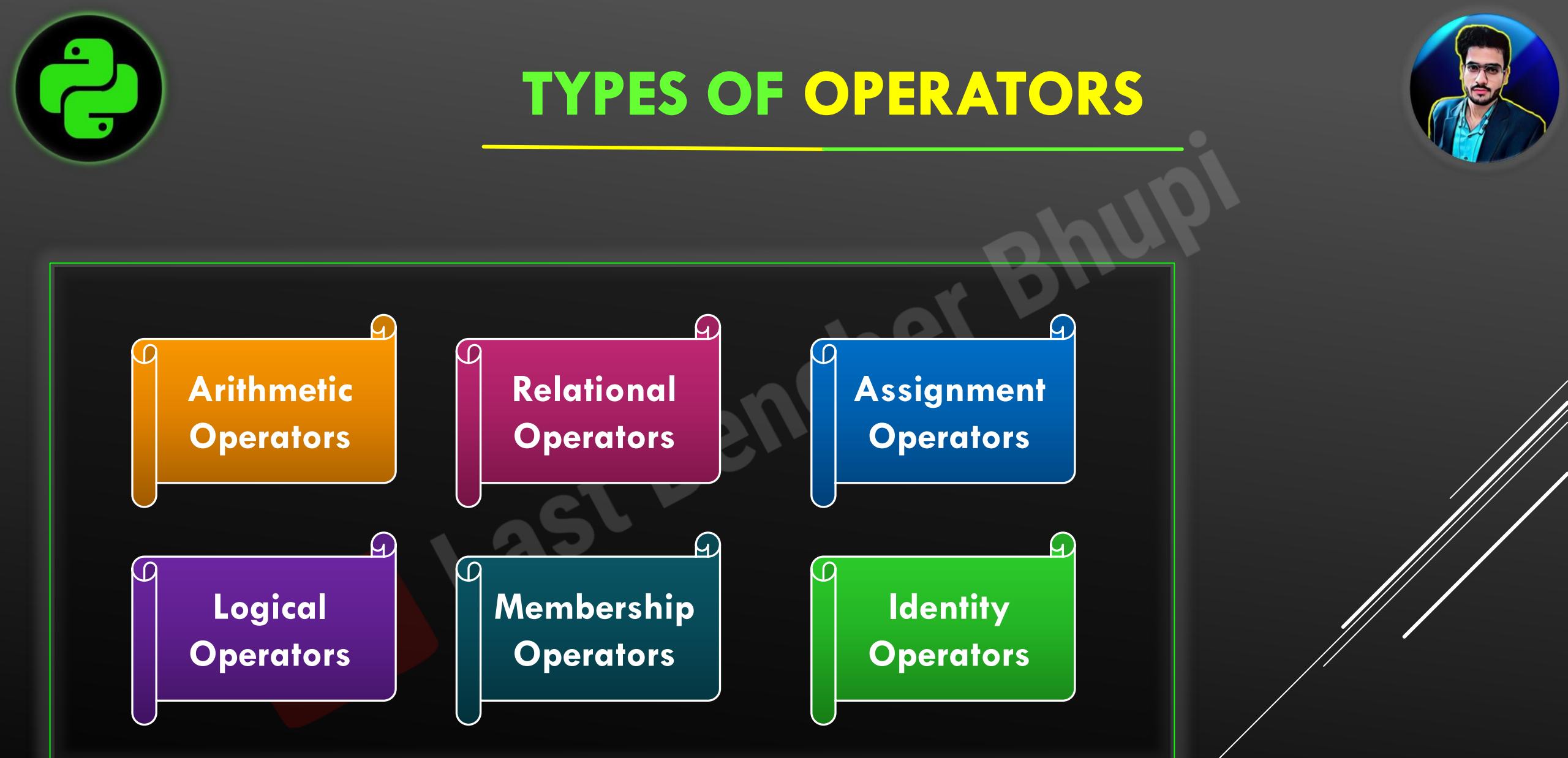
OPERATORS



Python operator is a symbol that performs an operation on one or more operands. An operand is a variable or a value on which we perform the operation.

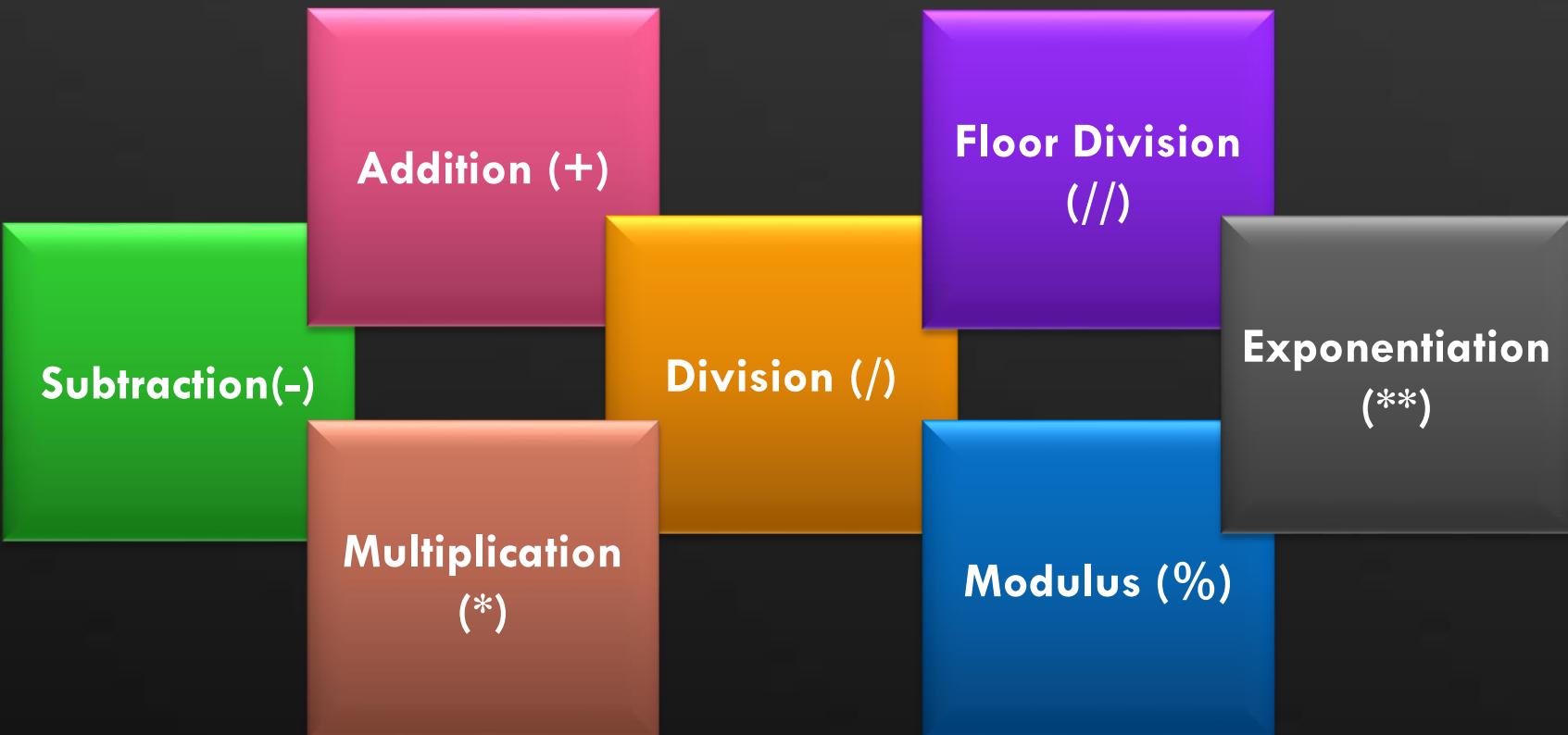
Operator is a symbol that performs certain operations.
Python provides the different types of operators







1 ARITHMETIC OPERATORS





ARITHMETIC OPERATORS



Operator	Meaning	Example	Result
+	Addition	<code>C = 12 + 1</code>	<code>C = 13</code>
-	Subtraction	<code>C = 12 - 1</code>	<code>C = 11</code>
*	Multiplication	<code>C = 12 * 1</code>	<code>C = 12</code>
/	Division	<code>C = 12 / 1</code>	<code>C = 12</code>
//	Floor Division	<code>C = 12 // 10</code>	<code>C = 1</code>
%	Modulus	<code>C = 12 % 10</code>	<code>C = 2</code>
**	Exponentiation	<code>C = 10 ** 2</code>	<code>C = 100</code>



ARITHMETIC OPERATORS



```
print("Arithmetic Operator")
a = 10
b = 5
print("Addition = ",a+b)
print("Subtraction = ",a-b)
print("Multiplication = ",a*b)
print("Division = ",a/b)
print("Floor Division = ",a//b)
print("Modulus = ",a%b)
print("Exponent = ",a**b)
```

Arithmetic Operator

Addition = 15
Subtraction = 5
Multiplication = 50
Division = 2.0
Floor Division = 2
Modulus = 0
Exponent = 100000



ARITHMETIC OPERATORS



- 1 / operator always performs **floating point** arithmetic. Hence it will always returns float value.
- 2 But Floor division (//) can perform both **floating point** and **integral arithmetic**. If arguments are int type then result is int type. If atleast one argument is float type then result is float type.
- 3 We can use +,* operators for str type also.
- 4 If we want to use + operator for str type then compulsory both arguments should be **str type** only otherwise we will get error.



ARITHMETIC OPERATORS



What is the Output

```
print("Alia"+10)
```

- a Alia10
- b Type Error
- c Value Error
- d Syntax Error





ARITHMETIC OPERATORS



What is the Output

```
print("Alia"*2)
```

- a AliaAlia 😊
- b Alia Alia 😕
- c Alia * 2 😕
- d Type Error 😕





2 RELATIONAL OPERATORS



Less than (<)

Equal to (==)

Greater than(>)

Greater than
or equal to
(>=)

Less than or
equal to (<=)

Not equal to
(!=)



RELATIONAL OPERATORS



Relational operators are also called as Comparison operators
It is used to compare values
It either returns True or False according to condition.

Operator	Meaning	Example	Result
>	Greater than	$5 > 6$	False
<	Less than	$5 < 6$	True
==	Equal to	$5 == 6$	False
!=	Not equal to	$5 != 6$	True
>=	Greater than or equal to	$5 >= 6$	False
<=	Less than or equal to	$5 <= 6$	True



RELATIONAL OPERATORS



```
print("Relational Operator")  
a = 10  
b = 5  
  
print(a>b)      == >  
print(a<b)      == >  
print(a==b)      == >  
print(a!=b)      == >  
print(a>=b)      == >  
print(a<=b)      == >
```

Relational Operator
True
False
False
True
True
False



RELATIONAL OPERATORS



What is the Output

```
print(10=="10")
```

- a True 😞
- b False 😊
- c Syntax Error 😞
- d Type Error 😞





RELATIONAL OPERATORS



What is the Output

```
print(10!=10)
```

- a True 😞
- b False 😊
- c Syntax Error 😞
- d Type Error 😞





3 ASSIGNMENT OPERATORS

Assign (`=`)

Add and Assign (`+=`)

Subtract and Assign (`-=`)

Divide and Assign (`/=`)

Multiply and Assign (`*=`)

Modulus and Assign (`%=`)

Exponent and Assign (`=`)**

Floor-Divide and Assign (`//=`)



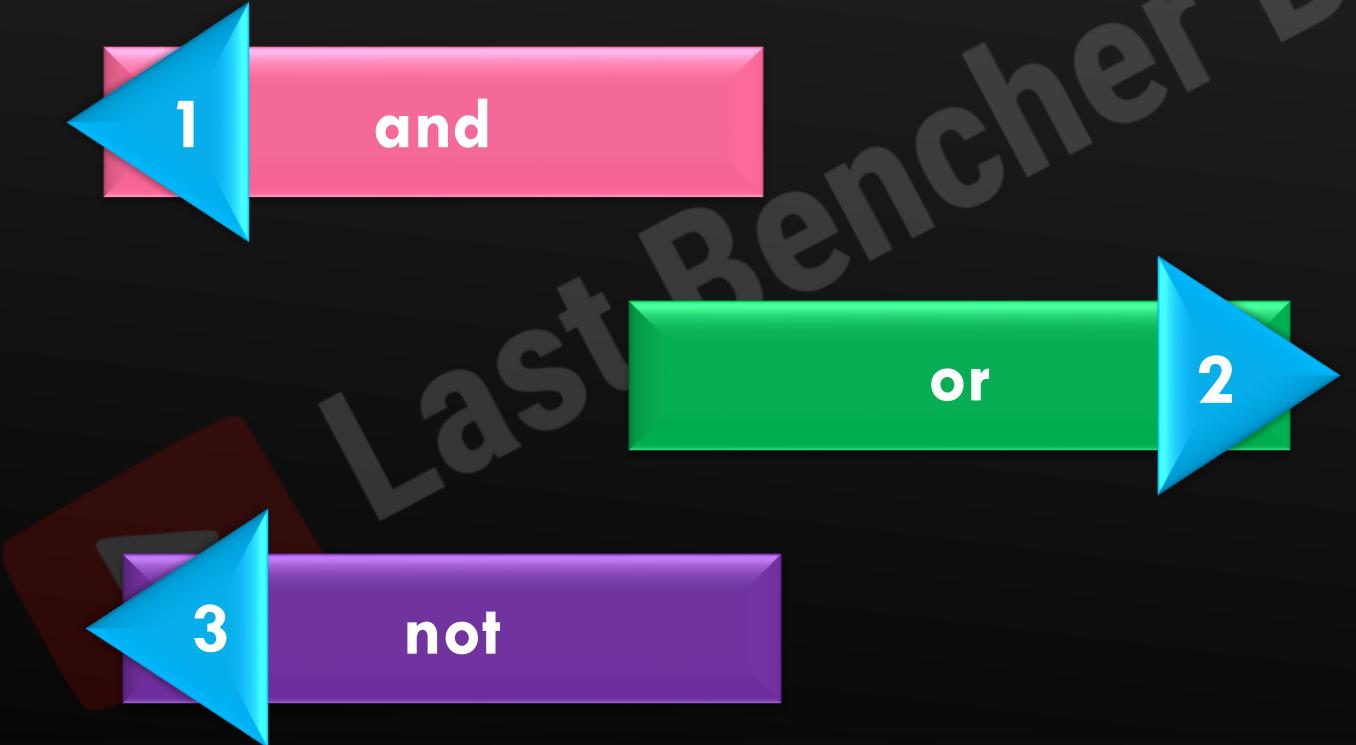
ASSIGNMENT OPERATORS



Operator	Meaning	Example
=	Assign a value	a = 5
+=	Adds and assign the result to the variable	a+=1 (a=a+1)
-=	Subtract and assign the result to the variable	a-=1 (a=a-1)
=	Multiplies and assign the result to the variable	a=5 (a=a*5)
/=	Division and assign the result to the variable	a/=5 (a=a/5)
//=	Floor Division and assign the result to the variable	a//=5 (a=a//5)
%=	Find Modulus and assign the result to the variable	a%=5 (a=a%5)
=	Find Exponentiation and assign the result to the variable	a=5 (a=a**5)



4 LOGICAL OPERATORS





LOGICAL OPERATORS



Logical operator are typically used with **Boolean** (logical) values.
They allow a program to make a decision based on multiple condition

Operator	Meaning	Example	Result
and	True if both the operands are true	$10 < 5 \text{ and } 10 < 20$	False
or	True if either of the operands is true	$10 < 5 \text{ or } 10 < 20$	True
not	True if operands is false	<code>not (10<20)</code>	False



LOGICAL OPERATORS



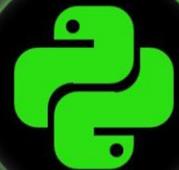
```
print("Logical Operator")  
  
print(10<5 and 10<20)  
print(10<5 or 10<20)  
print(not(10<20))
```

Logical Operator

False

True

False



LOGICAL OPERATORS



```
print(x==5 and y==10)  
print(x==20 and y==20)  
print(x==5 or y==20)  
print(x==20 or y==20)
```

True and True =>
False and False =>
True and False =>
False and False =>

True
False
True
False



LOGICAL OPERATORS



What is the Output Suppose $x = 5$

```
print(not x==20)
```

- a True 😊
- b False 😢
- c 0 😢
- d 1 😢





LOGICAL OPERATORS



```
x = 5,  y = 10, z = 15  
print(x==10 or y==15 or z==5)
```

a True ☹
b False ☺
c 0 ☹
d 1 ☹





5 MEMBERSHIP OPERATORS



Operator	Meaning	Example
In	True if value / variable found in the sequence	5 in x
Not In	True if value / variable is not found in the sequence	5 not in x

IN

NOT
IN



MEMBERSHIP OPERATORS



In – membership operator

- *This checks if a value is a member of a sequence.*
- *In our next slide example, we see that the string ‘katrina’ does not belong to the list cricketers. But the string ‘virat kohli’ belongs to it, so it returns true.*
- *Also, the string ‘ali’ is a substring to the string ‘alia bhatt’. Therefore, it returns true.*

IN



MEMBERSHIP OPERATORS



```
cricketers = ["Rohit", "KL Rahul", "Virat Kohli"]

print("Katrina" in cricketers)
# Returns false
```

```
str = "Alia Bhatt"

print("Alia" in str)
# Returns True
```

IN



MEMBERSHIP OPERATORS



Not In – Membership Operator

- This checks if a value is *not* a member of a sequence.
- In our next slide example, we see that the string ‘Katrina’ does not belong to the list Cricketers. But the string ‘Virat Kohli’ belongs to it, so it returns False.
- Also, the string ‘Ali’ is a substring to the string ‘Alia Bhatt’. Therefore, it returns False.

NOT IN



MEMBERSHIP OPERATORS



```
cricketers = ["Rohit", "KL Rahul", "Virat Kohli"]

print("Katrina" not in cricketers)
# Returns True
```

NOT IN

```
str = "Alia Bhatt"

print("Alia" not in str)
# Returns False
```



MEMBERSHIP OPERATORS



```
list = ["sunny","Binny","Pinni","Chinni"]
```

```
print("Sunny" not in list)
```

- a True 😊
- b False 😞
- c 0 😞
- d 1 😞





MEMBERSHIP OPERATORS



```
str = "HELLO WORLD"  
print("HELLO" not in str)
```

- a True 😞
- b False 😊
- c 0 😞
- d 1 😞

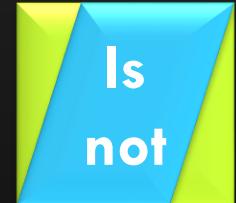




6 IDENTITY OPERATORS



Operator	Meaning	Example
Is	True if the operands are identical	X is true
Is not	True if the operands are not identical	X is not true





IDENTITY OPERATORS



Let us proceed towards identity Python Operator.

These operators test if the two operands share an identity. We have two identity operators- ‘is’ and ‘is not’.

*If two operands have the **same identity**, it returns **True**. Otherwise, it returns **False**.*



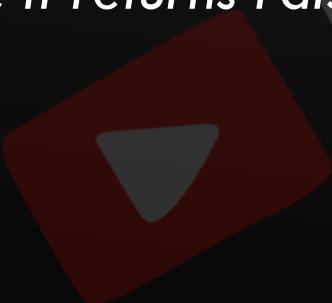


IDENTITY OPERATORS



'is not'

If two operands **Doesn't have the same identity**, it returns True.
Otherwise, it returns False.





IDENTITY OPERATORS



```
a1 = 5
b1 = 5

a2 = "Hello World"
b2 = "Hello World"

a3 = [1,2,3]
b3 = [1,2,3]

print(a1 is not b1)      False
print(a2 is b2)          True
print(a2 is b3)          False
```



IDENTITY OPERATORS



```
a = 10 , b = 10
```

```
print(a is b)
```

- | | | |
|----------|--------------|--|
| a | True | |
| b | False | |
| c | 0 | |
| d | 1 | |





IDENTITY OPERATORS



```
str1 = "Katrina", str2 = "katrina"  
print(str1 is not str2)
```

a True 😊

b False 😞

c Value Error 😞

d Syntax Error 😞

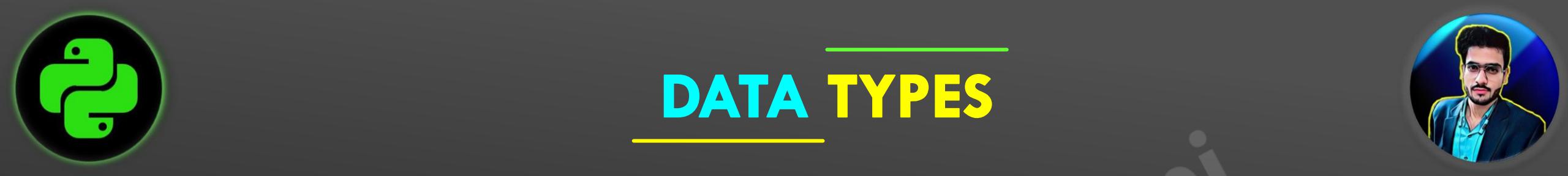




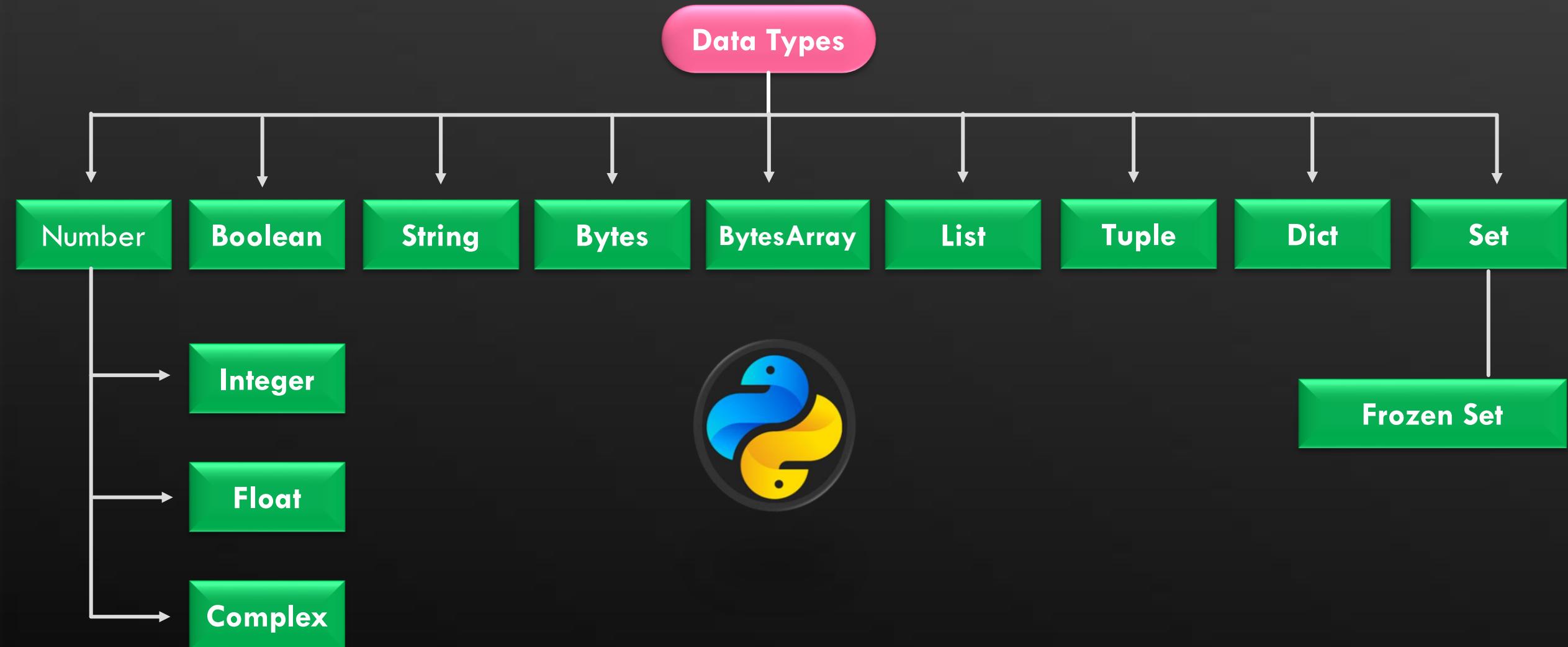
Programming Lab - 01



1. WAP to Add 2 Numbers.
2. WAP to Add 3 Numbers.
3. WAP to Sub 2 Numbers.
4. WAP to Multiply Two Numbers.
5. WAP to Divide Two Numbers.
6. WAP to Print Multiplication of a Table.



DATA TYPES





INTEGERS



We can use *int* data type to **represent whole numbers** (integral values)

Eg: `a=10`

```
type(a) #int
```

In Python 3, there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has, as are all things, but beyond that an integer can be as long as you need it to be:

Python

```
>>> print(123123123123123123123123123123123123123123123 + 1)  
123123123123123123123123123123123123123123123123123123123123123124
```



INT



FLOAT



We can use *float* data type to represent floating point values (decimal values)

Eg: **f=1.234**

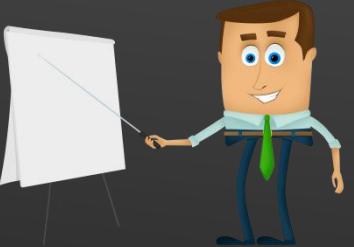
`type(f) float`

We can also represent floating point values by using exponential form (scientific notation)

Eg: **f=1.2e3**

`print(f) 1200.0`

The main advantage of exponential form is we can represent big values in less memory.





C O M P L E X



A complex number is of the form

$$a + bj$$

$\begin{matrix} \xrightarrow{\text{Real Part}} & \xrightarrow{\text{Imaginary Part}} \\ j^2 = -1 \\ j = \sqrt{-1} \end{matrix}$

We can use complex type generally in scientific Applications and electrical engineering Applications.



BOOLEAN



We can use *this data type* to represent boolean values.

The *only allowed values* for *this data type* are:

True and **False**

Internally Python represents True as 1 and False as 0

`b=True`

`type(b) =>bool`





STRINGS



Strings in
Python



STRING



Many of us who are familiar with programming languages like C, C++ etc. will have an answer like “String is a **collection or array of characters.**”

Well in Python also, we say the same definition for String data type. String is array of sequenced characters and is written inside single quotes, double quotes or triple quotes. Also, Python doesn't have character data type, so when we write 'a', it is taken as a string with length 1.



STRING



str represents String data type.

A String is a sequence of characters enclosed within single quotes or double quotes.

```
s1= 'World'    char = 'c'  
s1= "World"   print(type(char)) --> str
```

In Python , we can represent char values also by using str type and explicitly char type is not available.



STRING



By using single quotes or double quotes we cannot represent multi line string literals.

```
s1 = " World  
      Class "
```

For this requirement we should go for triple single quotes('') or triple double quotes(""")

```
s1 = ''' World      s1 = """ World  
      |   | Class '''    |   | Class """
```

We can also use triple quotes to use single quote or double quote in our String.



SLICING



slice means a piece [] operator is called slice operator , which can be used to retrieve parts of String.

The index can be either +ve or -ve.

+ve index means forward direction from Left to Right

-ve index means backward direction from Right to Left

It allows programmers to extract information from a string of data.

The slicing is not just restricted to strings but can be applied to tuples and lists as well.



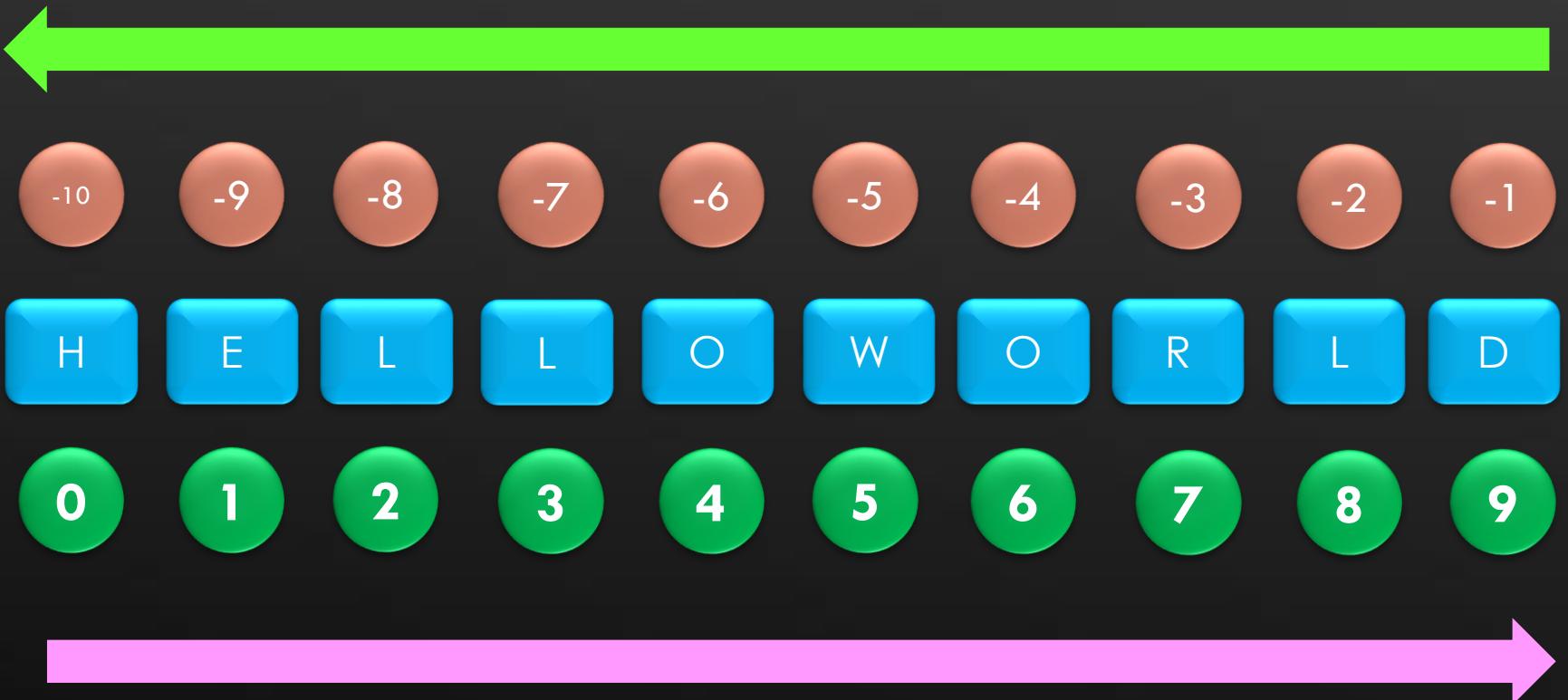


STRIP





SLICING





SLICING SYNTAX



How to get substring in Python?

Str [start : stop : step]

String
Variable

Start index
of substring
(included)

Stop index
of substring
(excluded)

Step Size
Default 1

</>



SLICING



Analyze the String

(-) index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
	H	E	L	L	O		W	O	R	L	D
(+) index	0	1	2	3	4	5	6	7	8	9	10





PREDICT THE OUTPUT



```
print(string[0])  
print(string[1])  
print(string[-1])  
print(string[-11])  
print(string[5])  
print(string[40])
```

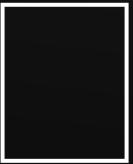




PREDICT THE OUTPUT



```
print(string[:])  
print(string[4:])  
print(string[:4])  
print(string[-5:])  
print(string[-9:])  
print(string[1:-1])  
print(string[1:40])  
print(string[40:40])
```





PREDICT THE OUTPUT



`print(string[::-2])`

`print(string[4::1])`

`print(string[:4:2])`

`print(string[-5::-1])`

`print(string[-9::5])`

`print(string[1:8:3])`

`print(string[1::2])`

`print(string[::-1])`



Mathmatical Operators

S T R I N G S



We can apply the following mathematical operators for Strings.



1. **+ operator for concatenation**
2. *** operator for repetition**



```
print("Vicky"+“Katrina”) #VickyKatrina
```

```
print("Kat" * 3) #KatKatKat
```



Mathmatical Operators

S T R I N G S



Note:

1. To use + operator for Strings, compulsory both arguments should be **str type**
2. To use * operator for Strings, compulsory one argument should be **str** and other argument should be **int**



IMPORTANT



Mathmatical Operators

S T R I N G S



What is the Output

"Katrina" + "3"

- a KatrinaKatrinaKatrina 😞
- b Type Error 😞
- c Katrina3 😊
- d Syntax Error 😞





Mathmatical Operators

S T R I N G S



What is the Output

"Katrina" * "3"

- a KatrinaKatrinaKatrina 😞
- b Type Error 😊
- c Katrina3 😞
- d Syntax Error 😞





PYTHON FORMATTING

STRING

How to format
a String in
Python

`%`

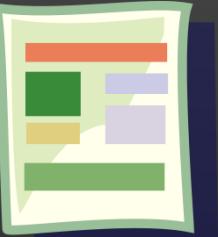
OLD STYLE

`{}`

.FORMAT

`f'`

F STRING





PYTHON STRING FORMATTING



Formatting a string is something you'll do all the time when you're coding in Python.

Sometimes, you may want to print variables along with a string. You can either use commas, or use string formatters for the same.

Name = "Alia"

str = "Hello", Name, "How are you."

**How to format
a String in
Python**



PYTHON STRING FORMATTING



%

OLD STYLE

1. Format a String the Old Way: `print 'Hello %s' % name`

This approach was more often used in Python2, when the language was still young and evolving. It's a technique that is easy to understand for veteran coders who come from a C programming background.

`%d` – for integers

`%s` – for strings

`%f` – for floating-point numbers





PYTHON STRING FORMATTING



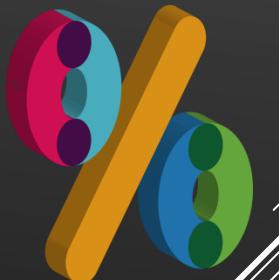
Example to Understand

```
# Use single variable
Name = "Alia"
print("Hello %s How are you " % Name)

# Use Multiple variable
Age = 26
print("Myself %s My Age is %d " % (Name,Age))
```

%

OLD STYLE





PYTHON STRING FORMATTING



2. Format a String using Format: “.format()”

This is a technique that many Python programmers consider a breath of fresh air, since it makes things easier for you.

It was introduced in early Python3. Essentially, the new syntax removed the ‘%’ symbols and instead provided .format() as a string method.

{ } .FORMAT



PYTHON STRING FORMATTING



2. Format a String using Format: ".format()"

{}

.FORMAT

Python Format()
Function

"{ } ".format(value)



PYTHON STRING FORMATTING



Example to Understand

```
Bride = "Deepika"  
Groom = "Ranvir"  
Date = "14 Nov 2018"
```

```
# Do remember the order  
print("{} is married to {} on {}".format(Bride,Groom,Date))
```

```
#Another way to print order doesnot matter here  
print("{girl} is married to {boy} on {day}".format(day=Date,girl=Bride,boy=Groom))
```



PYTHON STRING FORMATTING



f' F STRING

3. Format a String using F Strings

Also short for format string, f strings are the *most recent technique* that Python3 now supports so they're being adopted rapidly.

F strings addresses the verbosity(more words) of the .format() technique and provide a short way to do the same by adding the letter f as a prefix to the string.

it also happens to be simpler for students to learn, and easier to adopt if you're still new to coding.



PYTHON STRING FORMATTING



f' F STRING

Example to Understand

```
Bride = "Deepika"  
Groom = "Ranvir"  
Date = "14 Nov 2018"
```

```
print(f"{Bride} is married to {Groom} on {Date}")
```



String Functions



*Everything in Python is an **object**. A string is an object too. Python provides us with various methods to call on the string object.*

Let's look at each one of them.



```
33     self.logdups = True
34     self.debug = debug
35     self.logger = logging.getLogger(__name__)
36     if path:
37         self._file = open(os.path.abspath(path), 'w')
38         self._file.seek(0)
39         self._fingerprints = set()
40
41     @classmethod
42     def from_settings(cls, settings):
43         debug = settings.getbool('SUPERLUMINOSITY')
44         return cls(job_dir(settings), debug)
45
46     def request_seen(self, request):
47         fp = self.request_fingerprint(request)
48         if fp in self._fingerprints:
49             return True
50         self._fingerprints.add(fp)
```

String Functions in Python



LEN() – In Built Function



We can use `len()` function to find the number of characters present in the string.

```
# Len() in Python  
  
str = "Katrina"  
print(len(str)) # Gives you 7
```



LEN()



LEN() – In Built Function



```
str = "Alia Bhatt_"
print(len(str))
```

a	8	😢
b	9	😢
c	10	😢
d	11	😊





Counting SubStrings



We can find the number of occurrences of substring present in the given string by using `count()` method.

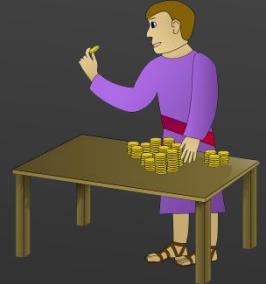
Counting substring

1 `count(substring)`

It will search through out the string

2 `count(substring, begin, end)`

It will search from begin index to end-1 index



COUNT()



Activity Time



```
str = "abbaabbaa"  
print(str.count('a'))
```

a 4

b 5

c 6

d Value Error





Activity Time



```
str = "abbaabbaa"  
print(str.count('a',3,7))
```

- | | | |
|---|---|---|
| a | 1 | 😢 |
| b | 2 | 😊 |
| c | 3 | 😢 |
| d | 4 | 😢 |





Replacing a String with Another String



replace() is an *inbuilt function* in the Python programming language that returns a copy of the string where all occurrences of a substring are replaced with another substring.

String.replace(old,new,count)

Syntax :

```
string.replace(old, new, count)
```

Optional

Python
String
Replace



Replacing a String with Another String



old – old substring you want to replace.

new – new substring which would replace the old substring.

count – the number of times you want to replace the old substring with the new substring. (Optional)

- If count is not specified then all the occurrences of the old substring are replaced with the new substring.
- This method returns the copy of the string i.e. it does not change the original string.

Python String

Replace



Replacing a String with Another String



Example to Understand

```
str="Python Python Python I avoid But Python likes me I can't Avoid"  
print(str.replace("Python","Katrina"))  
print(str.replace("Python","Katrina",3))
```





Splitting of Strings

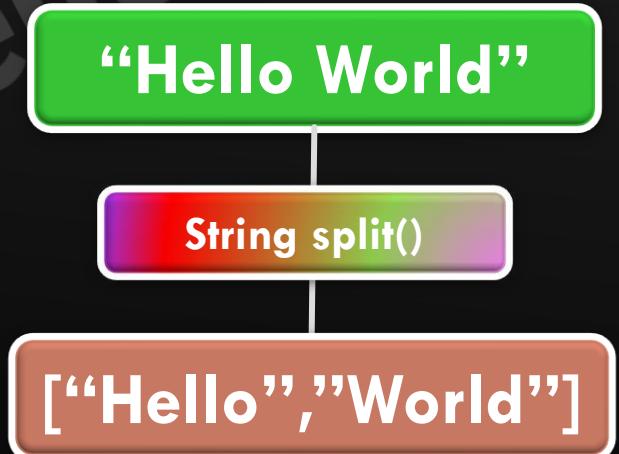


We can *split* the given string according to specified separator by using *split()* method.

```
l=s.split(seperator)
```

The default separator is *space*.

The return type of *split()* method is *List*





Splitting of Strings



String = “I am a string.”



split()



[“I” , “am” , “a” , “string.”]

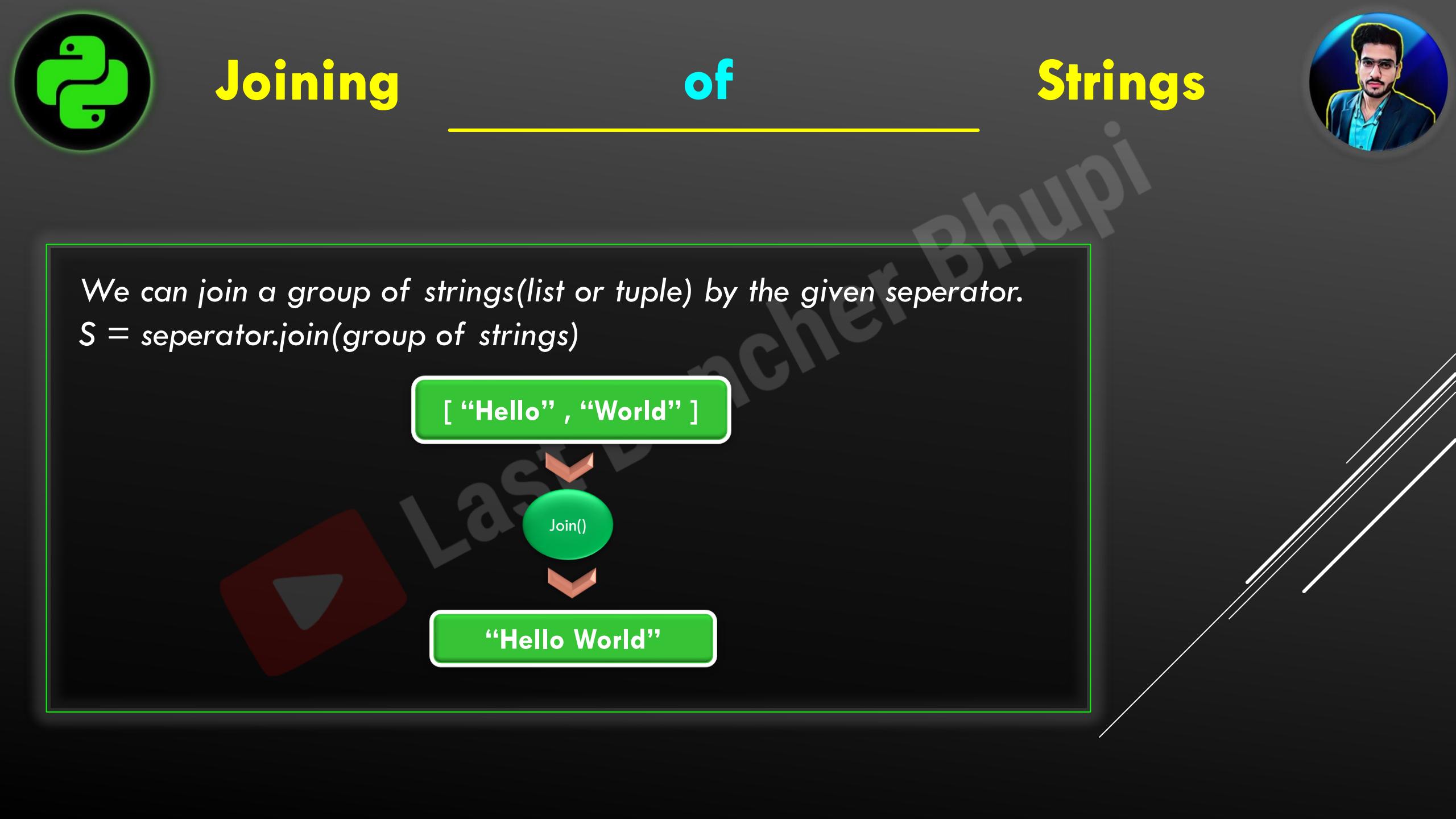


Splitting of Strings



Example to Understand

```
str="Python Python Python I avoid But Python likes me"  
  
# default separator is space  
  
print(str.split())  
  
str="Python,Python,Python,I,avoid,But,Python,likes,me"  
  
# providing (,) separator  
  
print(str.split(","))
```



Joining of Strings



We can join a group of strings(list or tuple) by the given separator.

S = separator.join(group of strings)





Joining of Strings



```
#list
list = ["Chowmien","Roll","Manchurian","Masala Dosa"]

#seperator is space
print(" ".join(list))

#seperator is :
print(":".join(list))
```

```
$python main.py
Chowmien Roll Manchurian Masala Dosa
Chowmien:Roll:Manchurian:Masala Dosa
```



Changing Case of a String



Changing case of a String

1	<code>upper()</code>	To convert all characters to upper case
2	<code>lower()</code>	To convert all characters to lower case
3	<code>swapcase()</code>	converts all lower case characters to upper case and all upper case characters to lower case
4	<code>title()</code>	To convert all character to title case. i.e first character in every word should be upper case and all remaining characters should be in lower case.
5	<code>capitalize()</code>	Only first character will be converted to upper case and all remaining characters can be converted to lower case



Activity Time



```
str = "Katrina and Vicky"
```

```
print(str.swapcase())
```

- a kATRINA And vICKY 😕
- b KATRINA and VICKY 😕
- c kATRINA AND vICKY 😊
- d Katrina AND Vicky 😕





Activity Time



```
str = "Katrina and Vicky"
```

```
print(str.capitalize())
```

- a Katrina And vicky ✘✘
- b Katrina And Vicky ✘✘
- c Katrina and Vicky ✘✘
- d Katrina and vicky 😊





Removing Spaces From String



Removing Spaces from the String

- | | | |
|---|-----------------------|-------------------------------------|
| 1 | <code>rstrip()</code> | To remove spaces at right hand side |
| 2 | <code>lstrip()</code> | To remove spaces at left hand side |
| 3 | <code>strip()</code> | To remove spaces both sides |



Activity Time



```
str = "Rocky Bhai "
```

```
str = str.lstrip()  
print(len(str))
```

- a 12 😞
- b 11 😊
- c 10 😞
- d 13 😞





Checking starting and ending of string



Checking starting and ending part of the string

1 s.startswith(substring) Returns True or False

2 s.endswith(substring) Returns True or False



Activity Time



```
str = "Babu Bhaiya"  
print(str.startswith("babu"))
```

- a **FALSE** 😞
- b **TRUE** 😞
- c **False** 😊
- d **True** 😞





Activity Time



```
str = "Babu Bhaiya"  
str2="Bhaiya"  
print(str.endswith(str2))
```

- a **FALSE**
- b **TRUE**
- c **False**
- d **True**





Check Type of Char Present in a String



To check type of characters present in a string

1	isalnum()	Returns True if all characters are alphanumeric(a to z , A to Z ,0 to 9)
2	isalpha()	Returns True if all characters are only alphabet symbols(a to z,A to Z)
3	isdigit()	Returns True if all characters are digits only(0 to 9)
4	islower()	Returns True if all characters are lower case alphabet symbols
5	isupper()	Returns True if all characters are upper case alphabet symbols
6	istitle()	Returns True if string is in title case
7	isspace()	Returns True if string contains only spaces



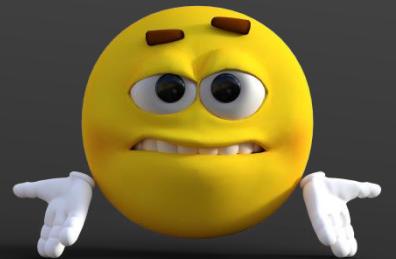
Activity Time



```
str = "Babu Bhaiya@"
```

```
print(str.isalnum())
```

- a FALSE 😞
- b TRUE 😞
- c False 😊
- d True 😞





Activity Time



```
str = "Babu Bhaiya"  
print(str.isalpha())
```

a FALSE 😞

b TRUE 😞

c False 😊

d True 😞





Finding Substrings



Finding Substrings

1	<code>find()</code>	Returns index of first occurrence of the given substring. If it is not available then we will get -1
2	<code>index()</code>	<code>index()</code> method is exactly same as <code>find()</code> method except that if the specified substring is not available then we will get <code>ValueError</code> .
3	<code>rfind()</code>	Same as <code>find()</code> except it find the occurrence of substring from right side
4	<code>rindex()</code>	same as <code>index()</code> except it find the occurrence of substring from right side



Activity Time



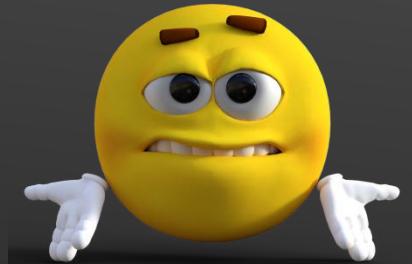
```
str = "The Amazing Spiderman"  
print(str.find("Amaz"))
```

a 5

b -1

c 4

d Value Error





Activity Time



```
str = "The Amazing Spiderman"  
print(str.find("spiderman"))
```

a 11

b -1

c 12

d Value Error





Activity Time



```
str = "The Amazing Spiderman"  
print(str.index("spiderman"))
```

a 11

b -1

c 12

d Value Error





- In Python the following 5 data types are considered as Fundamental
- In Python , we can represent char values also by using str type and explicitly char type is not available.
- Long Data Type is available in Python2 but not in Python3.



- 1 int
- 2 float
- 3 complex
- 4 bool
- 5 str



ACTIVITY TIME



Practice ! Practice ! Practice

Practice as much as you can as Strings are the most important topic in any programming language and also for the Interview Room

We can do String Related Programming Questions Later when we are familiar with programming in python (Control Flow i.e. loops if statements etc)





INDENTATION



*Indentation refers to the **whitespaces** (usually 4 spaces or a single tab) that signify the beginning of a suite (block) of code. All statements indented at the same level belong to the same suite.*

In other languages, a block of code is written inside curly braces ({ }).



The screenshot shows a Python code editor with the following code:

```
n = 10
if n>5:
    print "n is greater than 5"
```

An error message is displayed:

File "main.py", line 3
Print "n is greater than 5"
IndentationError: expected an indented block

A large green arrow points from the explanatory text above to this screenshot.



INDENTATION



Python has made the syntax a bit simpler by removing these curly braces from the block of your code. You do not need to type anything to mark the beginning and the end of a block.

*Instead in Python, we use indentation along with **colon (:)** for that purpose. The colon (:) introduces a new suite(block) of code*

The screenshot shows a Python code editor window titled "Indentation". The code in the editor is:

```
n = 10
if n>5:
    print "n is greater than 5"
```

A tooltip or status bar at the bottom right of the editor window displays the error message: "File \"main.py\", line 3, in <module>
 Print \"n is greater than 5\""
IndentationError: expected an indented block".

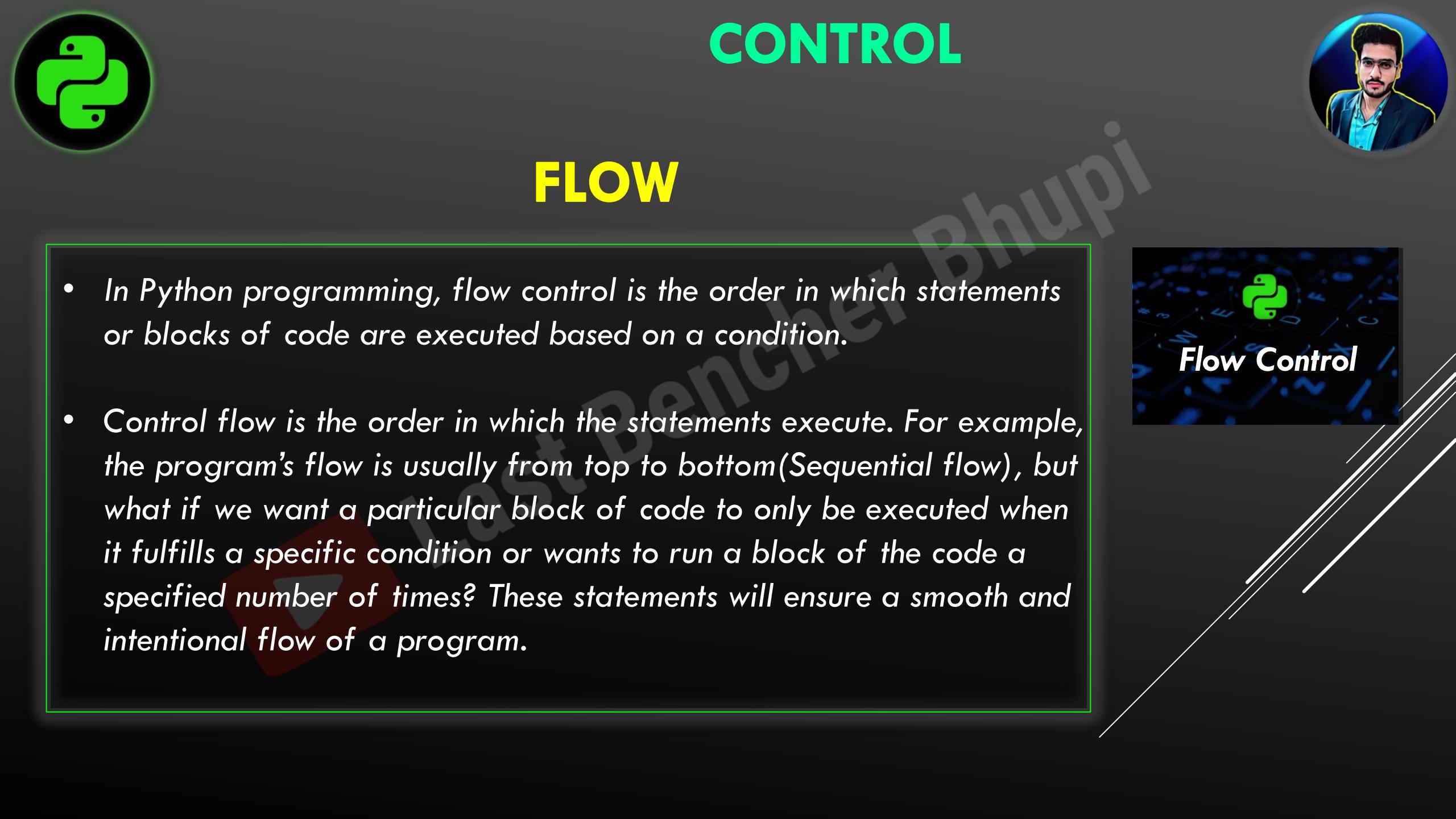


Example of INDENTATION



Example to Understand

```
1  
2 age = 14  
3  
4 if age>=18:  
5     print("You are eligible to vote")  
6 else:  
7     print("You are Not eligible to vote")  
8
```

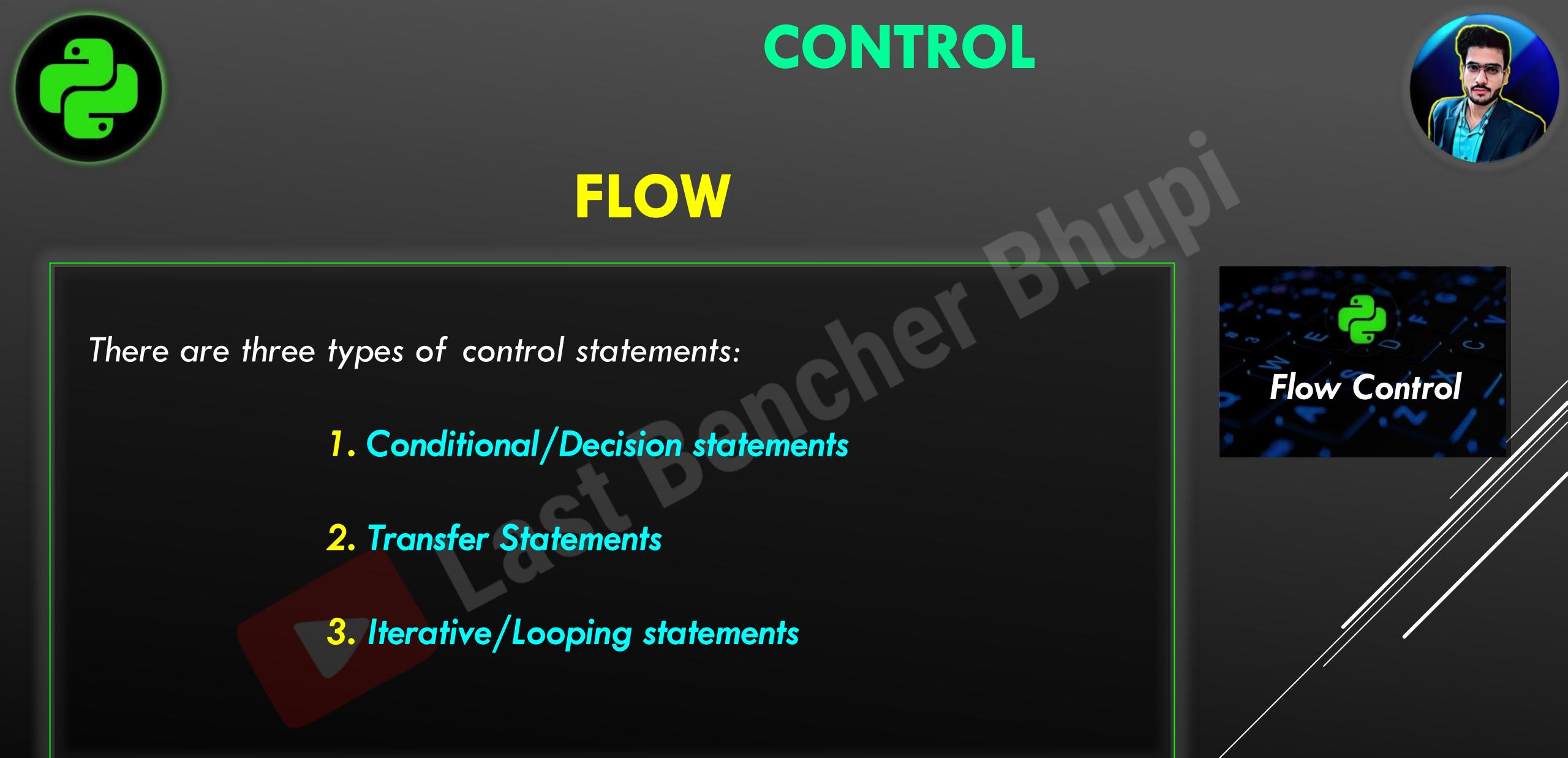


CONTROL

FLOW

- In Python programming, *flow control* is the order in which statements or blocks of code are executed based on a condition.
- Control flow is the order in which the statements execute. For example, the program's flow is usually from top to bottom(Sequential flow), but what if we want a particular block of code to only be executed when it fulfills a specific condition or wants to run a block of the code a specified number of times? These statements will ensure a smooth and intentional flow of a program.





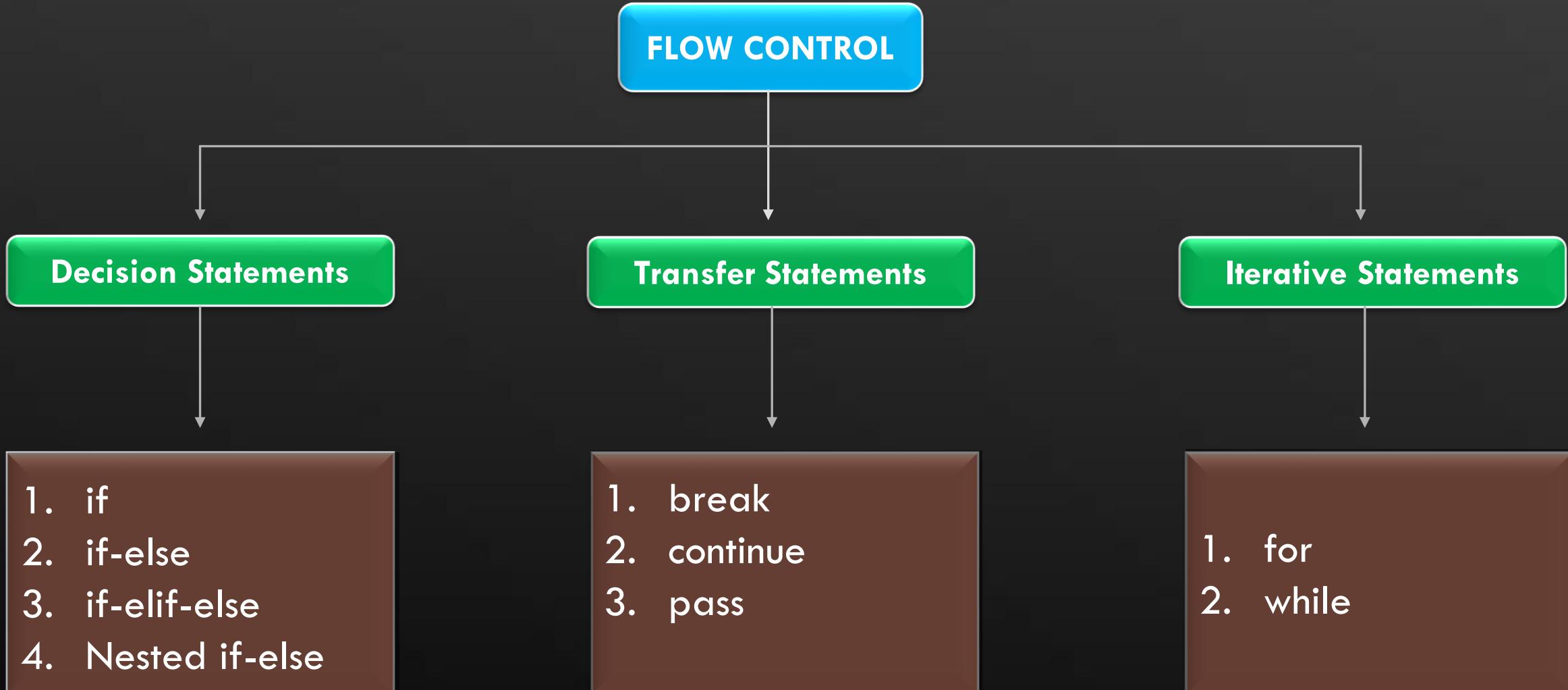
There are three types of control statements:

- 1. Conditional/Decision statements**
- 2. Transfer Statements**
- 3. Iterative/Looping statements**





FLOW CONTROL





1 DECISION MAKING STATEMENTS



In Python, condition statements act depending on whether a given condition is true or false. You can execute different blocks of codes depending on the outcome of a condition. Condition statements always evaluate to either True or False.





DECISION MAKING STATEMENTS



Sometimes, in a program, we may want to make a decision based on a condition. We know that an expression's value can be True or False. We may want to do something only when a certain condition is true.

For example, assume variables a and b. If a is greater, then we want to print “a is greater”. Otherwise, we want to print “b is greater”.

For this, we use an **if-statement**





DECISION MAKING STATEMENTS



01

If
statement

02

If else
statement

03

If - elif -
else
statement

04

Nested if
statement



IF STATEMENT



if statement is the most simple form of decision-making statement. It takes an expression and checks if the expression evaluates to True then the block of code in if statement will be executed.

If the expression evaluates to False, then the block of code is skipped.





IF STATEMENT



SYNTAX



```
if (expr):  
    statement 1  
    statement 2  
    statement 3  
    rest of the code
```

True

False



ACTIVITY TIME



```
1 num = 15
2
3 if num>=10:
4     print("Number is Smaller than 10")
5
6 print("Hello")
```





ACTIVITY TIME



```
1 num = 5  
2  
3 if num>=10:  
4     print("Number is Smaller than 10")  
5  
6 print("Hello")
```

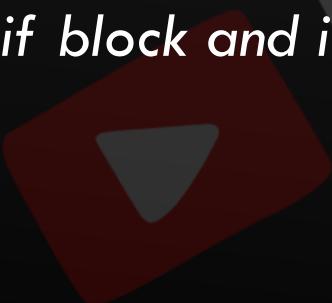




IF ELSE STATEMENT



From the name itself, we get the clue that the if-else statement checks the expression and executes the if block when the expression is True otherwise it will execute the else block of code. The else block should be right after if block and it is executed when the expression is False.





IF ELSE STATEMENT



SYNTAX



```
if (expr):  
    statement  
    statement  
else:  
    statement  
    statement
```

True

False

rest of the code





ACTIVITY TIME



```
if (1):
    print("You are in if block")
else:
    print("you are in else block")
```





ACTIVITY TIME



WAP to find Voting eligibility of a person.

```
age = 18

if age>=18:
    print("You are eligible to vote")
else:
    print("You are not eligible to vote")
```



CONCLUSION



If you are thinking that the output is you are eligible to vote then python interpreter to you be like





CONCLUSION



You know why because of **indentation error** remember we have just cover the small topic but.,

It will give you errors which you can't able to find especially if you are beginner

Result



```
$python main.py
  File "main.py", line 5
    print("You are eligible to vote")
          ^
IndentationError: expected an indented block
```



IF ELSE STATEMENT



WAP to find Voting eligibility of a person.

```
1  
2 age = 14  
3  
4 if age>=18:  
5     print("You are eligible to vote")  
6 else:  
7     print("You are Not eligible to vote")  
8
```



IF ELSE STATEMENT



```
1 name = input("Enter your name : ")
2
3
4 if name=='Katrina':
5     print("Hello Katrina! How are you ")
6 else:
7     print("Aeeeeyy! Who are you man ")
8
```



Programming Lab - 02



7. Write a program to find voting eligibility by user input
8. Write a program to find maximum between two numbers.
9. WAP to find whether a number is even or odd





IF - ELIF - ELSE STATEMENT



if statement is the most simple form of decision-making statement. It takes an expression and checks if the expression evaluates to True then the block of code in if statement will be executed.

If the expression evaluates to False, then the block of code is skipped.





IF - ELIF - ELSE STATEMENT



SYNTAX



```
if condition:  
    statement 1  
    statement 2  
    statement 3  
elif condition:  
    statement 4  
    statement 5  
else:  
    statement 6
```

Rest of the statements here





ACTIVITY TIME



```
if (False):
    print("You are in if block")
elif (5):
    print("you are in elif block")
else:
    print("You are in else block")
```





Programming Lab – 03



- 10.** Write a program to find maximum between three numbers.

- 11.** Write a program to check whether a number is negative, positive or zero.





NESTED IF STATEMENT



In Python, the nested if-else statement is an if statement inside another if-else statement. It is allowed in Python to put any number of if statements in another if statement.

Indentation is the only way to differentiate the level of nesting. The nested if-else is useful when we want to make a series of decisions.

04

**Nested if
statement**



NESTED IF STATEMENT



SYNTAX



```
if conditon_outer:  
    if condition_inner:  
        #statement of inner if  
    else:  
        #statement of inner else:  
        #statement ot outer if  
else:  
    Outer else  
#statement outside if block
```

04

Nested if
statement



NESTED IF STATEMENT



```
str = input("Enter Name : ")

if str.startswith("A"):
    print("Yes! it starts with A")

    if str=="Alia Bhatt":
        print("I know You are Alia Bhatt")
    else:
        print("You are not Alia Bhatt")
else:
    print("No! it's not starting with A")
```





Programming Lab - 04



12. WAP to Make Calculator

13. Write a to calculate profit or loss.

14. WAP To Make a Marksheets of a student taking 4 subjects as input and display the total, percent & its result



2 ITERATIVE STATEMENTS



When you want some statements to execute a hundred times, you don't repeat them 100 times.

Think of when you want to print numbers 1 to 99. Or that you want to say Hello to 99 friends.

In such a case, you can use **loops or iterative statements** in python.

Python
Iteration Statements



ITERATIVE STATEMENTS



Loops are one of the most powerful and basic concepts in programming. A loop can contain a set of statements that keeps on executing until a specific condition is reached.

Today, we are going to learn about the loops that are available in Python.



1. For Loop
2. While Loop
3. Nested Loop



ITERATIVE STATEMENTS



LOOPS



FOR
LOOP



WHILE
LOOP



NESTED
LOOP



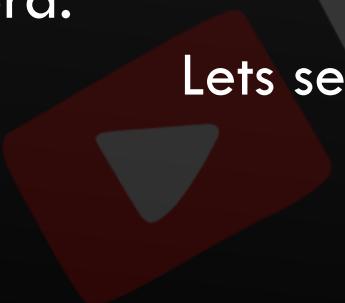
FOR LOOP



Python for loop can iterate over a sequence of items. The structure of a for loop in Python is different than that in C++ or Java.

That is, `for(int i=0;i<n;i++)` won't work here. In Python, we use the '`in`' keyword.

Lets see a Python for loop Example





FOR LOOP



SYNTAX



```
for var in iterable:  
    statement 1  
    statement 2  
    statement 3  
else:  
    statement 4
```



FOR
LOOP



FOR LOOP

1

The range() function
with For Loop

2

Iterating on lists or
similar constructs

3

Iterating on indices of
a list or a similar construct

4

The else statement
for for-loop



FOR LOOP With Range()



When using for loops in Python, the `range()` function is pretty useful to specify the number of times the loop is executed. It yields a sequence of numbers within a specified range.

`range(start, stop, step)`

1. The first argument is the starting value. It is zero by default.
2. The second argument is the ending value of the range.
3. The third argument is the number of steps to take after each yield.

```
list(range(10))  
list(range(4,10))  
list(range(2,10,2))
```



FOR LOOP With Range()



You can use the for loop to iterate over the range of objects.

```
for i in range(20):  
    print(i)  
  
for i in range(1,10):  
    print(i)  
  
for i in range(2,20,2):  
    print(i)
```

FOR
LOOP

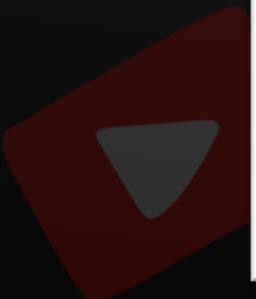


FOR LOOP



Similarly, we can iterate the same way in tuples, lists, sets, and strings.

```
str = "Katrina"  
for char in str:  
    print(char)
```





2

Iterating on lists or similar constructs

You aren't bound to use the `range()` function, though. You can use the `loop` to iterate on a list or a similar construct.

```
# creation of List
list = [10,20,30,40,50]

# iterating every object in List
for i in list:
    print(i)
```





3

Iterating on indices of list or a similar construct

The `len()` function returns the length of the list. When you apply the `range()` function on that, it returns the `indices of the list` on a range object. You can iterate on that.

```
# creation of list
list = ['Katrina', 'Deepika', 'Alia']

# iterating every object in List
for i in range(len(list)):
    print(list[i])
```





4

The else statement for for-loop



A for-loop may also have an else statement after it.

When the loop is exhausted, the block under the else statement executes.

```
for i in range(1,5):
    print(i)
else:
    print("Else Executed")
```



4

The else statement for for-loop



It doesn't execute if you break out of the loop or if an exception is raised. Just run the program and observe the Difference. These basic internal knowledge will prepare you to learn python very easily

```
for i in range(1,5):
    print(i)
    if i==2:
        break
else:
    print("Else Executed")
```



ACTIVITY TIME



Using For Loop

Write a program to print counting from 1 to 10.

Write a program to print counting from 10 to 1 (Reverse order).





FOR LOOP



Which of the following loop is not supported by the python programming language?

- a for loop 
- b while loop 
- c do-while loop 
- d None of These 





FOR LOOP



Observe the Output

```
for num in range(2, -5, -1):  
    print(num)
```

- a 2, 1, 0
- b 2, 1, 0, -1, -2, -3, -4, -5
- c 2, 1, 0, -1, -2, -3, -4
- d Syntax Error



WHILE LOOP



A while loop in python iterates till its condition becomes False. In other words, it executes the statements under itself while the condition it takes is True

When the program control reaches the while loop, the condition is checked. If the condition is true, the block of code under it is executed.

Remember to indent all statements under the loop equally. After that, the condition is checked again.

This continues until the condition becomes false.

WHILE
LOOP



WHILE LOOP



SYNTAX



```
while condition:  
    statement 1  
    statement 2  
    statement 3  
else:  
    statement 4
```





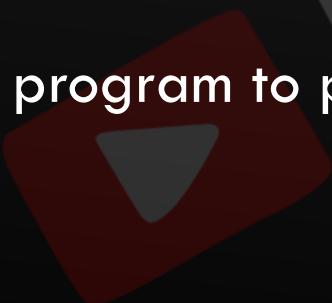
ACTIVITY TIME



Using While Loop

Write a program to print counting from 1 to 10.

Write a program to print counting from 10 to 1 (Reverse order).





WHILE LOOP



1

An Infinite Loop

The else statement for While Loop

2



WHILE AS INFINITE LOOP



Be careful while using a while loop. Because if you forget to increment the counter variable in python, or write logic, the condition may never become false.

In such a case, the loop will **run infinitely**, and the conditions after the loop will starve. To stop execution, press **Ctrl+C**.

However, an infinite loop may actually be useful.



WHILE AS INFINITE LOOP



This is important we have to run the condition explicitly as per our own needs if don't do that there may be a chance of infinite loop

```
a = 1
while (a<=5):
    print(a)
    a+=1 # important to iterate
```



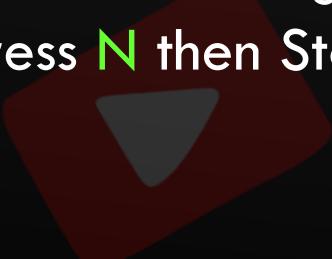
ACTIVITY TIME



Write a Simple Program to Ask **name** from **user** and greet them
With the **message** that you want to add more name

If user press **Y** then again same process

If user press **N** then Stop the execution





ACTIVITY TIME



Observe Carefully

```
while True:  
    name = input("What's your Name : ")  
    print("Hello! Good Morning ",name)  
  
    val = input("Do you want to add More Y/N : ")  
    if val=='N':  
        break  
    else:
```



WHILE LOOP WITH ELSE



A while loop may have an else statement after it. When the condition becomes false, the block under the else statement is executed.

```
a = 1
while (a<=5):
    print(a)
    a+=1 # important to iterate
else:
    print("Else Block Executed")
```

However, it doesn't execute if you break out of the loop or if an exception is raised.



WHILE LOOP WITH ELSE



It doesn't execute if you **break** out of the loop or if an **exception** is raised. Just run the program and observe the Difference. These basic internal knowledge will prepare you to learn python very easily

```
a = 1
while (a<=5):
    print(a)
    if a==2:
        break
    a+=1
else:
    print("Else Block Executed")
```



WHILE LOOP



```
number = 5
while number <= 5:
    if number < 5:
        number = number + 1
    print(number)
```

- a The program will loop indefinitely
- b The value of number will be printed exactly 1 time
- c The while loop will never get executed
- d The value of number will be printed exactly 5 times





WHILE LOOP



```
counter = 1
sum = 0
while counter <= 6:
    sum = sum + counter
    counter = counter + 2
print(sum)
```

- | | | |
|---|----|--|
| a | 12 | |
| b | 9 | |
| c | 7 | |
| d | 8 | |





WHILE LOOP



After Doing Previous 2 Practicals 😂





FOR V/S WHILE



As a Learner, After learning Loops Concept We always have a question which one to prefer firstly, Here is your Answer





FOR V/S WHILE

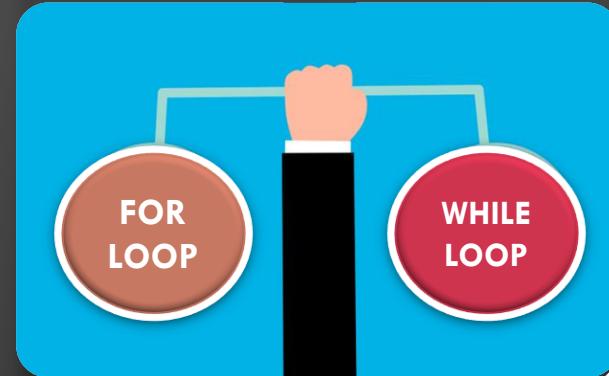


An unknown number of times:

For example, Ask the user to guess the lucky number. You don't know how many attempts the user will need to guess correctly. It can be 1, 20, or maybe indefinite. In such cases, use a while loop.

Fixed number of times:

Print the multiplication table of 2. In this case, you know how many iterations you need. Here you need 10 iterations. In such a case use for loop.





NESTED LOOP



Nested for loop is a for loop inside another for a loop.

A nested loop has one loop inside of another. It is mainly used with two-dimensional arrays. For example, printing numbers or star patterns. Here outer loop is nothing but a row, and the inner loop is columns.





NESTED LOOP



SYNTAX



```
# outer for Loop
for element in sequence

# inner for Loop
body of outer for loop - starts

for element in sequence:
    body of inner for loop
    body of outer for loop - ends

other statements
```



ACTIVITY TIME



Example to Understand

```
# outer for loop
for i in range(1,5):
    # inner for loop
    print("i= ",i)

    for k in range(1,5):
        print("k= ",k,end=" ")

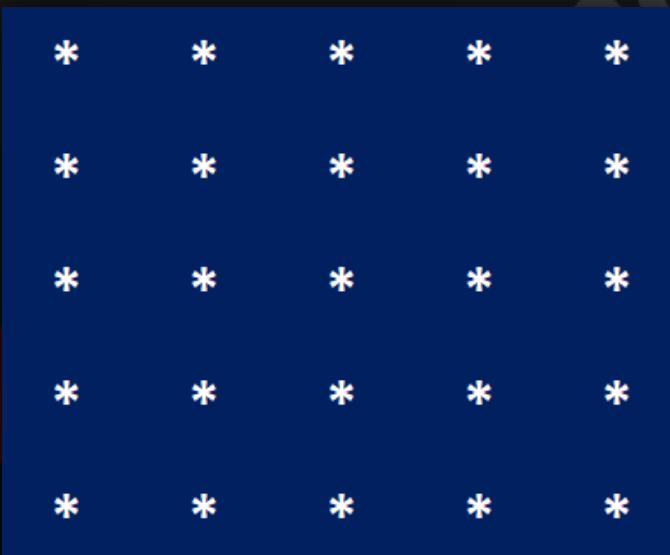
print()
```



ACTIVITY TIME



Write a program to print Pattern like this





NESTED LOOP



All Right We have completed the nested loops it is very important especially in pattern programs so practice as much as you can We will do this nesting concept through out in our programming

Note: If the break statement is used inside a nested loop (loop inside another loop), it will terminate the innermost loop.



IMPORTANT



TRANSFER OR LOOP CONTROL STMTS



Python allows us to control the flow of the execution of the program in a certain manner. For this we use the continue, break and pass keywords.

For this, we have three keywords in Python-



1. **break**
2. **continue**
3. **pass**



TRANSFER OR LOOP CONTROL STMTS

Transfer
Statements

Python

BREAK

CONTINUE

PASS



BREAK



The break statement inside a loop is used to exit out of the loop. Sometimes in a program, we need to exit the loop when a certain condition is fulfilled.

When you put a break statement in the body of a loop, the loop stops executing, and control shifts to the first statement outside it.

You can put it in a for or while loop.





BREAK



It is used to **terminate the loop**, and program control **resumes at the next statement** following the loop.





CONTROL FLOW OF BREAK STATEMENT



```
for i in sequence:  
    statement 1  
    statement 2  
    ...  
    if condition: Terminate the Loop  
        break ← Statement 3 & 4 will  
        statement 3 not execute  
        statement 4  
  
    Next Statement After Loop
```



PROGRAM TO UNDERSTAND BREAK



Run the program and observe

```
for i in range(1,6):
    print(i)
    if i==3:
        break
```



CONTINUE



The `continue` statement **skip the current iteration** and **move to the next iteration**. In Python, when the `continue` statement is encountered inside the loop, it skips all the statements below it and immediately jumps to the next iteration.

In simple words, the `continue` statement is used inside loops. Whenever the `continue` statement is encountered inside a loop, control directly jumps to the start of the loop for the next iteration, skipping the rest of the code present inside the loop's body for the current iteration.





CONTINUE



In some situations, it is helpful to skip executing some statement inside a loop's body if a particular condition occurs and directly move to the next iteration.





CONTORL FLOW OF CONTINUE STMT



```
for i in sequence:  
    statement 1  
    statement 2  
    ...  
    if condition:  
        continue ← Moved to the  
                    next iteration  
        statement 3 ← Skipped for  
                    particular  
        statement 4 ← iteration
```

Next Statement After Loop



PROGRAM TO UNDERSTAND CONTINUE



Run the program and observe

```
for i in range(1,6):
    if i==3:
        continue
    print(i)
```



PASS



The `pass` is the keyword In Python, which won't do anything.

Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the `pass` keyword.

A `pass` statement is a Python null statement. When the interpreter finds a `pass` statement in the program, it returns no operation. Nothing happens when the `pass` statement is executed.





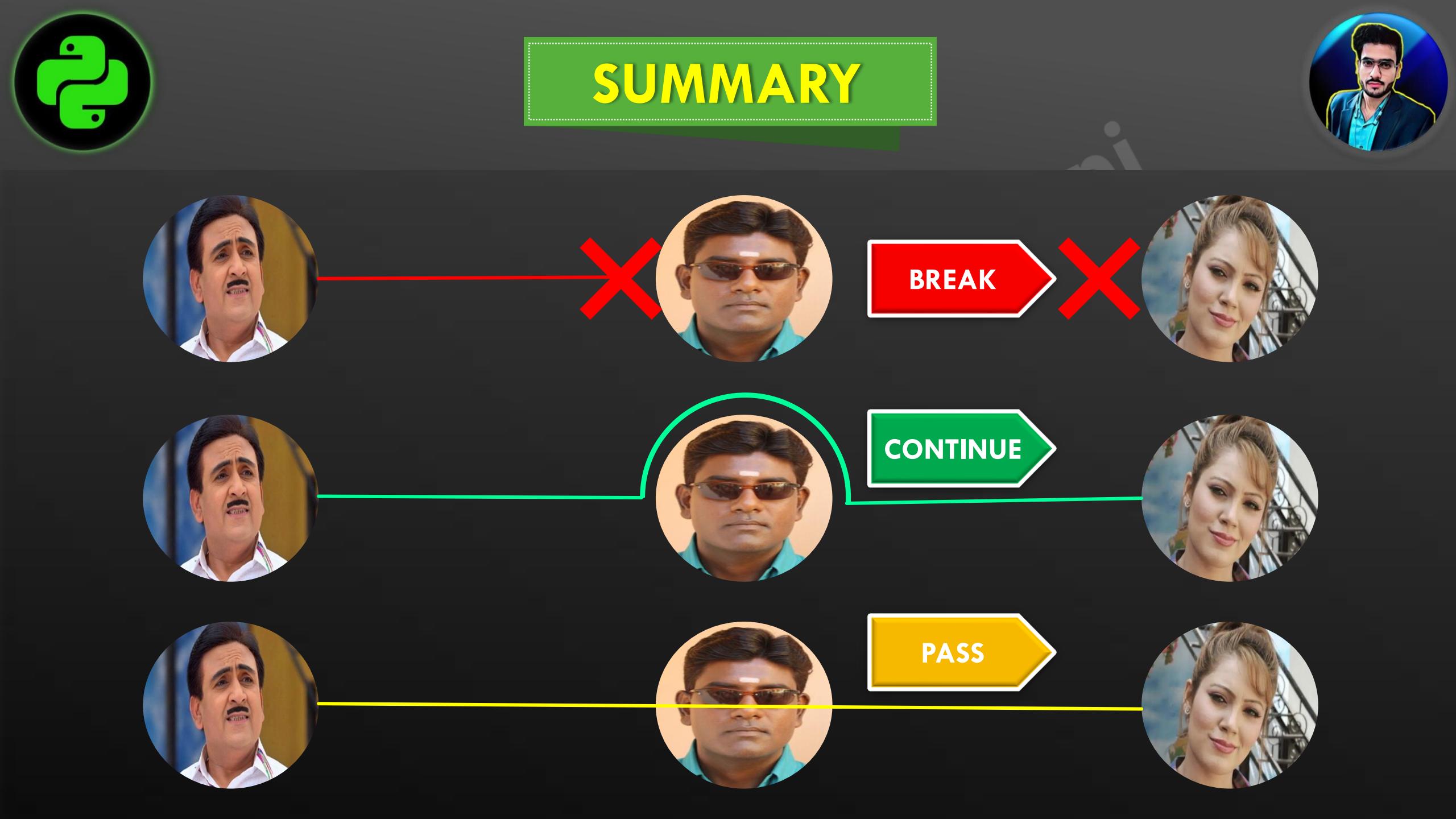
PASS

Sometimes We don't know the full implementation of the program just we need to declare that so pass is used to provide null body and there is no syntax error

```
for i in range(1,6):
    pass

def sum():
    pass
```







SUMMARY

Statement	Description
break	Terminate the current loop . Use the break statement to come out of the loop instantly.
continue	Skips the current iteration of a loop and move to the next iteration
pass	Do nothing. Ignore the condition in which it occurred and proceed to run the program as usual.

Control Statements in Python



BREAK



What is the output ?

```
for val in "string":  
    if val == "i":  
        break  
    print(val)  
  
print("The end")
```



ASCII



Stands for "American Standard Code for Information Interchange."

ASCII is a character encoding that uses numeric codes to represent characters. These include upper and lowercase English letters, numbers, and punctuation symbols.

It is a code for representing 128 English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77.





ACTIVITY TIME



1. Search on Google & Explore the Table of Ascii Characters

2. WAP to print Ascii code of any Character.

3. WAP to Convert a Character Lowercase to Uppercase and vice versa

ASCII

A → 65

a → 97



Programming Lab - 05



- 15.** WAP to print EVEN No Up to = 50

- 16.** WAP to print EVEN No to Nth Number

- 17.** WAP to print ODD No to Nth Number

- 18.** WAP to print EVEN and Odd to Nth Number

- 19.** WAP to print EVEN and Odd to Nth Number With their Sum



TERNARY OPERATOR



Python's if-else statements are very **easy to write** and **read**. But they come with a downside. That is, if you want to print something based on some condition, you need 4 lines to do that.

```
x = 10
if x > 5:
    print("greater")
else:
    print("smaller")
```



That's an awful lot of lines for such a simple operation. What's the solution, you ask? Ternary operators in Python!



TERNARY OPERATOR



The ternary operator in Python is nothing but a **one-line version** of the if-else statement. It provides a way to write conditional statements in a single line, replacing the multi-line if-else syntax.

Syntax of Python ternary operator

```
<true_value> if <conditional_expression> else <false_value>
```





TERNARY OPERATOR



Python ternary operator works with **three operands**:

- 1. conditional_expression:** This is a boolean condition that evaluates to either true or false.
- 2. true_value:** The value returned by the ternary operator if the conditional_expression evaluates to True.
- 3. false_value:** The value returned by the ternary operator if the conditional_expression evaluates to False.





TERNARY OPERATOR



Example

```
#<true_value> if <conditional_expression> else <false_value>

nice_weather=True

print("Go out for a walk" if nice_weather else "watch a movie at home")

nice_weather=False

print("Go out for a walk" if nice_weather else "watch a movie at home")
```



Programming Lab - 06



By Using Ternary Operator

20. WAP to print Whether a Number is Even or Odd

21. WAP to find Largest No B/w Two Numbers



Programming Lab - 07



- 22.** WAP to Swap the 2 Numbers

- 23.** WAP to display a series that is divisible by 7

- 24.** WAP to Calculate the income tax as per the given Slab

Income Slab	Tax Rate
0 - 500000	0%
500000 - 750000	10%
750000 - 1000000	20%
Above 1000000	30%



TYPE CASTING



Type Casting is the method to **convert the variable** data type into a certain data type in order to the operation required to be performed by users.

We can convert one type value to another type.



1. `int()`
2. `float()`
3. `bool()`
4. `str()`





TYPE CASTING



*There can be **two** types of Type Casting in Python –*

Implicit Type Casting

Explicit Type Casting

**Python
Type
Casting**



TYPE CASTING



Implicit Type Casting

In this, methods, Python converts data type into another data type automatically. In this process, users don't have to involve in this process.





Implicit Type Casting



```
3 # Python program to demonstrate
4 # implicit type Casting
5
6 # Python automatically converts a to int
7 a = 7
8 print(type(a))
9
10 # Python automatically converts b to float
11 b = 3.0
12 print(type(b))
13
14 # Python automatically converts c to float as it is a float addition
15 c = a + b
16 print(c)
17 print(type(c))
18
19 # Python automatically converts d to float as it is a float multiplication
20 d = a * b
21 print(d)
22 print(type(d))
```



TYPE CASTING



Explicit Type Casting

In this method, Python need user involvement to convert the variable data type into certain data type in order to the operation required.





TYPE CASTING



Mainly in type casting can be done with these data type function:

int() - *Int() function take float or string as an argument and return int type object.*

float() - *float() function take int or string as an argument and return float type object.*

str() - *str() function take float or int as an argument and return string type object.*



EXPLICIT TYPE CASTING



int to float

```
# Python program to demonstrate  
# type Casting  
  
# int variable  
a = 5  
  
# typecast to float  
n = float(a)  
  
print(n)  
print(type(n))|
```

float to int

```
# Python program to demonstrate  
# type Casting  
  
# int variable  
a = 5.9  
  
# typecast to int  
n = int(a)  
  
print(n)  
print(type(n))|
```



EXPLICIT TYPE CASTING



str to int

```
# Python program to demonstrate  
# type Casting  
  
# string variable  
a = "5"  
  
# typecast to int  
n = int(a)  
  
print(n)  
print(type(n))
```

str to float

```
# Python program to demonstrate  
# type Casting  
  
# string variable  
a = "5.9"  
  
# typecast to float  
n = float(a)  
  
print(n)  
print(type(n))
```



EXPLICIT TYPE CASTING



int to boolean

```
# Python program to demonstrate
# type Casting

# int variable
a = 1

# typecast to bool
n = bool(a)

print(n)
print(type(n))
```



bool(`abc(u)`)
bool(`u`)



TYPE CASTING



What is the value of b

`b = int("10")`

- a Ten
- b 10
- c Value Error
- d Syntax Error





TYPE CASTING



What is the value of b

`b = int("ten")`

- a 10
- b Ten
- c Value Error
- d Syntax Error





TYPE CASTING



What is the value of b

b = float(True)

- a 1
- b 1.0
- c Value Error
- d Syntax Error





TYPE CASTING



What is the value of b

`b = float("10")`

- a 10
- b 10.0
- c Value Error
- d Syntax Error





TYPE CASTING



What is the value of b

1. b = bool(0)
2. b = bool(0.0)
3. b = bool("0")
4. b = bool("0.0")
5. b = bool("10")
6. b = bool(10)
7. b = bool(False)
8. b = bool(True)
9. b = bool(true)
10. b = bool(false)



TYPE CASTING



What is the value of b

```
b = bool("")
```

a True

b False

c Value Error

d Syntax Error





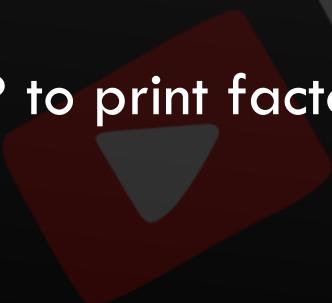
Programming Lab - 08



- 25.** WAP to Print Fibonacci Series

- 26.** WAP to print whether a number is prime or not.

- 27.** WAP to print factorial of a given no.





Programming Lab - 09



- 28.** WAP to print Prime Number 1 to 100.

- 29.** WAP to print Prime Number 1 to Nth Number

- 30.** WAP to print Prime Number 1 to Nth Number with their sum.



Programming Lab - 10



- 31.** WAP to reverse a number

- 32.** WAP to check whether a Number is Palindrome or not.





Mutable VS Immutable



MUTABLE



IMMUTABLE



Mutable VS Immutable



Basis	Mutable	Immutable
Meaning	Mutable means we can change the value after creation	Mutable means we cannot change the value after creation
Example	List , Dictionary , Sets	Int , Float , String , Tuples



FUNDAMENTAL DATA TYPES

V/S

IMMUTABILITY

All Fundamental Data types are **immutable**. I.e once we creates an object , we cannot perform any changes in that object. If we are trying to change then with those changes a **new object will be created**. This non-changeable behaviour is called **immutability**





FUNDAMENTAL DATA TYPES

V/S

IMMUTABILITY



In Python if a new object is required, then PVM wont create object immediately. First it will check is any object available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.

But the problem in this approach is , several references pointing to the same object , by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required. According to this once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.



PROGRAM TO EXPLAIN IMMUTABILITY



Run and Analyze the program

```
a = 10 # a is immutable  
b = 10 # b is immutable  
  
print(a is b)  
  
print(id(a))  
print(id(b))
```



print(id(p))

print(id(s))

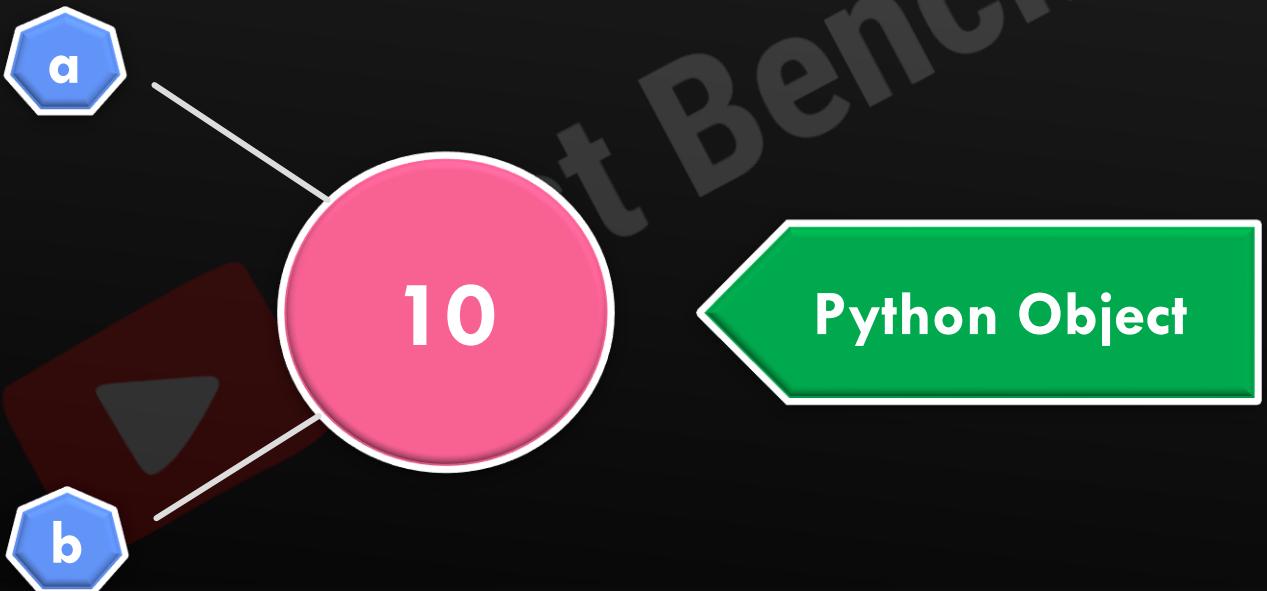


PROGRAM TO EXPLAIN IMMUTABILITY



We Can't Change this Python Object as it is Immutable

Suppose if $a = 10$, $b = 10$ then both are sharing the same object





CONCLUSION



Now you *might* be thinking that we can change the value of *a* and *b*

```
a = 10 # a is immutable  
b = 10 # b is immutable  
  
print(a is b)  
a = 20  
print(a is b)  
  
print(id(a))  
print(id(b))
```

```
print(id(b))  
print(id(a))
```





*When we are trying to change the value of a and b then
A new object will be created and the variable a as in previous example
Is pointing towards that object*



The diagram illustrates the behavior of mutable objects in Python. It shows two variables, `b` and `a`, represented as blue hexagons. Variable `b` is connected by a line to a pink circle containing the number `10`. Variable `a` is connected by a line to a green circle containing the number `20`. A large red arrow points from the `10` circle to the `20` circle, indicating that modifying the value of `a` creates a new Python object. Two green arrows point away from the circles, labeled "Python Object" and "New Python Object Created".



PATTERNS

1	1	1	5	5	5
1 2	2 1	2 2	4 4	4 5	5 4
1 2 3	3 2 1	3 3 3	3 3 3	3 4 5	5 4 3
1 2 3 4	4 3 2 1	4 4 4 4	2 2 2 2	2 3 4 5	5 4 3 2
1 2 3 4 5	5 4 3 2 1	5 5 5 5 5	1 1 1 1 1	1 2 3 4 5	5 4 3 2 1

← PATTERN PROGRAMS →

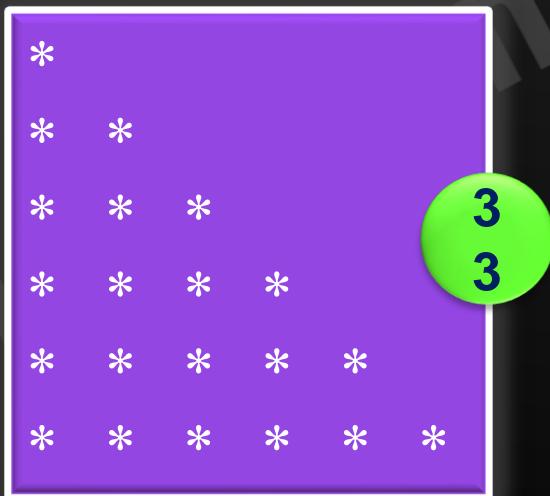
1	1	1	5	5	5
1 2	2 1	2 2	4 4	4 5	5 4
1 2 3	3 2 1	3 3 3	3 3 3	3 4 5	5 4 3
1 2 3 4	4 3 2 1	4 4 4 4	2 2 2 2	2 3 4 5	5 4 3 2
1 2 3 4 5	5 4 3 2 1	5 5 5 5 5	1 1 1 1 1	1 2 3 4 5	5 4 3 2 1



Programming Lab - 11



WAP to print Star Pattern





Programming Lab - 12



1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

3
4

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

3
5

1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

3
6

5
4 4
3 3 3
2 2 2 2
1 1 1 1 1

3
7

5
5 4
5 4 3
5 4 3 2
5 4 3 2 1

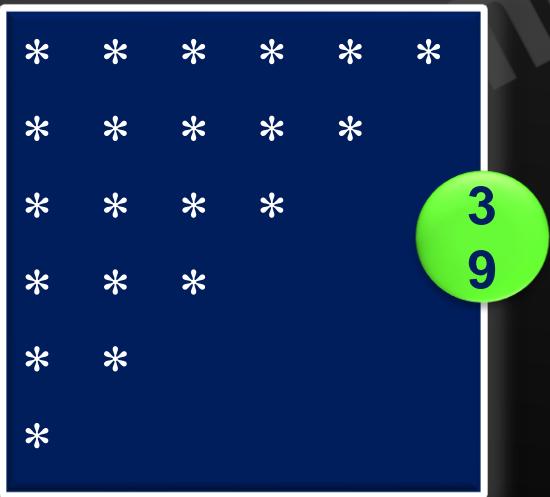
3
8



Programming Lab - 13



WAP to print Star Pattern





Programming Lab - 14



5	5	5	5	5
4	4	4	4	
3	3	3		
2	2			
1				

4
0

5	4	3	2	1
5	4	3	2	
5	4	3		
5	4			
5				

4
1

1	2	3	4	5
6	7	8	9	
10	11	12		
13	14			
15				

42

1	2	3	4	5
1	2	3	4	
1	2	3		
1	2			
1				

43

1	1	1	1	1
2	2	2	2	
3	3	3		
4	4			
5				

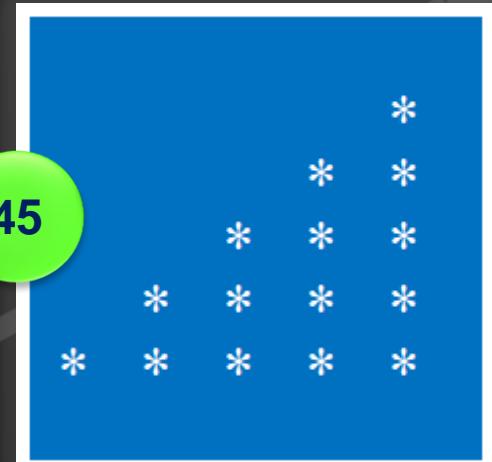
44



Programming Lab - 15



WAP to print Star Pattern



45





Programming Lab - 16



46

				5
		4	4	
	3	3	3	
2	2	2	2	
1	1	1	1	1

47

				5
		5	4	
	5	4	3	
	5	4	3	2
5	4	3	2	1

48

				1
		2	3	
	4	5	6	
7	8	9	10	
11	12	13	14	15

49

				1
		2	2	
	3	3	3	
4	4	4	4	
5	5	5	5	5

50

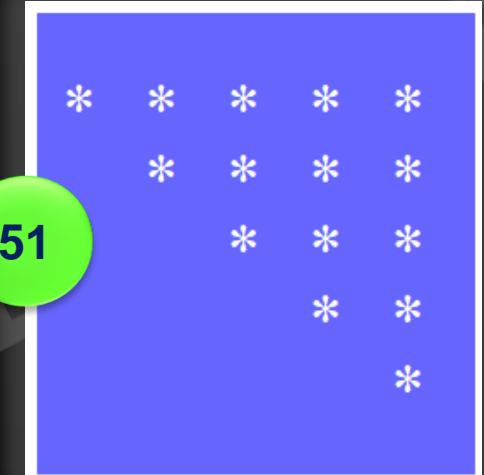
				1
	1	2		
1	2	3		
1	2	3	4	
1	2	3	4	5



Programming Lab - 17



WAP to print Star Pattern



51





Programming Lab - 18



52

5	5	5	5	5
4	4	4	4	
3	3	3	3	
2	2			
1				

53

5	4	3	2	1
5	4	3	2	
5	4	3		
5	4			
5				

54

1	2	3	4	5
6	7	8	9	
10	11	12		
13	14			
15				

55

1	1	1	1	1
2	2	2	2	2
3	3	3	3	
4	4			
5				

56

1	2	3	4	5
1	2	3	4	
1	2	3		
1	2			
1				

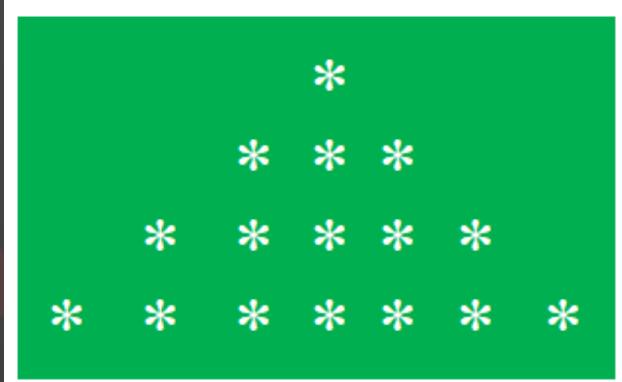


Programming Lab - 19



WAP to print Star Pattern

57





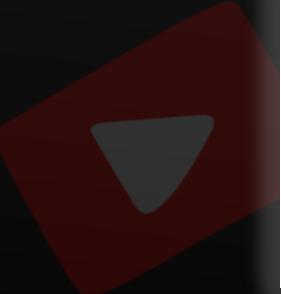
Programming Lab - 20



WAP to print Star Pattern

58

```
* * * * * * *  
* * * * * *  
* * *  
*
```





Programming Lab - 21



WAP to print Star Pattern

59

```
    *
   * *
  * * *
 * * * *
* * * * *
* * * * *
* * * *
*
```



Programming Lab - 22



60. WAP Sum of every number in a Digit

for ex = 12345 => $1+2+3+4+5 = 15$ Output

61. WAP Sum of every Odd number in a Digit

for ex = 12345 => $1+3+5 = 9$ Output

62. WAP Sum of every Even number in a Digit

for ex = 12345 => $2+4 = 6$ Output



BYTES



*bytes data type represents a group of byte numbers just like an array
i.e. it is immutable*

```
x = [10,20,30,40]
b = bytes(x)

print(type(b)) ==> bytes
print(b[0])      ==> 10
print(b[-1])     ==> 40

for i in b :
    print(i)
```



Python Bytes()



BYTES



Python Bytes()

Conclusion 1:

The **only allowed values for byte data type** are **0 to 256**. By mistake if we are **trying to provide any other values** then we will get **value error**.

Conclusion 2:

Once we creates bytes data type value, we **cannot change its values** , otherwise we will get **TypeError**



BYTES ARRAY



bytearray is exactly same as bytes data type except that its elements can be modified i.e. **it is mutable**

```
a = [2, 255, 4]
b = bytearray(a)

b[2] = 140 # i.e it's mutable

print(b[2])
print(type(b))
```



BytesArray()



BE ATTENTIVE





LISTS



IMPORTANT



LISTS



*Lists are used to store **multiple items** in a single variable.*

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

```
I = ["Katrina","Vicky",007,True,10.5]
```



Python Lists



LISTS



If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.

- 1. insertion order is preserved*
- 2. heterogeneous objects are allowed*
- 3. duplicates are allowed*
- 4. Growable in nature*
- 5. values should be enclosed within square brackets.*



Python Lists



LISTS



EMPTY LIST



CREATION OF LISTS



1. We can create *empty list object* as follows...

```
# Empty List  
list = []  
  
print(type(list))
```





CREATION OF LISTS



2. If we know elements already then we can create list as follows

```
list = [10,20,30,40]  
print(type(list))
```





CREATION OF LISTS



3. We can create list with `list()` Function It is function which can typecast compatible objects to list type

```
str = "Katrina Kaif"  
list = list(str)  
  
print(list)  
print(type(list))
```





CREATION OF LISTS



**4. With dynamic input:
By using eval() Function**

```
list=eval(input("Enter List:"))

print(list)
print(type(list))
```





CREATION OF LISTS



5. With *split()* Function

```
str ="Alia bhatt Weds Ranbir Kapoor"  
  
list = str.split(" ")  
print(list)  
print(type(list))
```





ACCESSING ELEMENTS OF LISTS



We can access elements of the list either by using index or by using slice operator(:)

1. By using index:

2. By using slice operator:





ACCESSING ELEMENTS OF LISTS



```
L = [ 140 , "HELLO" , 15.25 , True , 0 ]
```





ACCESSING ELEMENTS OF LISTS



1. By using index:

List follows zero based index. ie index of first element is zero.

List supports both +ve and -ve indexes.

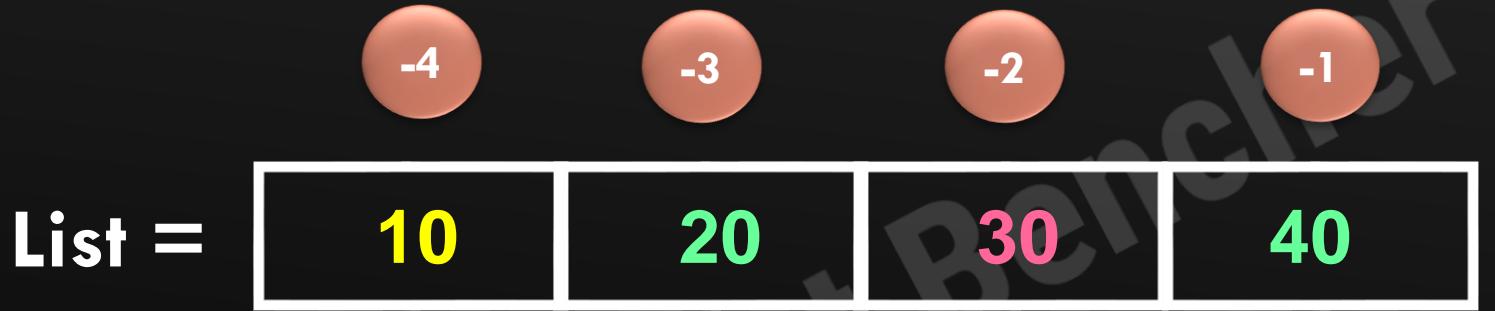
-ve index meant for Right to Left



+ve index meant for Left to Right



ACCESSING ELEMENTS OF LISTS



```
print(List[0])
```

```
print(List[-1])
```

```
print(List[100])
```



ACCESSING ELEMENTS OF LISTS



2. By using slice operator:

The *list* is an *iterable data-type* and hence we can *slice* the *list*. The *list* is the same as an *array* in another programming language. By using *slicing* we can *traverse* through the whole *list*, we can *reverse* it and *print* it from a specific *start index* to *ending index*.

Slicing concept you have already learned in Strings It is same that of list

[start:stop:step]

List Slicing



ACCESSING ELEMENTS OF LISTS

Slicing of List in Python?

`list [start : stop : step]`

List
Variable

Start index
of List
(included)

Stop index
of List
(excluded)

Step Size
Default 1



ACTIVITY TIME



```
n=[1,2,3,4,5,6,7,8,9,10]
```

- 1) print(n[2:7:2])
- 2) print(n[4::2])
- 3) print(n[3:7])
- 4) print(n[8:2:-2])
- 5) print(n[4:100])



2) print(n[4:100])
3) print(n[8:2:-2])



ACTIVITY TIME



1. WAP to Reverse a List using Slicing





TRaversing ELEMENTS OF LISTS



The sequential access of each element in the list is called traversal.

1. By using *while* loop:

2. By using *for* loop:



TRaversing Elements Of Lists



1. By using *while* loop:

```
n=[1,2,3,4,5,6,7,8,9,10]  
  
i=0  
while i<len(n):  
    print(n[i])  
    i+=1
```

Traversing a
List

using While Loop



TRaversing Elements Of Lists



2. By using **for** loop:

```
n=[0,1,2,3,4,5,6,7,8,9,10]  
  
for i in n:  
    print(i)
```

Traversing a
List

using For Loop

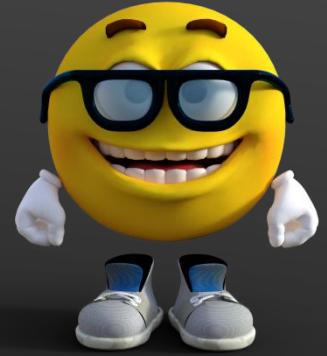


DISPLAY ONLY EVEN NUMBERS



```
n=[0,1,2,3,4,5,6,7,8,9,10]

for i in n:
    if i%2==0:
        print(i)
```





ACTIVITY TIME



1. WAP to display elements in Lists by index wise
negative and positive index





BUILT IN FUNCTIONS FOR LIST



To get information about list

1	<code>len()</code>	returns the number of elements present in the list
2	<code>.count()</code>	It returns the number of occurrences of specified item in the list
3	<code>.index()</code>	returns the index of first occurrence of the specified item



LEN () & COUNT()



len()

Python List

```
list1 = [10,20,30,40]
print(len(list1))
```

count()

Python List

```
list2 = [1,2,2,2,2,3,3]
print(list2.count(1))
print(list2.count(2))
print(list2.count(3))
print(list2.count(4))
```



INDEX()



Note: If the specified element not present in the list then we will get ValueError. Hence before index() method we have to check whether item present in the list or not by using in operator.

```
list = [1,2,2,2,2,3,3]
print(list.index(1)) ==>0
print(list.index(2)) ==>1
print(list.index(3)) ==>5
print(list.index(4)) ==>ValueError: 4 is not in list
```

index()

Python List



BUILT IN FUNCTIONS FOR LIST



Manipulating elements of List

1	.append()	We can use append() function to add item at the end of the list.
2	.insert()	To insert item at specified index position
3	.extend()	To add all items of one list to another list
4	.remove()	We can use this function to remove specified item from the list.If the item present multiple times then only first occurrence will be removed
5	.pop()	It removes and returns the last element of the list.



APPEND()



We can use append() function to add item at the end of the list.

```
list=[]  
  
list.append("A")  
list.append("B")  
list.append("C")  
  
print(list)
```





APPEND()



List =

10	20	30	40
----	----	----	----

List = []

List.append(10)
List.append(20)
List.append(30)
List.append(40)

RUN



APPEND()



To add all elements to list upto 100 which are divisible by 10

```
list=[]

for i in range(101):
    if i%10==0:
        list.append(i)

print(list)
```



INSERT()



To **insert item** at specified index position

```
list=[1,2,3,4,5]  
  
list.insert(1,888)  
print(list)
```

If the specified index is greater than max index then element will be inserted at last position. If the specified index is smaller than min index then element will be inserted at first position.

insert()

Python List



INSERT()



List =

10	20	30	40
----	----	----	----

List =

10	20	500	30	40
----	----	-----	----	----

```
List = [10,20,30,40]  
List.insert(2,500)  
print(List)
```

insert()
Python List

RUN



EXTEND()



To **add all items** of one list to another list

l1.extend(l2)

all items present in l2 will be added to l1

```
order1=["Chicken", "Mutton", "Fish"]
order2=["JalebiFafda", "PavBhaji", "ChillyPotato"]
```

```
order1.extend(order2)
```

```
print(order1)
```

extend()

Python List



REMOVE()



We can use this function to **remove specified item** from the list. If the item present multiple times then only first occurrence will be removed.

```
actress=['Alia','Katrina','Deepika']
actress.remove('Katrina')
print(actress)
```

remove()

Python List



REMOVE()



If the specified item not present in list then we will get **ValueError**

```
actress=['Alia','Katrina','Deepika']

actress.remove('Priyanka') ==> ValueError

print(actress)
```

remove()

Python List

Note: Hence before using **remove()** method first we have to check specified element present in the list or not by using **IN** operator.



POP()



It removes and returns the last element of the list.
This is only function which manipulates list and returns some element.

```
n=[10,20,30,40]  
  
print(n.pop())  
print(n.pop())  
  
print(n)  => 30 and 40 removed
```

pop()

Python List



POP()



List = [10, 20, 30, 40] POP

List = [10, 20, 30] POP

List = [10, 20]



POP()



Note:

1. If the list is empty then pop() function raises IndexError
2. pop() is the only function which manipulates the list and returns some value
3. In general we can use pop() function to remove last element of the list. But we can use to remove elements based on index.

pop()

Python List



POP()



n.pop(index)==>To remove and return element present at specified index.

n.pop()==>To remove and return last element of the list

```
n=[10,20,30,40,50,60]
print(n.pop()) #60
print(n.pop(1)) #20
print(n.pop(10)) ==>IndexError: pop index out of range
```



SUMMARY



List objects are dynamic. i.e based on our requirement we can increase and decrease the size.

`append()`,`insert()` ,`extend()` ➔ for **increasing** the size/growable nature

`remove()`, `pop()` ➔ for **decreasing** the size /shrinking nature



BUILT IN FUNCTIONS FOR LIST



Ordering elements of List:

1 `.reverse()` We can use to reverse() order of elements of list

2 `.sort()` If want to sort the elements of list according to default natural sorting order then we should go for sort() method.



REVERSE()



We can use to **reverse()** order of elements of list.

```
n=[10,20,30,40]  
n.reverse()  
print(n)
```



reverse()

Python List



REVERSE()

List =

10	20	30	40
----	----	----	----

Reverse

List =



SORT()



In list by default insertion order is preserved. If want to sort the elements of list according to default natural sorting order then we should go for sort() method.

For numbers → default natural sorting order is Ascending Order
For Strings → default natural sorting order is Alphabetical Order



```
n=[20,5,15,10,0]
n.sort()
print(n)
```

sort()

Python List



SORT()



To sort in **reverse** of default natural sorting order:

We can sort according to reverse of default natural sorting order by using `reverse=True` argument.

```
n=[20,5,15,10,0]
n.sort(reverse=True)
print(n)
```





SORT()

n =



n.sort()

n =

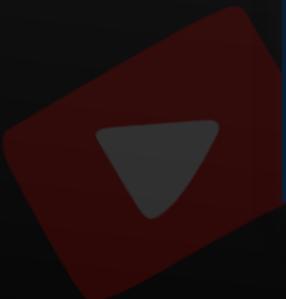


CLEAR()



We can use `clear()` function to remove all elements of List.

```
n=[10,20,30,40]  
print(n)  
  
n.clear()  
print(n)
```



`clear()`

Python List



APPEND() VS INSERT()



append()

In List when we add any element it will come in last i.e. it will be last element.

insert()

In List we can insert any element in particular index number



REMOVE() V/S POP()



remove()	pop()
We can use to remove special element from the List.	We can use to remove last element from the List.
It can't return any value.	It returned removed element.
If special element not available then we get VALUE ERROR.	If List is empty then we get Error.



Aliasing AND Cloning of List



Aliasing of List



Cloning of List



Aliasing of List



The process of giving another reference variable to the existing list is called Aliasing.

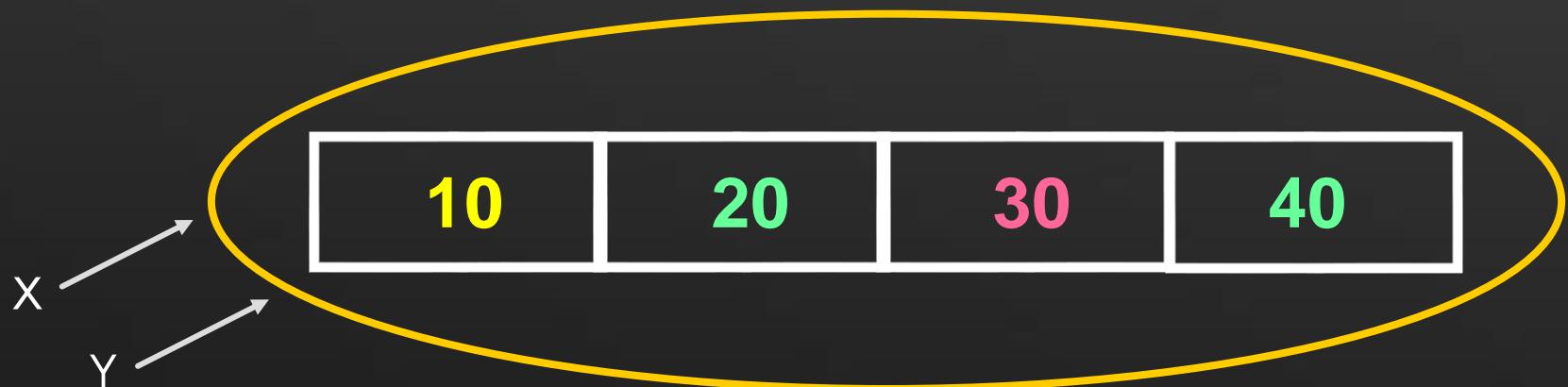
```
x=[10,20,30,40]  
y=x  
print(id(x))  
print(id(y))
```

PYTHON

Aliasing List



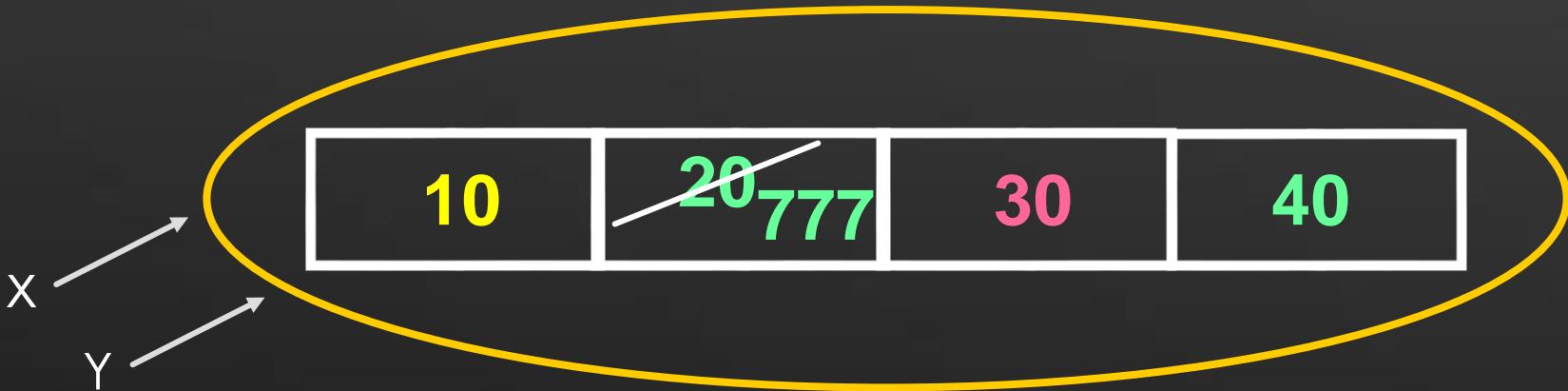
Aliasing of List



The problem in this approach is by using one reference variable if we are changing content, then those changes will be reflected to the other reference variable.



Aliasing of List



```
x=[10,20,30,40]
y=x    # Aliasing

y[1]=777
print(x)
```

To overcome this problem we should go for cloning.



Cloning of List



The process of creating exactly duplicate independent object is called cloning.

We can implement cloning by



- using slice operator
- using copy() function

Python

**Copying and
Cloning []**



Cloning of List



1. Cloning by using slice operator

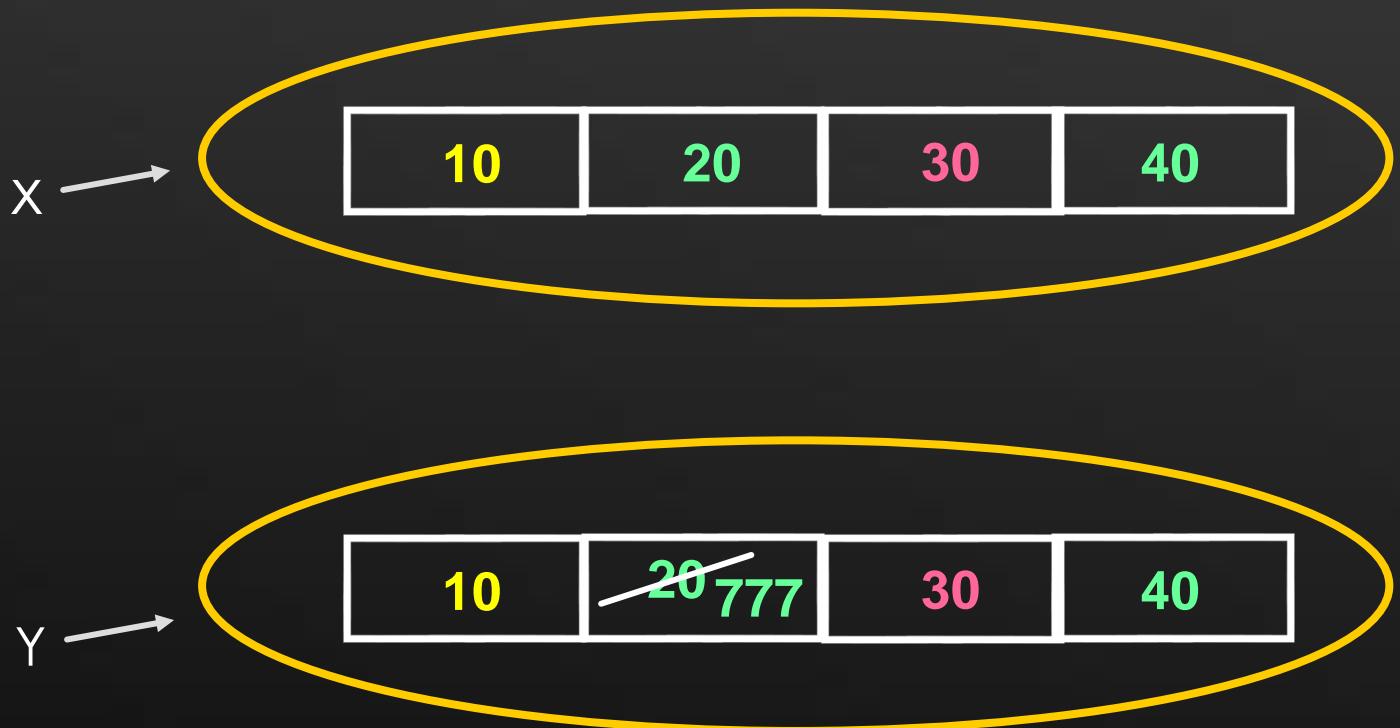
```
x=[10,20,30,40]  
y=x[:]  
y[1]=777  
  
print(x) ==>[10,20,30,40]  
print(y) ==>[10,777,30,40]
```

Python

**Copying and
Cloning []**



Cloning of List





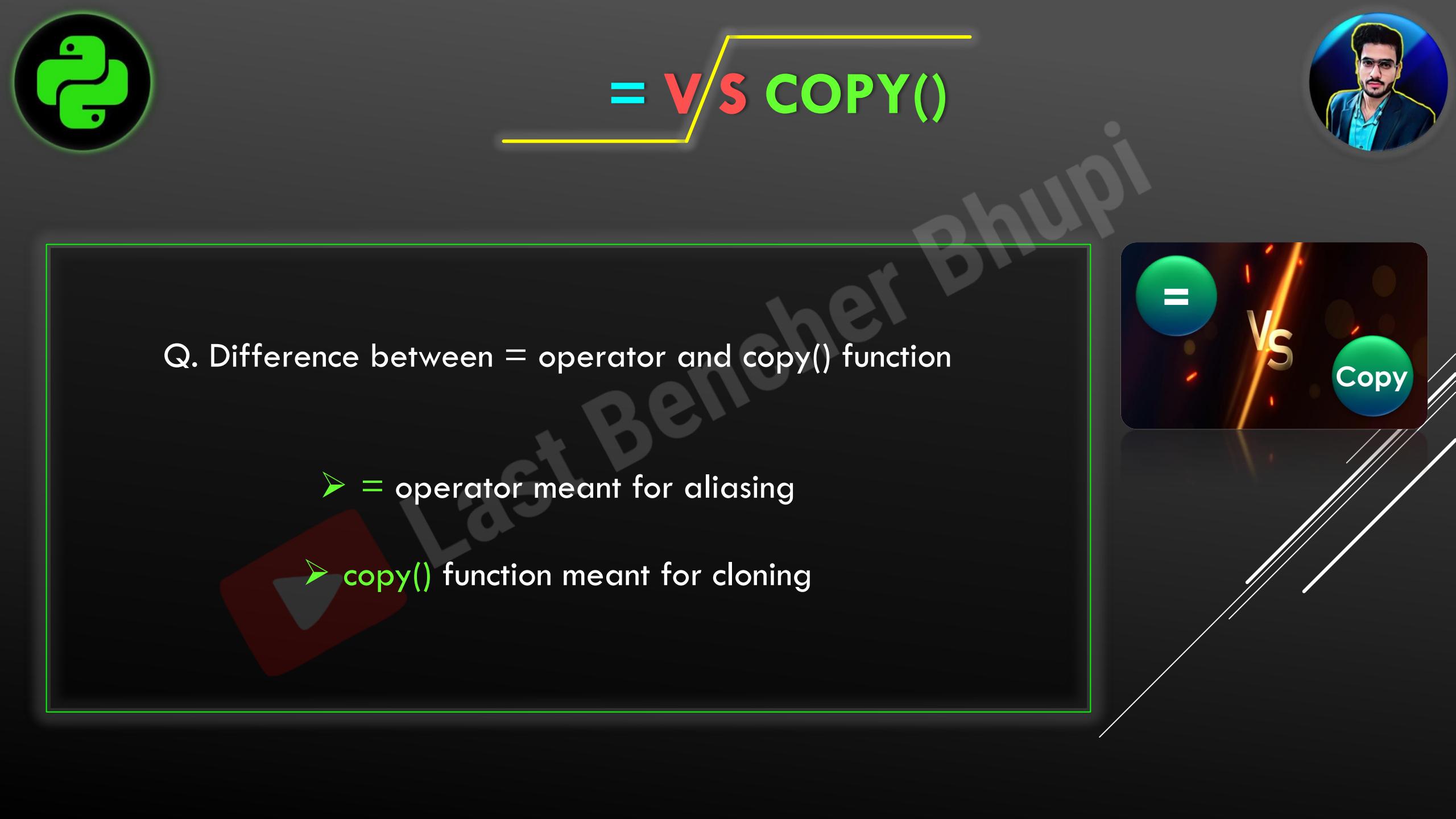
Cloning of List



2. Cloning by using `copy()` function

```
x=[10,20,30,40]  
y=x.copy()  
y[1]=777  
  
print(x) ==>[10,20,30,40]  
print(y) ==>[10,777,30,40]
```





Q. Difference between = operator and copy() function

- = operator meant for aliasing
- copy() function meant for cloning

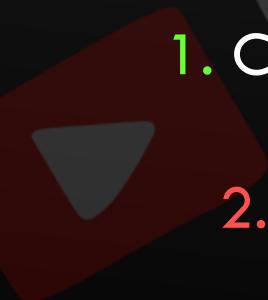


MATHMATICAL OPERATORS FOR LIST



Mathematical operators for List Objects

We can use + and * operators for List objects.



1. Concatenation operator(+)

2. Repetition Operator(*)





MATHMATICAL OPERATORS FOR LIST



1. Concatenation operator(+)

We can use + to concatenate 2 lists into a single list

```
a=[10, 20, 30]  
b=[40, 50, 60]  
  
c=a+b  
print(c) ==>[10, 20, 30, 40, 50, 60]
```

Note: To use + operator compulsory **both arguments** should be list objects , otherwise we will get TypeError .





MATHMATICAL OPERATORS FOR LIST



2. Repetition Operator(*)

We can use repetition operator * to repeat elements of list specified number of times

```
x=[10,20,30]  
y=x*3  
  
print(y)==>[10,20,30,10,20,30,10,20,30]
```





Comparing Lists



We can use comparison operators for List objects.

```
x=["KGF","RRR","PUSHPA"]
y=["KGF","RRR","PUSHPA"]
z=["Kgf","Rrr","Pushpa"]

print(x==y) True
print(x==z) False
print(x != z) True
```



Comparing Lists



Note: Whenever we are using comparison operators(==,!=) for List objects then the following should be considered

1. The number of elements
2. The order of elements
3. The content of elements (case sensitive)

Note: Whenever we are using relational operators(<,<=,>,>=) between List objects , only first element comparison will be performed.





MEMBERSHIP OPERATORS FOR LIST



- We have already learned Membership Operator in Operators Concept We are here just to aware the working of these operator with python Lists
- We can check whether element is a member of the list or not by using membership operators.



in operator

not in operator





MEMBERSHIP OPERATORS FOR LIST



Example

```
n=[10,20,30,40]  
  
print (10 in n)      True  
print (10 not in n) False  
print (50 in n)      False  
print (50 not in n) True
```

Python Membership
Operators

IN NOT
IN



ACTIVITY TIME



```
n = ['Katrina', 'Alia', 'Deepika', 'Aishwarya', 'Rashmika']
```

On the basis of this Find 'rashmika' is present in the list or not.





Nested LIST



Nested Lists

PYTHON

- Lists are useful data structures commonly used in Python programming. A nested list is a list of lists, or any list that has another list as an element (a sublist).
- Sometimes we can take **one list inside another list**. Such type of lists are called nested lists.



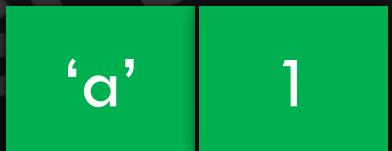
Nested LIST



list



List[2]



An Example of Nested List



Nested LIST



Run and Analyze the Example

```
n=[10,20,[30,40]]  
print(n)  
print(n[0])  
print(n[2])  
print(n[2][0])  
print(n[2][1])
```



Nested
Lists

PYTHON



List Comprehensions



It is very **easy** and **compact** way of creating list objects from any iterable objects (like list, tuple, dictionary, range etc) based on some condition.



SYNTAX

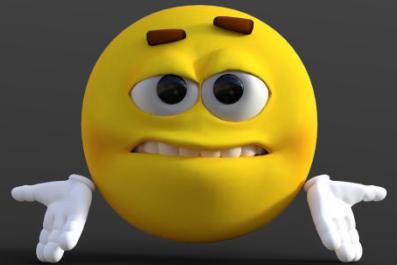


List = [expression for item in list if condition]



ACTIVITY TIME

1. Simple list to store num from 1 to 10
2. Simple list to store even numbers from 1 to 100
3. Simple list to store square of num from 1 to 10



**Using List
Comprehension**





Programming Lab - 23



- 63.** WAP to Input values through user in list and print their sum.

- 64.** WAP to print Odd & Even Numbers to in a list to Nth Number.





Programming Lab - 24



- 65.** WAP to Input values through user in list and print it.

- 66.** WAP to Input values through user in list and Update it.

- 67.** WAP to Input values through user in list and Insert value in it.

- 68.** WAP to Input values through user in list and perform search operation.

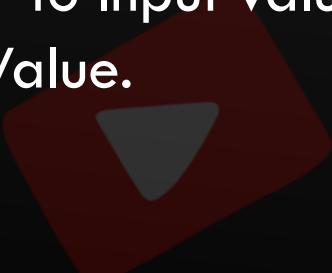


Programming Lab - 25



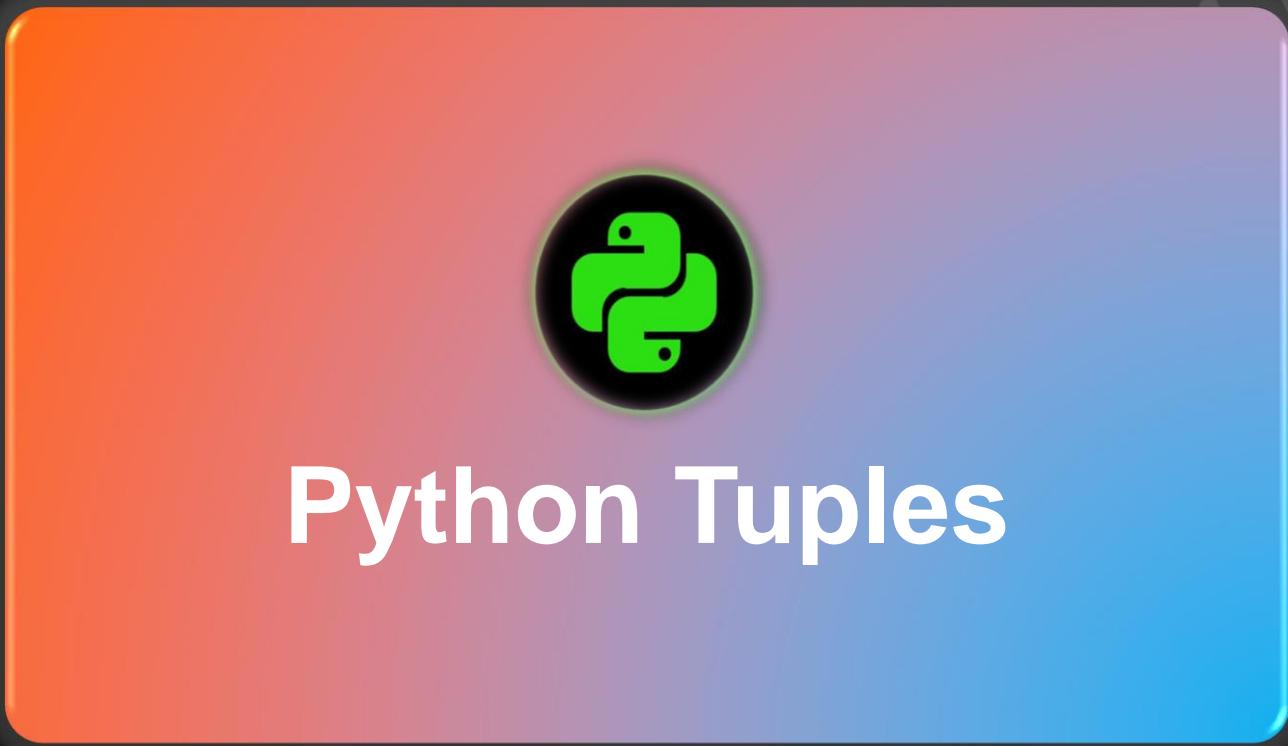
- 69.** WAP to Input values through user in list and Delete an Element by Index.

- 70.** WAP to Input values through user in list and Delete an Element by Value.





TUPLES



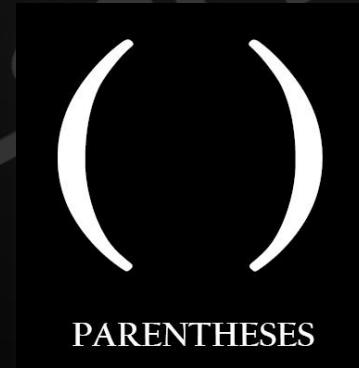
IMPORTANT



TUPLES



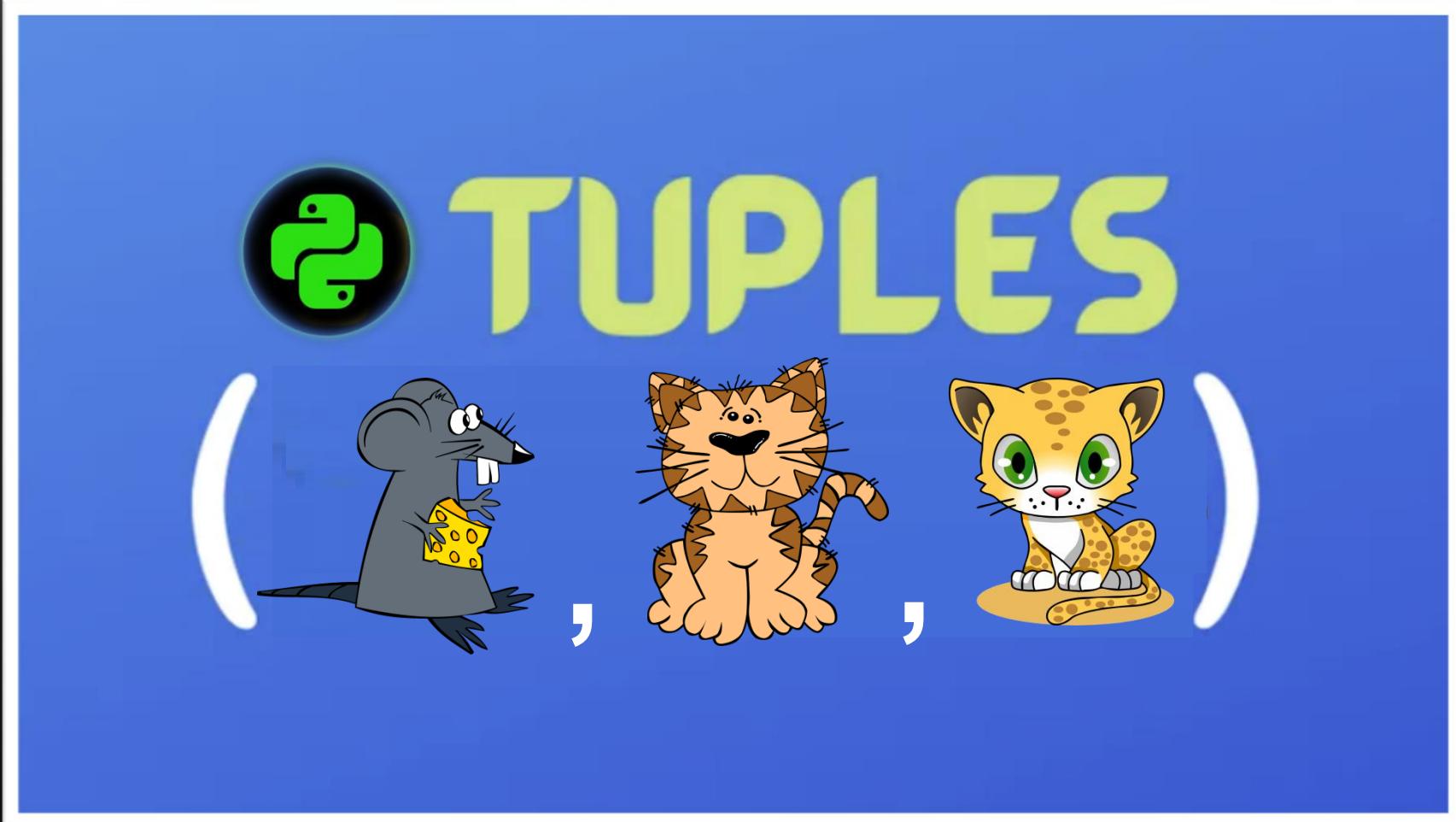
Python tuples are used to store an ordered sequence of values. Tuples are **immutable**, this means that the values in a tuple cannot be changed once the tuple is defined. The values in a tuple are comma-separated and they are surrounded by parentheses.



PARENTHESES



TUPLES





TUPLES



1. Tuple is exactly same as List except that it is immutable. i.e once we creates Tuple object , we cannot perform any changes in that object.
Hence Tuple is Read Only version of List.
2. If our data is fixed and never changes then we should go for Tuple
3. Insertion Order is preserved
4. Duplicates are allowed





TUPLES



5. Heterogeneous objects are allowed.
6. We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also. Tuple support both +ve and -ve index. +ve index means forward direction(from left to right) and -ve index means backward direction(from right to left)
7. We can represent Tuple elements within Parenthesis and with comma separator. Parenthesis are optional but recommend to use





CREATION OF TUPLES



1. We can create empty Tuple object as follows...

```
# Empty Tuple  
t = ()  
  
print(type(t)) -> <type 'tuple'>
```





CREATION OF TUPLES



2. If we know elements already then we can create tuple as follows

```
t=(10,20,30,40)  
print(type(t)) -> <type 'tuple'>
```





CREATION OF TUPLES



3. We can create list with tuple() Function It is function which can typecast compatible objects to tuple type

```
list=[10,20,30]
t=tuple(list)

print(t)

t=tuple(range(10,20,2))
print(t)
```





CREATION OF TUPLES



What is the output

L = [10]

print(type(L))

- a Tuple
- b List
- c Int
- d Value Error





CREATION OF TUPLES



What is the output

T = (10)

print(type(T))

- a Tuple
- b List
- c Int
- d Value Error





CONCLUSION



Most of you are saying it's a tuple , But my friend it's not a Tuple 😂

Confidence to dekho





CONCLUSION



Note: We have to take special care about single valued tuple, compulsory the value should ends with comma , otherwise it is not treated as tuple. It is treated as int



```
t=(10)  
print(t) --> 10  
print(type(t)) --> int
```

```
t=(10,)  
print(t) --> (10,)  
print(type(t)) -->
```





Which of the following are valid TUPLES



Write down the answers in your notebook then run the one by one and analyze as per your answers

1. `t=()`
2. `t=10, 20, 30, 40`
3. `t=10`
4. `t=10,`
5. `t=(10)`
6. `t=(10,)`
7. `t=(10, 20, 30, 40)`





Accessing Elements of TUPLES



We can access elements of the tuples either by using index or by using slice operator(:)

1. By using index:

2. By using slice operator:



Python Tuples



Accessing Elements of TUPLES



1. By using index:

Tuple follows zero based index. ie index of first element is zero.

Tuple supports both +ve and -ve indexes.

-ve index meant for Right to Left



+ve index meant for Left to Right

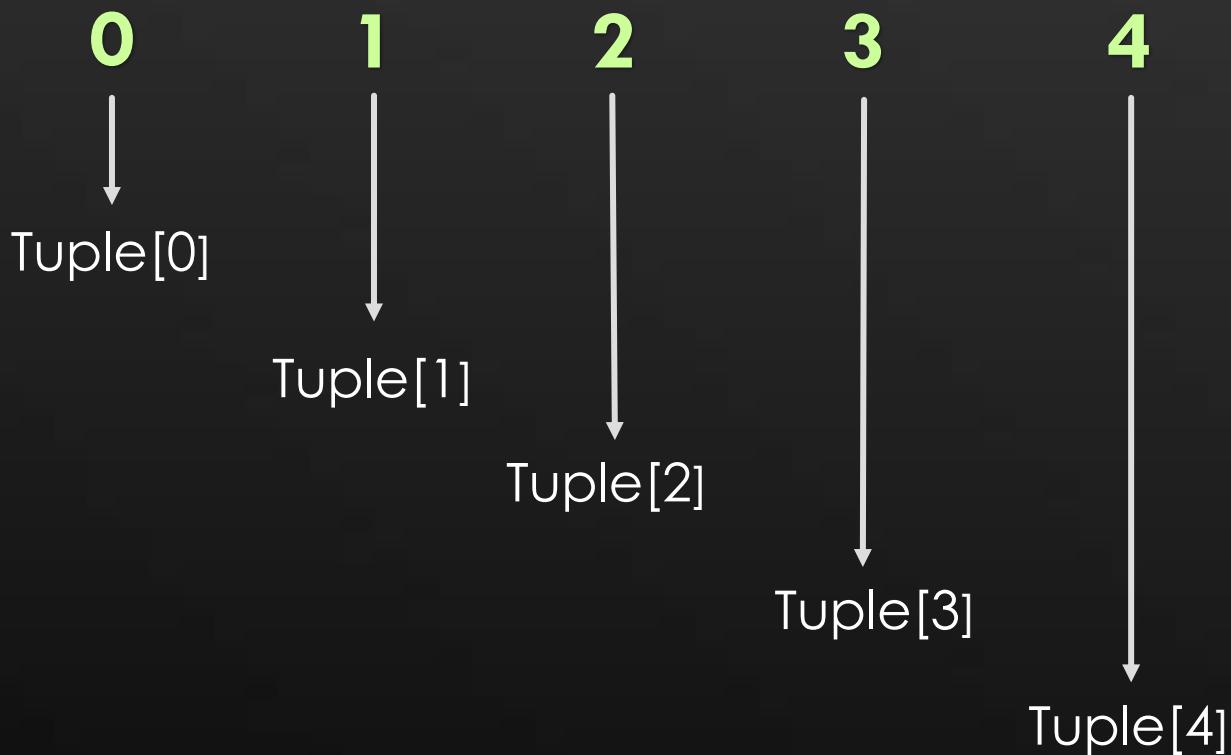


Python Tuples



Accessing Elements of TUPLES

Tuple = ('a' , 'e' , 'i' , 'o' , 'u')





Accessing Elements of TUPLES



2. By using slice operator:

We have already covered this concept in Lists it performs same functionality as that of list if you can't remember then refer Video 94 Accessing List





ACTIVITY TIME



```
t=(10,"Kiara",30,"50.0",50,"Yash")  
  
print(t[2:5])  
print(t[2:100])  
print(t[::-2])
```





ACTIVITY TIME



1. WAP to Reverse a Tuple using Slicing





“ index out of range “

TUPLES



In the previous slides we have seen how to use positive and negative indexes to access the items in a tuple.

Let's see what happens if we use indexes that fall outside the boundaries of the tuple.

```
Movies = ('KGF2', 'RRR', 'Pushpa', 'Bahubali', 'KGF')
```

For this tuple the maximum valid positive index is 4 and it refers to the last item. Here is what happens if we use the index 5...



“ index out of range “

TUPLES



```
print(Movies[5])
```

```
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(Movies[5])
IndexError: tuple index out of range
```

Something similar also applies to negative indexes...

The Python interpreter raises an “tuple index out of range” exception when trying to access an item in the tuple by using a positive index greater than the size of the tuple minus one. The same error also occurs when trying to access an item by using a negative index lower than the size of the tuple.



Traversing the Elements of TUPLES



The sequential access of each element in the Tuple is called traversal.

1. By using **while** loop:

2. By using **for** loop:

It is exactly same as the lists Refer Video 95



Traversing the Elements of TUPLES



1. By using *while* loop:

```
t = (1,2,3,4,5)  
  
i=0  
while (i<len(t)):  
    print(t[i])  
    i=i+1
```

Traversing a
Tuple

using While Loop



Traversing the Elements of TUPLES



2. By using *for loop*:

```
t = (1,2,3,4,5)  
for i in t:  
    print(i)
```

Traversing a
Tuple

using For Loop



ACTIVITY TIME



What is the Output ?

```
tuple = (1,2,3,4,5)

k=0
while k<len(tuple):

    print(tuple[k])
    if tuple[k] == 4:
        break
    k++
```





CONCLUSION



If you are coming from C or Java Background then you must predicted something but the answer is **Syntax error**

k++ be Like





TUPLE VS IMMUTABILITY



A tuple is immutable, this means that once created you cannot add more elements to a tuple i.e, we cannot change its content.

When you do that you get back a `TypeError` exception that says that an object of type tuple does not support item assignment.

Hence tuple objects are immutable.





TUPLE VS IMMUTABILITY



```
Movies = ('KGF2', 'RRR', 'Pushpa')

# Try to assign value at 1 index

Movies[1] = "Alia Bhatt"
```

```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    Movies[1] = "Alia Bhatt"
TypeError: 'tuple' object does not support item assignment
```





MATHMATICAL OPERATORS FOR TUPLES



Mathematical operators for Tuple Objects

We can use + and * operators for Tuple objects.



1. Concatenation operator(+)
2. Repetition Operator(*)





MATHMATICAL OPERATORS FOR TUPLES



1. Concatenation operator(+)

We can use + to concatenate 2 tuples into a single tuple

```
t1 = (10, 20, 30)
t2 = (40, 50, 60)

t3 = t1 + t2

print(t3)
```



Note: To use + operator compulsory both arguments should be tuple objects , otherwise we will get **TypeError** .



MATHMATICAL OPERATORS FOR TUPLES



2. Repetition Operator(*)

We can use repetition operator * to repeat elements of tuples specified number of times

```
t1=(10,20,30)  
t2=t1*3  
print(t2)
```





Deleting a Tuple



Tuples are immutable objects so we cannot change or modify tuples.

Hence now we conclude that we cannot delete an element from the tuple.

But we can delete the whole tuple.

```
tup=(10,20,30)  
  
print(tup) --> (10,20,30)  
del(tup)  
print(tup) --> NameError
```





BUILT IN FUNCTIONS FOR TUPLES



Some Important Functions of Tuple

1	<code>len()</code>	To return number of elements present in the tuple
2	<code>.count()</code>	To return number of occurrences of given element in the tuple
3	<code>.index()</code>	returns index of first occurrence of the given element. If the specified element is not available then we will get ValueError.
4	<code>sorted()</code>	To sort elements based on default natural sorting order
5	<code>min()</code>	Returns Minimum value
6	<code>max()</code>	Returns Maximum value



LEN()



To return number of elements present in the tuple

```
tup=(10,20,30)  
print(len(tup)) --> 3
```



LEN()

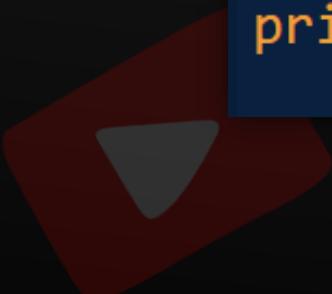


COUNT()



To return number of occurrences of given element in the tuple

```
t=(10,20,10,10,20)  
print(t.count(10)) --> 3
```



COUNT()



INDEX()



Returns index of *first occurrence* of the given element.
If the specified element is not available then we will get **ValueError**.

```
t=(10,20,10,10,20)  
  
print(t.index(10)) --> 0  
print(t.index(30)) --> ValueError:
```



INDEX ()



SORTED()



To sort elements based on default natural sorting order

```
t=(40,10,30,20)  
t1=sorted(t)  
print(t1) --> Ascending
```

We can sort according to reverse of default natural sorting order as follows

```
t1=sorted(t,reverse=True)  
print(t1) --> Descending
```





MIN() & MAX()



These functions return min and max values

```
t=(40,10,30,20)  
  
print(min(t)) --> 10  
print(max(t)) --> 40
```



Min ()
And
Max ()

PYTHON



PACKING AND UNPACKING OF TUPLES





TUPLE PACKING



TUPLE PACKING

We can create a tuple by packing a group of variables.

```
a=10  
b=20  
c=30  
d=40  
  
t=a,b,c,d  
print(t) #(10, 20, 30, 40)
```

Here a,b,c,d are packed into a tuple t. This is nothing but tuple packing.





TUPLE UNPACKING



TUPLE UNPACKING

Tuple unpacking is the reverse process of tuple packing

We can *unpack a tuple and assign its values to different variables*

```
t=(10,20,30,40)  
a,b,c,d = t # UnPacking  
print("a=",a,"b=",b,"c=",c,"d=",d)
```



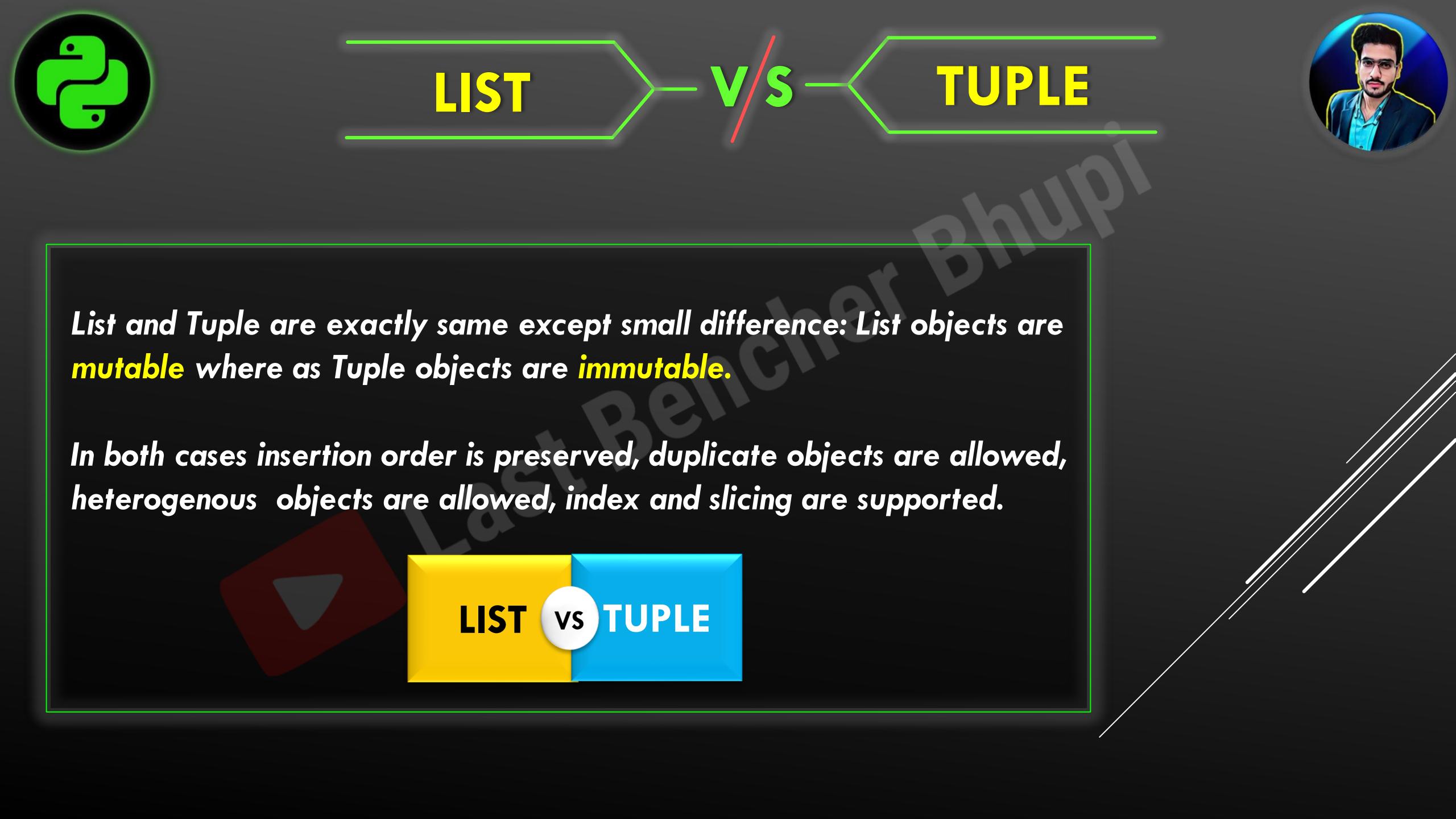


LIST

V/S

TUPLE





LIST

TUPLE

v/s

*List and Tuple are exactly same except small difference: List objects are **mutable** where as Tuple objects are **immutable**.*

In both cases insertion order is preserved, duplicate objects are allowed, heterogenous objects are allowed, index and slicing are supported.

LIST vs TUPLE





LIST V/S TUPLE



Basis	LISTS	VS	TUPLES
Nature	List is a collection of different types of data items that is ordered and changeable i.e, Mutable In Nature		Tuples is a collection of different types of data items that is ordered and changeable i.e, Immutable In Nature
Declaration	The list items are surrounded in Square Brackets []		The tuple items are surrounded in Parenthesis () which is optional
Preference	If the Content is not fixed and keep on changing then we should go for List.		If the content is fixed and never changes then we should go for Tuple.
Space	List occupies more space than tuples		Tuples occupies less space as compared to lists
Time	Lists takes more time to execute than tuples		Tuples takes less time to execute
Keys for Dict	List Objects can not used as Keys for Dictionaries because Keys should be Hashable and Immutable.		Tuple Objects can be used as Keys for Dictionaries because Keys should be Hashable and Immutable
Example	L = [1,2,3,4]		T = (1,2,3,4) or T = 1,2,3,4



When to use a Tuple instead of List



- Program **execution is faster** when manipulating a tuple than it is for the equivalent list. (This is probably not going to be noticeable when the list or tuple is small.)
- Sometimes you don't want data to be modified. If the values in the collection are meant to **remain constant** for the life of the program, using a tuple instead of a list guards against accidental modification.
- There is another Python data type that you will encounter shortly called a dictionary, which requires as one of its components a **key** that is of an immutable type. A tuple can be used for this purpose, whereas a list can't be.

Lesson Pencils Bhupinder

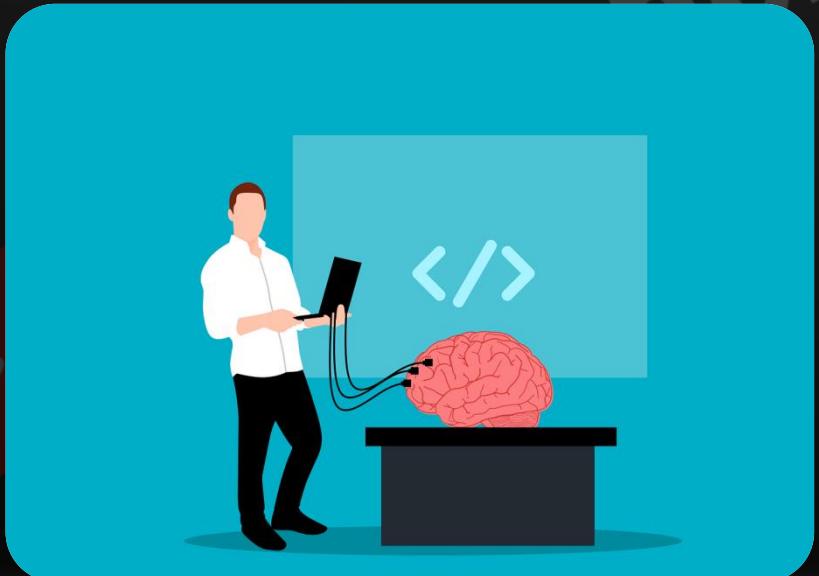
LIST vs **TUPLE**



Tuple Space / Size



Some of you are thinking that how tuples are efficient in space or how it is faster than list then here is the proof we are doing programmatically to show this .





Tuple Space / Size



```
import sys

list1 = [1,2,3,4,5]
tuple1 = (1,2,3,4,5)

print("List V/s Tuples")

print("Size of list = ",sys.getsizeof(list1))
print("Size of tuple = ",sys.getsizeof(tuple1))
```



Tuple Execution Time



```
import timeit

ListT = timeit.timeit(stmt="[1,2,3,4]",number=2000000)
TupleT = timeit.timeit(stmt="(1,2,3,4)",number=2000000)

print("List V/s Tuples")

print("Time taken by list = ",ListT)
print("Time taken by tuple = ",TupleT)
```



Programming Lab - 26



- 71.** WAP to convert tuples into list and vice versa

- 72.** WAP to check if the element is present in tuple or not

- 73.** WAP to update a value in tuple



ACTIVITY TIME



What is the value of A

```
A = tuple("Python")
```

```
print(A)
```

- a (python)
- b ("Python")
- c ('P' , 'y' , 't' , 'h' , 'o' , 'n')
- d None of the above





DICTIONARY





DICTIONARY



Like a real-life dictionary has words and meanings, Python dictionaries have **keys** and **values**. They are an important data structure in Python and we will learn how to create, access, manipulate and use them.

If we want to represent a group of objects as **key-value pairs** then we should go for Dictionary.

```
Dictionary = {  
    Key : value,  
    Key : value,  
    Key : value  
}
```



{ DICTIONARY }



Dictionaries are in **curly brackets**.

Key-value pairs are separated by **commas** and keys and values are separated by **colons**. Keys in dictionaries are unique and immutable.

Unlike other data structures, dictionaries hold two items in pairs instead of a single item. This is like a real-life dictionary. You can search for a value if you know the key. One key cannot have two values.



{ DICTIONARY }



Example

Think of a phone book. You have names of people and the phone numbers associated to those people. A dictionary is exactly the data structure you need if you want to implement a phone book.

A phone book is nothing more than a collection of name: phone number pairs.

Similarly, a dictionary is a collection of key: value pairs.



Dictionary



{ DICTIONARY }



Points to Remember

- ❑ Duplicate keys are not allowed but values can be duplicated.
- ❑ Heterogeneous objects are allowed for both key and values.
- ❑ insertion order is not preserved
- ❑ Dictionaries are mutable
- ❑ Dictionaries are dynamic
- ❑ indexing and slicing concepts are not applicable

Note: In C++ and Java Dictionaries are known as "Map"



IMPORTANT



DICTIONARY

Creating Dictionaries //

Accessing Items //

Adding Items //

Removing Items //

Changing Values //

Copying Dictionaries //



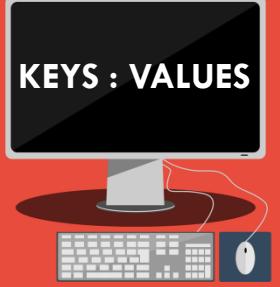
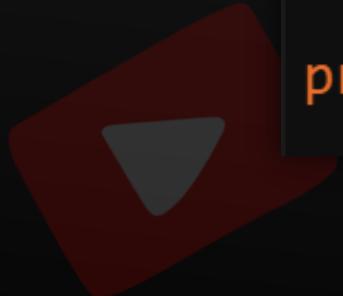
CREATION OF DICTIONARY



Create an empty Dictionary

```
d = {}
```

```
print(type(d))
```





CREATION OF DICTIONARY



We can add entries as follows

```
d = {}  
d['Name']="Alia"  
d['Age']=50  
  
print(d)  
-> {'Name': 'Alia', 'Age': 50}
```



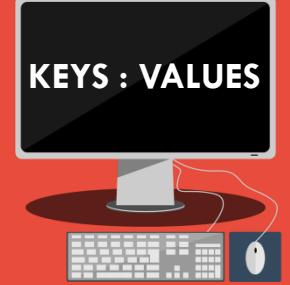


CREATION OF DICTIONARY



If we know data in advance then we can create dictionary as follows

```
d = {'Name': "Alia", 'Age': 50}  
  
print(d)  
-> {'Name': 'Alia', 'Age': 50}
```



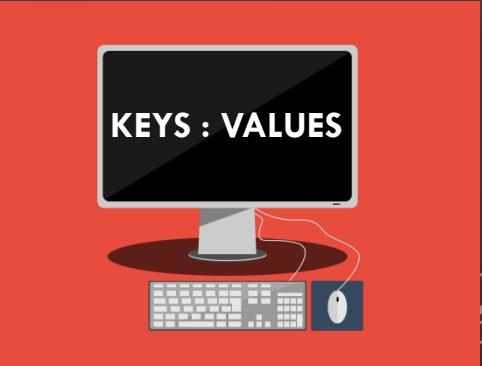


CREATION OF DICTIONARY



If we know data in advance then we can create dictionary as follows

```
capitals = {'India':'Delhi', 'Pakistan':'Islamaba", "Uganda":"Kampala"}  
print(capitals)
```



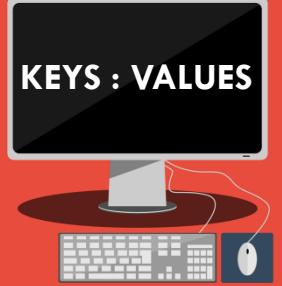


CREATION OF DICTIONARY



Dictionary with Mixed Keys

```
dictionary = {1:"Tapsee", 'two':50000,"A":56000}  
  
print(dictionary)
```





ACCESS ELEMENTS OF DICTIONARY



When working with a python dictionary, it is sometimes necessary to access dictionary data such as keys, values and items (key-value pairs). Access to this data can take place in various ways.

The methods below can be applied to access a certain value from a dictionary.



Keys	Values
'A'	apple
'B'	Ball
'C'	Cat



ACCESS ELEMENTS OF DICTIONARY



Using Square brackets

To access dictionary elements, you can use square brackets along with the key to obtain its value.

```
dic = {1:"Tapsee",'two':500.60,"A":56000}
```

```
print(dic[1])      -> Tapsee
```

```
print(dic['two']) -> 500.6
```

Keys	Values
'A'	apple
'B'	Ball
'C'	Cat



ACCESS ELEMENTS OF DICTIONARY



Using Square brackets

If the given key does not exist in the dictionary, Python raises a Key Error exception.

```
dic = {1:"Tapsee", 'two':500.60, "A":56000}
```

```
print(dic[0])      -> KeyError
```

```
print(dic["Tapsee"]) -> KeyError
```

Keys	Values
'A'	apple
'B'	Ball
'C'	Cat



ACTIVITY TIME



1. WAP to enter name and percentage marks in a dictionary and display information on the screen





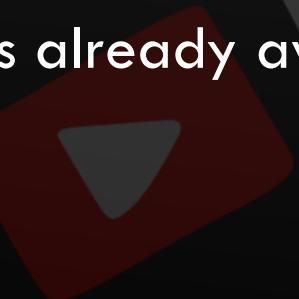
TO UPDATE DICTIONARY



```
dict[key] = value
```

If the key is not available then a new entry will be added to the dictionary with the specified key-value pair

If the key is already available then old value will be replaced with new value.





TO UPDATE DICTIONARY



```
dict = {"Name":"Tapsee"}  
  
# Key Age is not present in dict  
dict["Age"]=60  
print(dict)  
  
# Key name is already present it will update  
dict["Name"]="Riya Chakraborty"  
print(dict)
```



DELETE IN DICTIONARY



You can delete an entire dictionary. Also, unlike a tuple, a Python dictionary is mutable. So you can also delete a part of it.

1. Deleting an **entire Python dictionary**
2. Deleting a **single key-value pair**





DELETE IN DICTIONARY



1. Deleting an entire Python dictionary

To delete the whole Python dict, simply use its name after the keyword 'del'.

```
Dic = {"Name": "Tapsee", "Gender": "Female"}  
  
del Dic    --> dic deleted  
print(Dic) --> NameError
```

Keys	Values
'Name'	Tapsee
'Gender'	Female





DELETE IN DICTIONARY



2. Deleting a single key-value pair

To delete just one key-value pair, use the keyword 'del' with the key of the pair to delete.

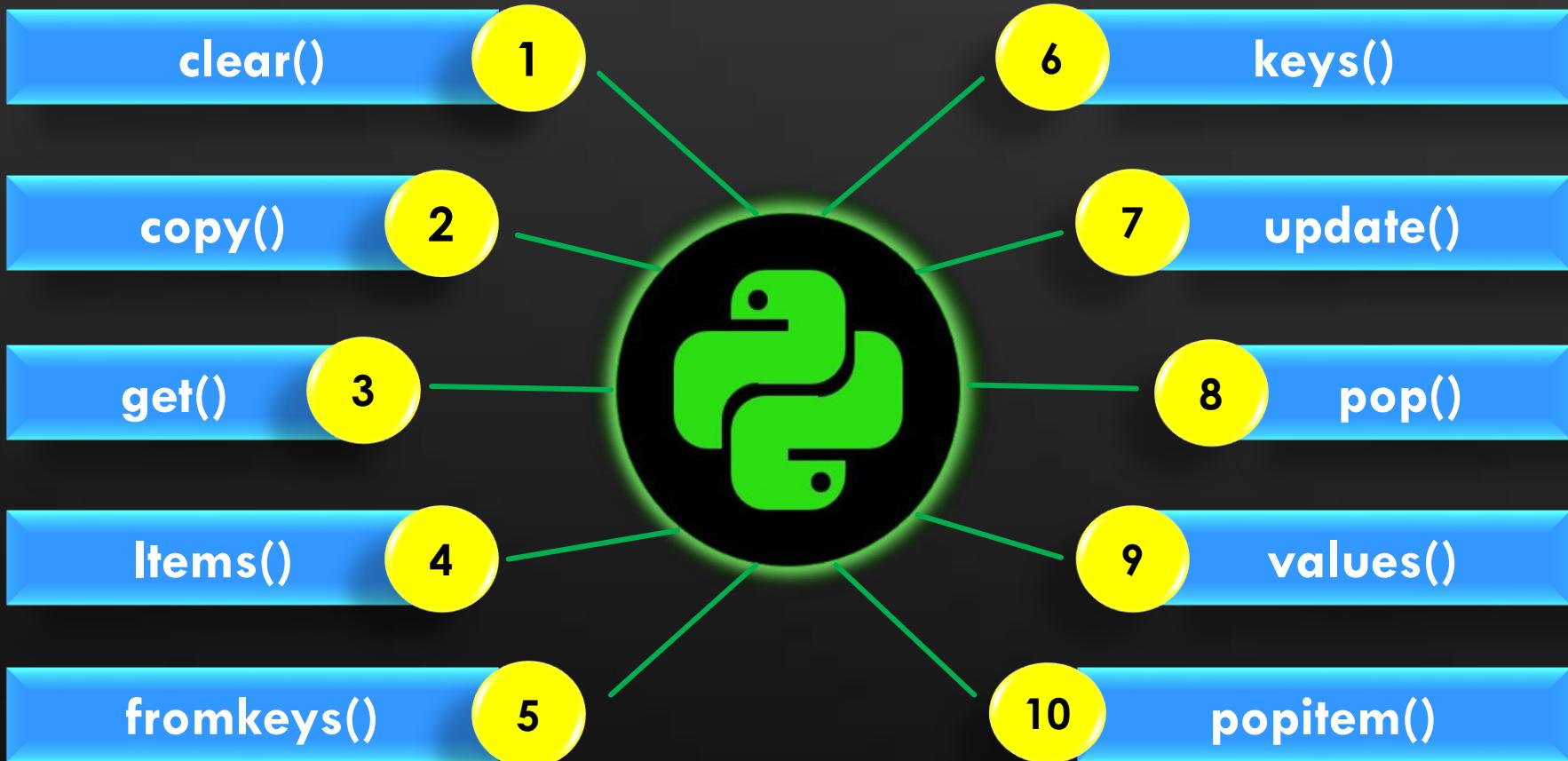
```
Dic = {"Name": "Tapsee", "Gender": "Female"}  
  
del Dic["Gender"]      --> Name pair deleted  
print(Dic)            --> Printing
```

Keys	Values
'Name'	Tapsee
'Gender'	Female





DICTIONARY METHODS





1. CLEAR()



1

clear()

clear() removes all the key-value pairs of a dictionary.

```
d = {"Name": "Tapsee", "Age": 50}
```

```
d.clear() --> Clears all K-V Pairs
```

```
print(d) --> Prints Empty Dictionary
```



2. COPY()



2

copy()

To create exactly **duplicate** dictionary(cloned copy)

```
d1 = {"Name": "Tapsee", "Age": 50}
```

```
d2=d1.copy()  
print(d1)  
print(d2)
```



3. GET()



To get the value associated with the key

3

get()

`d.get(key)`

If the key is available then returns the corresponding value otherwise returns **None**. It wont raise any error.

`d.get(key,defaultvalue)`

If the key is available then returns the corresponding value otherwise returns **default value**.



3. GET()



```
d1 = {"Name": "Megha", "Age": 20}

print(d1.get("Name"))
print(d1.get("Salary"))
print(d1.get("Salary", "Not Present"))
```



4. ITEMS()



4

Items()

It returns list of tuples representing key-value pairs.

```
d = {"Name": "Megha", "Age": 20}  
print(d.items())
```

```
# Another option to Traverse  
for k,v in d.items():  
    print(k, " = ", v)
```



5. FROMKEYS()



This method takes keys from an iterable and creates a new dictionary from it.

5

fromkeys()

```
keys = ['key1', 'key2', 'key3']

d = d.fromkeys(keys,0)

print(d)
```



6. KEYS()



It returns all keys associated with dictionary

6

keys()

```
d = {"Name": "Megha", "Age": 20}  
print(d.keys())
```

```
# Another option to Traverse  
for k in d.keys():  
    print(k)
```



7. UPDATE()



All items present in the dictionary d2 will be added to dictionary d1

7

update()

```
d1 = {"Name": "Noora Fatehi", "Age": 20}  
d2 = {"Profession": "Actress", "Networth": 10000000}  
  
d1.update(d2) # d1 = d1 + d2  
print(d1)
```



8. POP()



8

pop()

d1.pop(key)

It removes the entry associated with the specified key and returns the corresponding value

If the specified key is not available then we will get **KeyError**

```
d1 = {"Name": "Megha", "Age": 20}

print(d1.pop("Name")) --> Removed Name
print(d1.pop("name")) --> KeyError
```



9. VALUES()



9

values()

It returns **all values** associated with the dictionary

```
d1 = {"Name":"Megha","Age":20}  
print(d1.values())
```

```
# Another option to Traverse  
for v in d1.values():  
    print(v)
```



10. POPITEM()



This removes the last added key-value pair from a dictionary and returns it.

10

popitem()

```
d1 = {"Name": "Megha", "Age": 20}  
print(d1)  
print(d1.popitem())  
print(d1)
```

It returns **KeyError** if Dictionary is Empty



DICTIONARY



```
D = {"0":"0","1":"1","2":"2"}  
print(D[0])
```

a 0

b "0"

c Syntax Error

d Key Error





Programming Lab - 27



- 74.** WAP to Input keys and values and update

- 75.** WAP to Input keys and values and delete

- 76.** WAP to Input keys and values and search



Programming Lab - 28



- 77.** WAP to Input keys and values from the keyboard and print the sum of values ?

- 78.** WAP to find number of occurrences of each letter present in the given string ?



SETS



Sets

PYTHON

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

Note: Set items are unchangeable, but you can remove items and add new items.



SET



An item appears only once in a set, no matter how many times it is added.

Set

{1,2,3,4}



SETS



- 1 Duplicates are not allowed.
- 2 Insertion order is not preserved . But we can sort the elements.
- 3 Indexing and slicing not allowed for the set.
- 4 Heterogeneous elements are allowed.
- 5 We can represent set elements within curly braces and with comma separation
- 6 We can apply mathematical operations like union, intersection, difference etc on set objects.



CREATION OF SET



1. Directly with **Curly Braces** if we know the elements

```
S = {1,2,3,4,5}
```

```
print(type(S))
```





CREATION OF SET



2. We can create set objects by using `set()` function

`S = set(any sequence)`

```
L = [1,2,3,4,5,6,7,8,9,10]
```

```
S = set(L)
```

```
print(S)
```

```
print(type(S))
```



SET



Set = { }

print(type(Set))

- a list :(
- b set :(
- c frozenset :(
- d dict :)





CONCLUSION



If your answer is Set then your situation may be like Rajpal Yadav





CONCLUSION





CREATION OF SET



Note: While creating empty set we have to take special care.
Compulsory we should use **set()** function.

S = {} ==> It is treated as dictionary but not empty set.



```
S = {}

print(type(S)) --> dict
```

Set

PYTHON



IMPORTANT FUNCTIONS OF SETS



Important Functions of Sets

1	<code>add()</code>	Adds item to the set
2	<code>update()</code>	To add multiple items to the set.
3	<code>copy()</code>	Returns copy of the set. It is cloned object.
4	<code>pop()</code>	It removes and returns some random element from the set.
5	<code>remove()</code>	It removes the specified element from the set. (<code>KeyError</code> if Not present)
6	<code>discard()</code>	It removes the specified element from the set.
7	<code>clear()</code>	To remove all elements from the Set.



1. ADD()



Adds item to the set

```
s = {10,20,30}  
s.add(100)  
  
print(s) --> {100,10,20,30}
```



1

add()



2. UPDATE()



To add multiple items to the set.

Arguments are not individual elements and these are Iterable objects like List , range etc. All elements present in the given Iterable objects will be added to the set.

2

update()

```
s = {10,20,30}  
lst = [1,2,4]  
s.update(lst)  
print(s)  
s.update(range(5))  
print(s)
```



3. COPY()



Returns copy of the set.
It is cloned object.

```
s1 = {10,20,30}
```

```
s2=s1.copy()
```

```
print(s2)  
print(s1)
```



3

copy()



4. POP()



It removes and returns some random element from the set.

4

pop()

```
s = {10,20,30}  
  
print(s)  
print(s.pop())  
print(s)
```





5. REMOVE()



5 remove()

It removes specified element from the set.

If the specified element not present in the Set then we will get
KeyError

```
s = {10,20,30,40}

print(s.remove(10)) --> will remove 10
print(s)
print(s.remove(100)) --> KeyError
```



6. DISCARD()



It removes the specified element from the set.

If the specified element not present in the set then we won't get any error.

```
s = {10,20,30,40}  
  
s.discard(10)  --> will remove 10  
print(s)  
s.discard(100) --> No KeyError (None)  
print(s)
```

6

discard()



7. CLEAR()



To remove all elements from the Set.

7

clear()

```
s = {10,20,30,40}
```

```
s.clear() --> Clears all Data
```

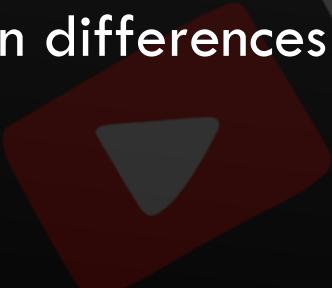
```
print(s) --> Empty Set
```



Lets Discuss



- Q. What is the difference between `remove()` and `discard()` functions in Set?
- Q. Explain differences between `pop()` and `remove()` functions in Set?





ADD() VS UPDATE()



We can use `add()` to add **individual** item to the Set
where as we can use `update()` function to add **multiple** items to Set.

`add()` function can take only **one argument**
where as `update()` function can take **any number** of arguments but
all arguments should be iterable objects

`add` vs `update`



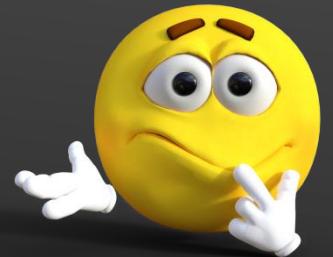
SETS

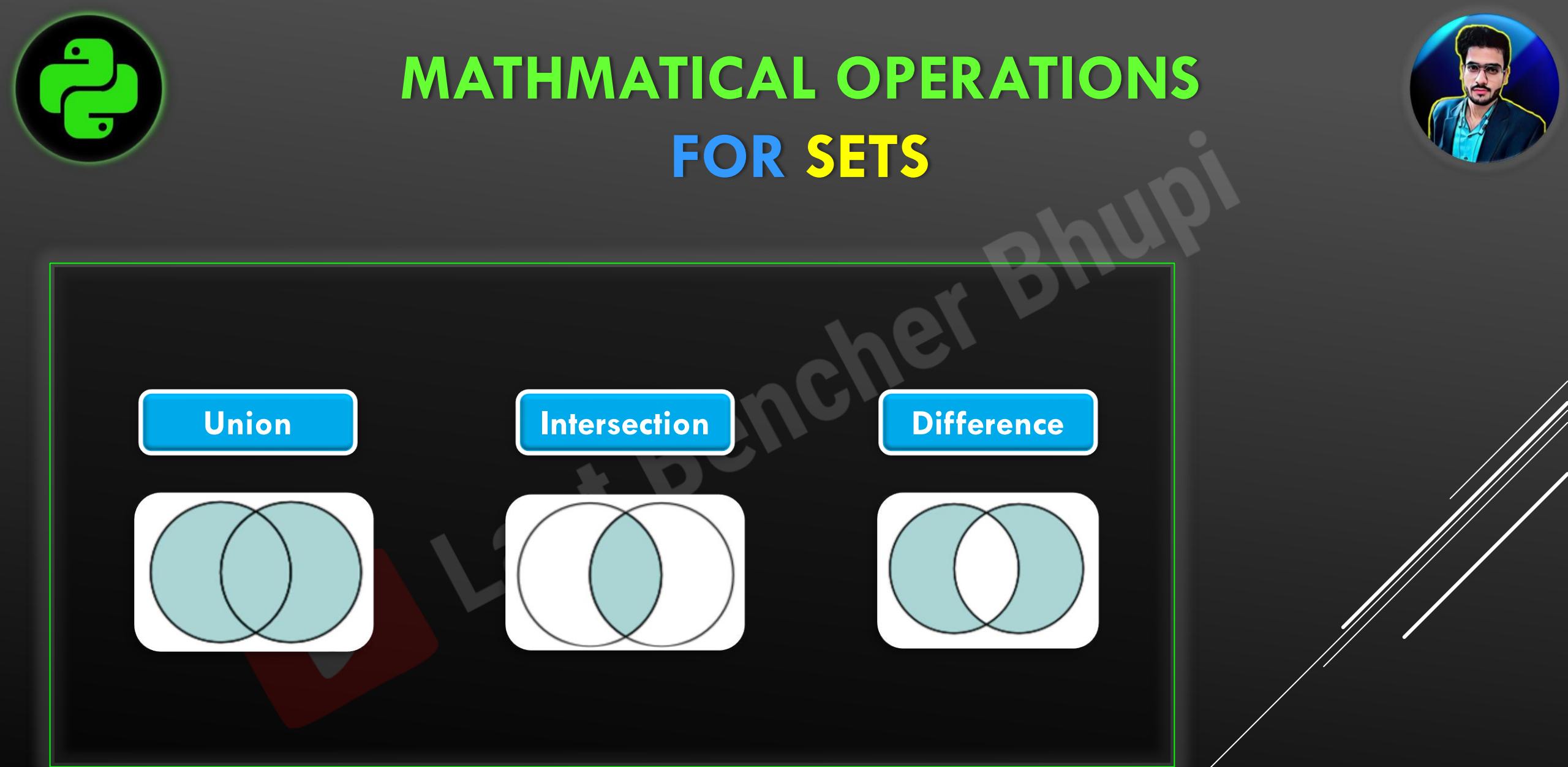


Which of these is valid for set s

`s = set()`

- a `s.add(10)`
- b `s.add(10,20,30)`
- c `s.update(10)`
- d `s.update(range(1,10,2),range(0,10,2))`







MATHMATICAL OPERATIONS FOR SETS



UNION

s1.union(s2) ==> We can use this function to return all elements present in both sets

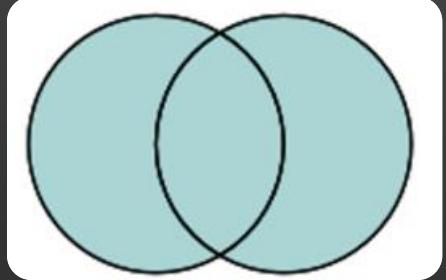
s1.union(s2) or s1 | s2

```
s1 = {1,2,3,4}
```

```
s2 = {5,6,7,8}
```

```
print(s1.union(s2))
```

```
print(s1|s2)
```





MATHMATICAL OPERATIONS FOR SETS



INTERSECTION

`s1.intersection(s2)` or `s1&s2`

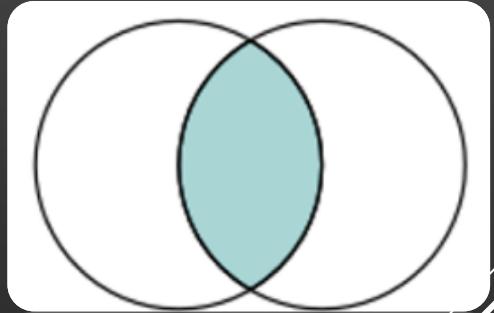
Returns common elements present in both s1 and s2

```
s1 = {1,2,3,4}
```

```
s2 = {2,6,7,1}
```

```
print(s1.intersection(s2))
```

```
print(s1&s2)
```





MATHMATICAL OPERATIONS FOR SETS

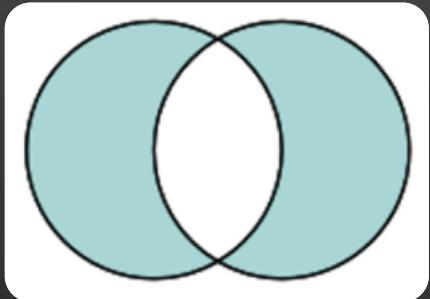


DIFFERENCE

`x.difference(y)` or `x-y`

returns the elements present in x but not in y

```
s1 = {1,2,3,4}  
s2 = {2,6,7,1}  
  
print(s1.difference(s2))  
print(s1-s2)
```





Programming Lab - 29



- 79.** WAP to eliminate duplicates present in the list?

- 80.** WAP to print different vowels present in the given word

Note : By using sets()



FROZEN SET



FROZEN SET

IN

PYTHON



FROZEN SET



It is exactly **same as set** except that it is **immutable**.
Hence we cannot use add or remove functions.

```
frozenset([1,2,3,4])
```



iterable



FROZEN SET



Frozen set is just an **immutable version** of a Python set object. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

Due to this, frozen sets can be used as keys in Dictionary or as elements of another set.



HOW TO CREATE FROZEN SET



```
s = {10,20,30,40}
```

```
fs = frozenset(s)
```

This fs is freezed



HOW TO CREATE FROZEN SET



```
l = [1,2,3,4,5]
fs = frozenset(l)

print(type(fs))

fs.add(10)    -->AttributeError
fs.remove(5)  -->AttributeError
```



Programming Lab - 30



81. WAP to calculate Number of Digits in a Number

82. WAP to calculate Cube of Each Digit in a Number

for Example Input = 123

output = $3^3 = 27$

$2^3 = 8$

$1^3 = 1$



Programming Lab - 31



83. WAP to calculate Power of Each Digit in a Number Where value of Power = Number of Digits in a Number

for Example Input = 1234 here, Number of Digits is 4

That means Power = 4,

So the Preferred output will be

$$4^4 = 256$$

$$3^4 = 81$$

$$2^4 = 16$$

$$1^4 = 1$$



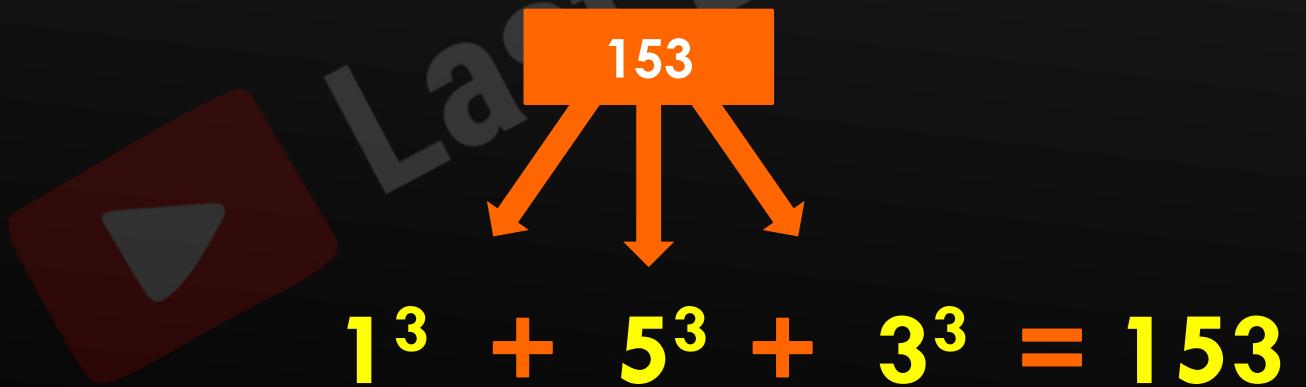


Programming Lab - 32



Now Based on Previous 3 Programs you can able to do the Next one
Which is

84. To Check Whether a Number is Armstrong or not.





Programming Lab - 33



85. To Check Whether a Number is Perfect or not.

$$6 = 1 + 2 + 3$$

Perfect Number

Proper Divisors



SUMMARY



Data Type	Description	Nature	Example
Int	We can use to represent the whole / integral numbers	Immutable	a = 5
Float	We can use to represent the decimal / floating point numbers	Immutable	b = 10.5
Complex	We can use to represent the complex numbers	Immutable	c = 10 + 5j
Bool	We can use to represent the logical values(Only allowed values are True and False)	Immutable	flag = True flag = False
Str	To represent sequence of Characters	Immutable	s = "Bhupi"
Bytes	To represent a sequence of byte values from 0-255	Immutable	list = [1,2,3,4] b = bytes(list)



SUMMARY



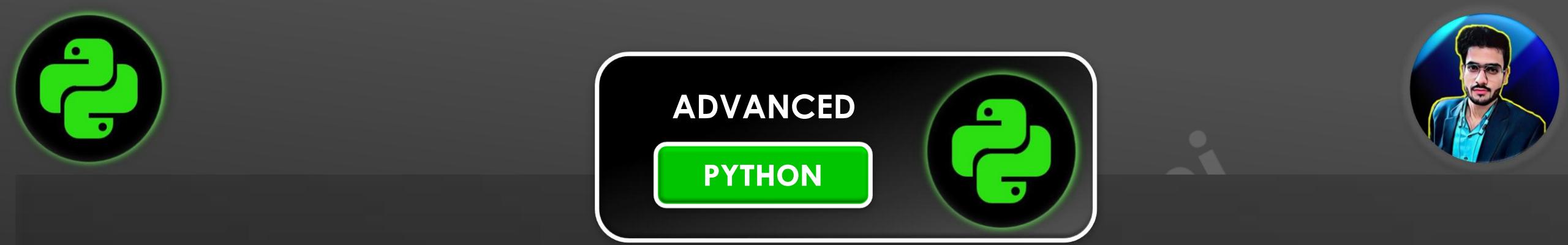
Data Type	Description	Nature	Example
ByteArray	To represent a sequence of byte values from 0-255	Mutable	<code>ba = bytearray([1,2,3])</code>
Range	To represent a range of values	Immutable	<code>r = range(10)</code> <code>r1 = range(0,10)</code> <code>r2 = range(0,10,2)</code>
List	To represent an ordered collection of objects	Mutable	<code>L = [10,11,12]</code>
Tuple	To represent an ordered collections of objects	Immutable	<code>T = (10,11,12)</code>
Dict	To represent a group of key value pairs	Mutable	<code>d={101:'Bhupi', 102:'Nimrat'}</code>
Set	To represent an unordered collection of unique objects	Mutable	<code>s={1,2,3,4,5,6}</code>
FrozenSet	To represent an unordered collection of unique objects	Immutable	<code>fs=frozenset(s)</code>



We have completed the Basics of python and Basic Data Structure in of which is the foundation of learning python.

Now as the next step we are going to build Multistory Building to that foundation

All the Best !



NEXT TOPIC

- Functions – Filter , Map , Reduce , Lambda
- Recursion
- Modules
- Packages
- File Handling
- Exception Handling
- Object Oriented Python
- Decorator & Generator
- Deep dive in String (Logic Building)





Syllabus

PYTHON

- Basic Fundamentals of Python
- Data types
- Int float list tuple dict set frozenset
- Flow Control (Loops , if else, break , pass)
- Type Casting
- Immutability vs Mutability
- String
- Slicing
- Interview Preparation
- Functions – Filter , Map , Reduce , Lambda
- Recursion
- Modules
- Packages
- File Handling
- Exception Handling
- Object Oriented Python
- Decorator & Generator
- Deep dive in String (Logic Building)



THANK YOU



```
print("Thank You")
```

