Big Data Hadoop and Spark Developer

Introduction to Python for Apache Spark

Learning Objectives

By the end of this lesson, you will be able to:

- Define Python and its features
- Identify the benefits of Python
- List the frequently used libraries in Python
- Execute a Python script in different modes
- Describe variables, operators, and conditional and looping statements in Python

Introduction to Python

What Is Python?

- Python is a general-purpose, object-oriented, and high-level interpreted programming language.
- It was designed by Guido van Rossum in 1991.
- It is used for web development, scripting, web scraping, and data analytics.

Features of Python

The features of Python are:

01 Easy to learn

02 Interactive mode

03 Portable

04 Free and open source

05 Extendable

06 Easy integration

Features of Python

01

Easy to learn: Python is an easy-to-learn programming language with a straightforward structure and a well-defined syntax.

02

Interactive mode: Python has support for an interactive mode that allows interactive testing and debugging of code snippets.

03

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all the platforms.

Features of Python

04

Free and open source: Python is an open-source programming language that is easy to learn and is freely available.

05

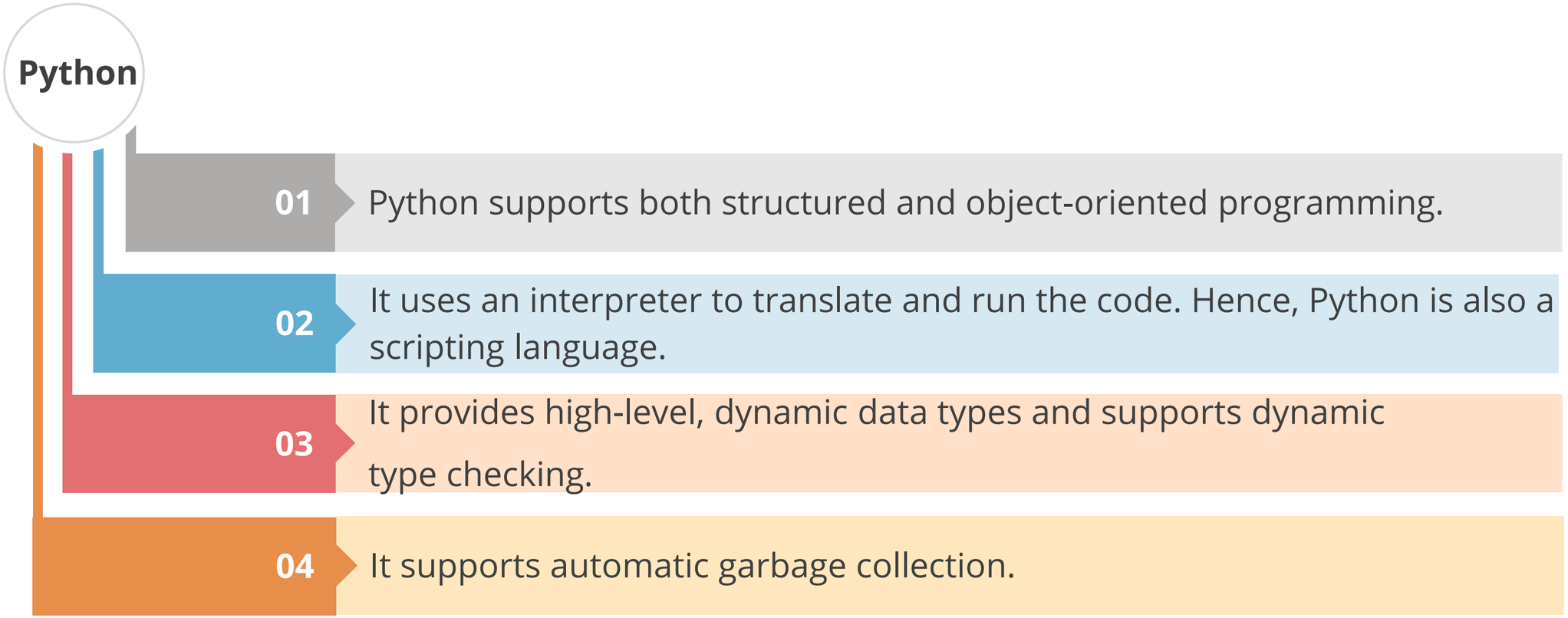
Extendable: Python allows the interpreter to use low-level modules. Adding or customizing these modules can help the programmers increase the efficiency of their tools.

06

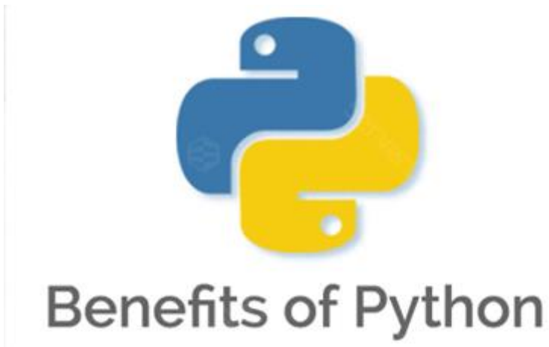
Easy integration: Python can be easily integrated with other programming languages, such as C, C++, and Java.

Characteristics of Python

The important characteristics of Python are:



Benefits of Python



1. **Beginner's language:** Python is great for beginners as it allows the user to create programs from simple text processing**.**
2. **Interpreted**: Python is processed at runtime by the interpreter and there is no need to compile the code.
3. **Interactive:** It is possible to directly interact with the interpreter and write programs.
4. **Object-oriented:** This programming style encapsulates code inside objects.

Modes of Python

Python: Modes

Python can work in the following two modes:

Batch script mode	• In this mode, all statements of a particular program can be written in one file. • Then, the entire file is executed in one go.
Interpreter mode	• In this mode it is possible to submit each line of code separately and work on the code as a whole. • Python includes an interpreter that can be used for this purpose.

Modes: Batch Scr̄ipt Mode Steps

Example: counter.py

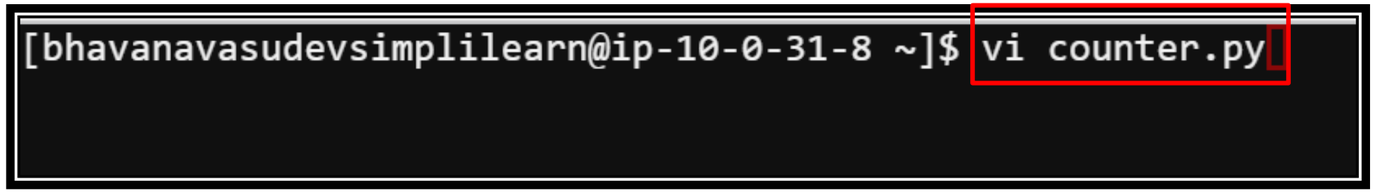
```
//Creating a new python file command//
vi counter.py
counter = 0
while counter < 5:
  print ('Counter Value = ', counter)
  counter = counter + 1

//Run the python file command//
python counter.py

Output:
('Counter Value = ', 0)
('Counter Value = ', 1)
('Counter Value = ', 2)
('Counter Value = ', 3)
('Counter Value = ', 4)
```

Steps to perform:

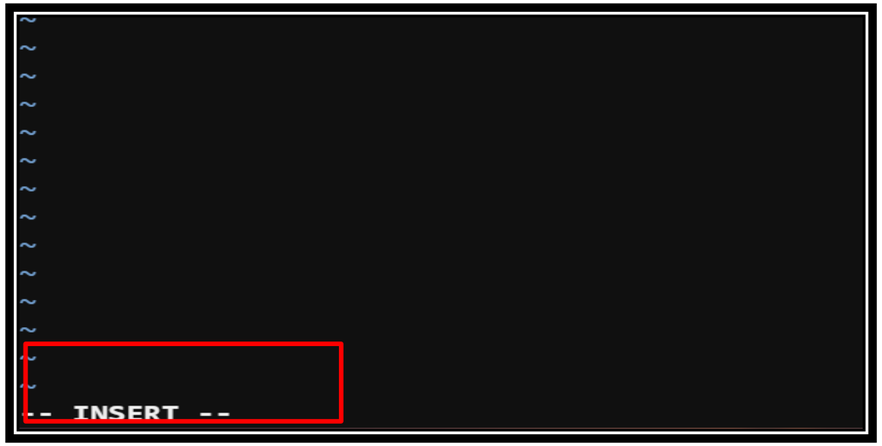
- Log in to the **Webconsole** and create a Python file with the **.py** extension
- Click on the letter ***"I"*** to enter the insert mode
- Type an expression or a statement
- To save and exit batch script mode, click on the **"esc"** key and type **":wq"**
- Run the Python file to check the output

Batch Script Mode: Steps

Step 1: Log in to the **Terminal** and run the command given below to create a new Python file



Step 2: Click on **“i”** on your keyboard to enter the insert mode



Batch Script Mode: Steps

Step 3: Enter a Python code

```
counter = 0
while counter < 5:
    print ('Counter Value = ', counter)
    counter = counter + 1
```

Batch Script Mode: Steps

Step 4: To save and exit, click on the **“esc”** key and type **“:wq”**

Batch Script Mode: Steps

Step 5: Run the Python script to view the output

```
[bhavanavasudevsimplilearn@ip-10-0-31-8 ~]$ python counter.py
('Counter Value = ', 0)
('Counter Value = ', 1)
('Counter Value = ', 2)
('Counter Value = ', 3)
('Counter Value = ', 4)
```

Modes: Interpreter Mode Steps

Example: Interpreter mode

```
python3
>> print(“hello”)
//Output//
Hello

>> print(“Simplilearn”)
//Output//
Simplilearn
```

Steps to perform:

- Open the Python shell or interpreter in **Terminal** by typing the given command

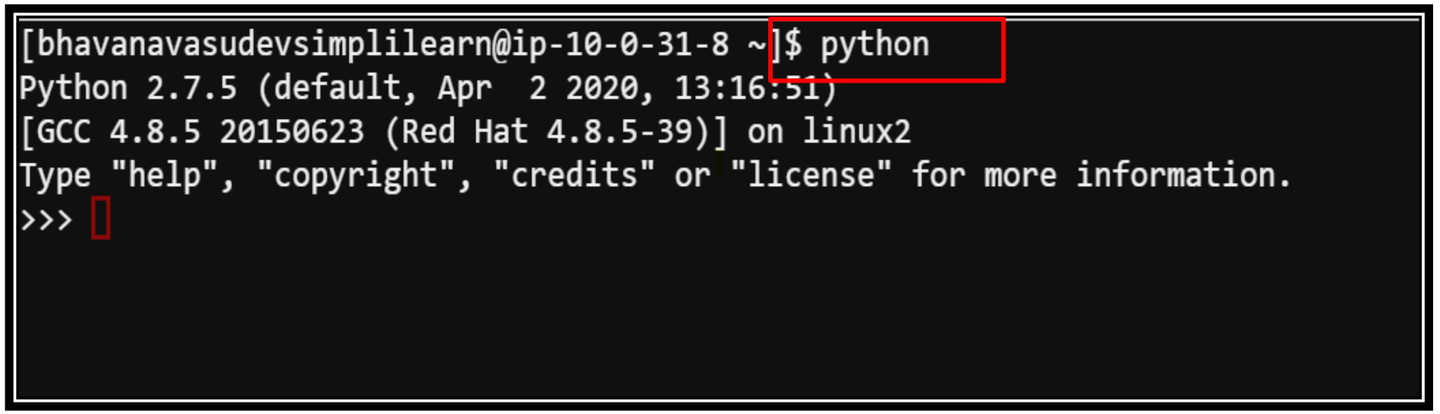
  Command:

  python

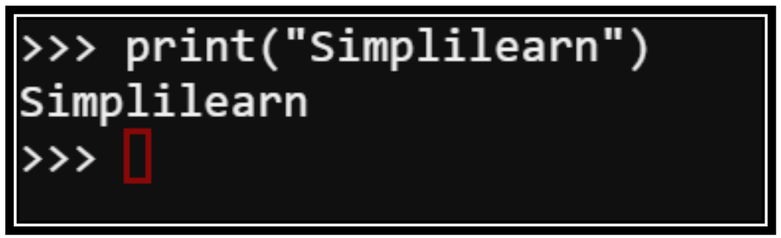
- Type an expression or a statement and click on **“enter”**
- Every expression and statement that is typed is evaluated and executed immediately

Interpreter Mode: Steps

Step 1: Log in to the **Webconsole** and run the command given below to enter interpreter mode



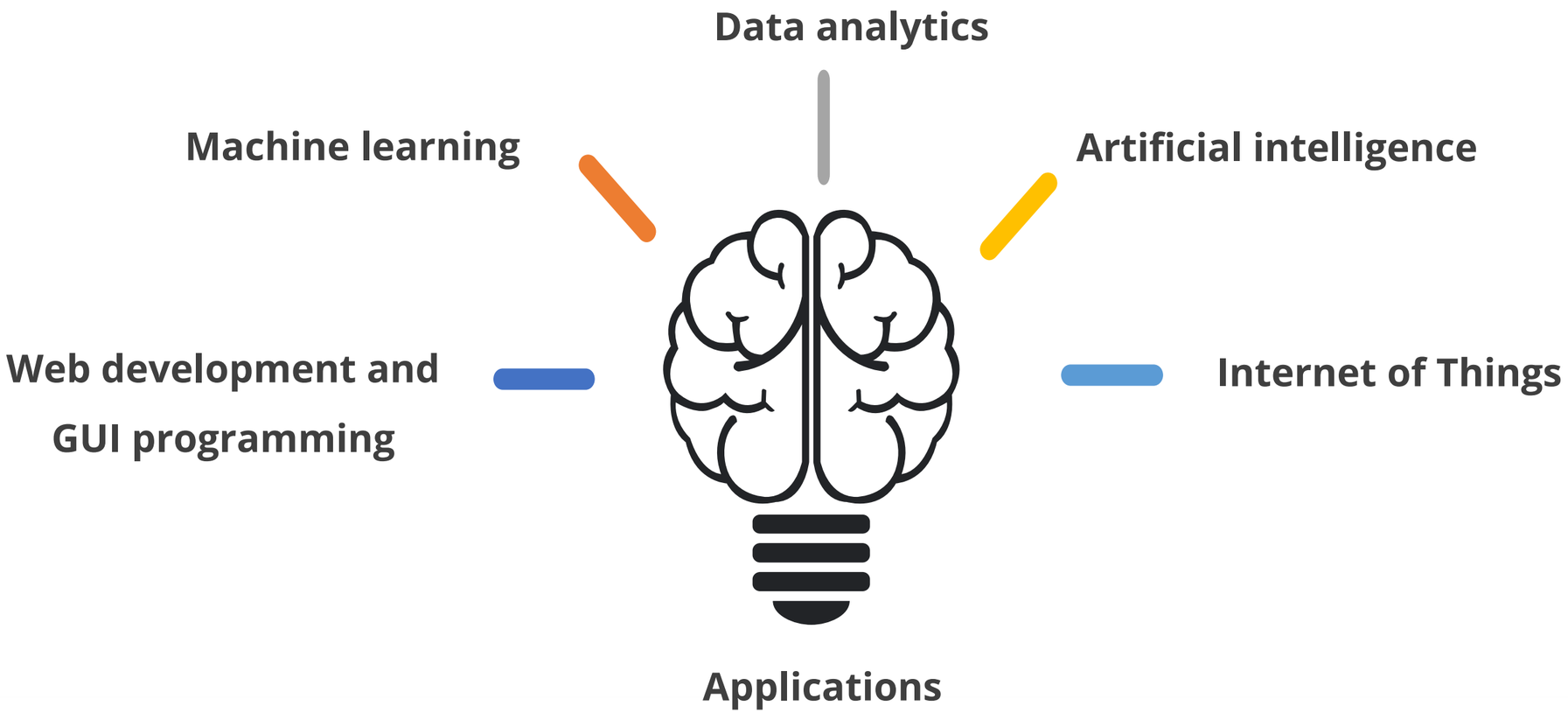
Step 2: Enter an expression and click on enter to execute it and view the output



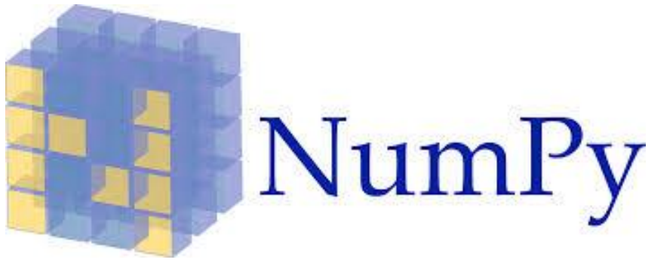
Applications of Python

Applications of Python

Python has various applications in multiple fields including:

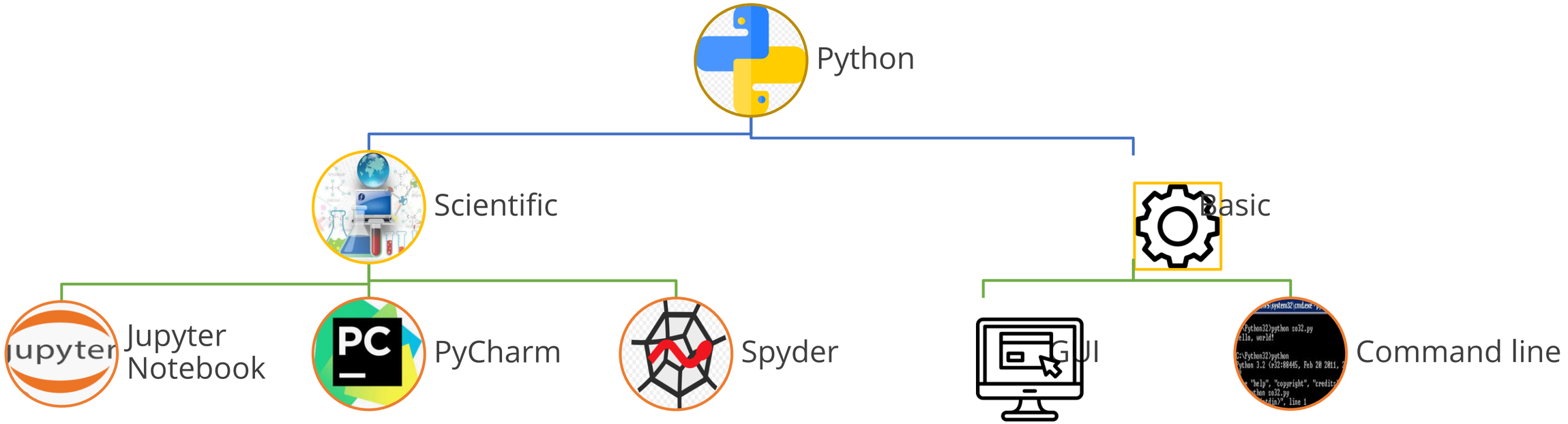


Frequently Used Libraries In Python



Python: Environments

Python has IDLE (integrated development and learning environment), which supports both basic and scientific learning environments. Codes are written in and outputs are generated from IDLE.



Variables in Python

Variables: Features

A Python variable is a reserved memory location to store values.

Features of variables

- A variable can hold a value or a string.
- A variable name should have some meaning.

  Example: age = 20

  age is the variable name that indicates the age of a person.
- A variable's name must start with a letter or an underscore.
- An assignment statement creates new variables and gives them values.

Types of Values Stored in a Variable

Python can store values of different types in a variable:

String
Stores text

Example: name.py

```
Name = “Adam”
print(Name)

Output : Adam
```

Int
Stores real numbers

Example: num.py

```
Age = 11   or  Age =
int(“11”)
print(Age)

Output : 11
```

Float
Stores decimal values

Example: pi_value.py

```
Pi_value = 3.142
print(Pi_value)

Output : 3.142
```

Types of Values Stored in a Variable

Python can store values of different types in a variable:

Boolean

Stores true or false values

Example: bool.py

```
Is_python =  True
print(Is_python)

Output :True
```

List

Stores multiple values of any type in one variable

Example: list.py

```
My_list =  [“a”,”b”,123]
print(My_list)

Output : [‘a’,’b’,123]
```

Dictionary

Stores data as a key-value pair

Example: dict.py

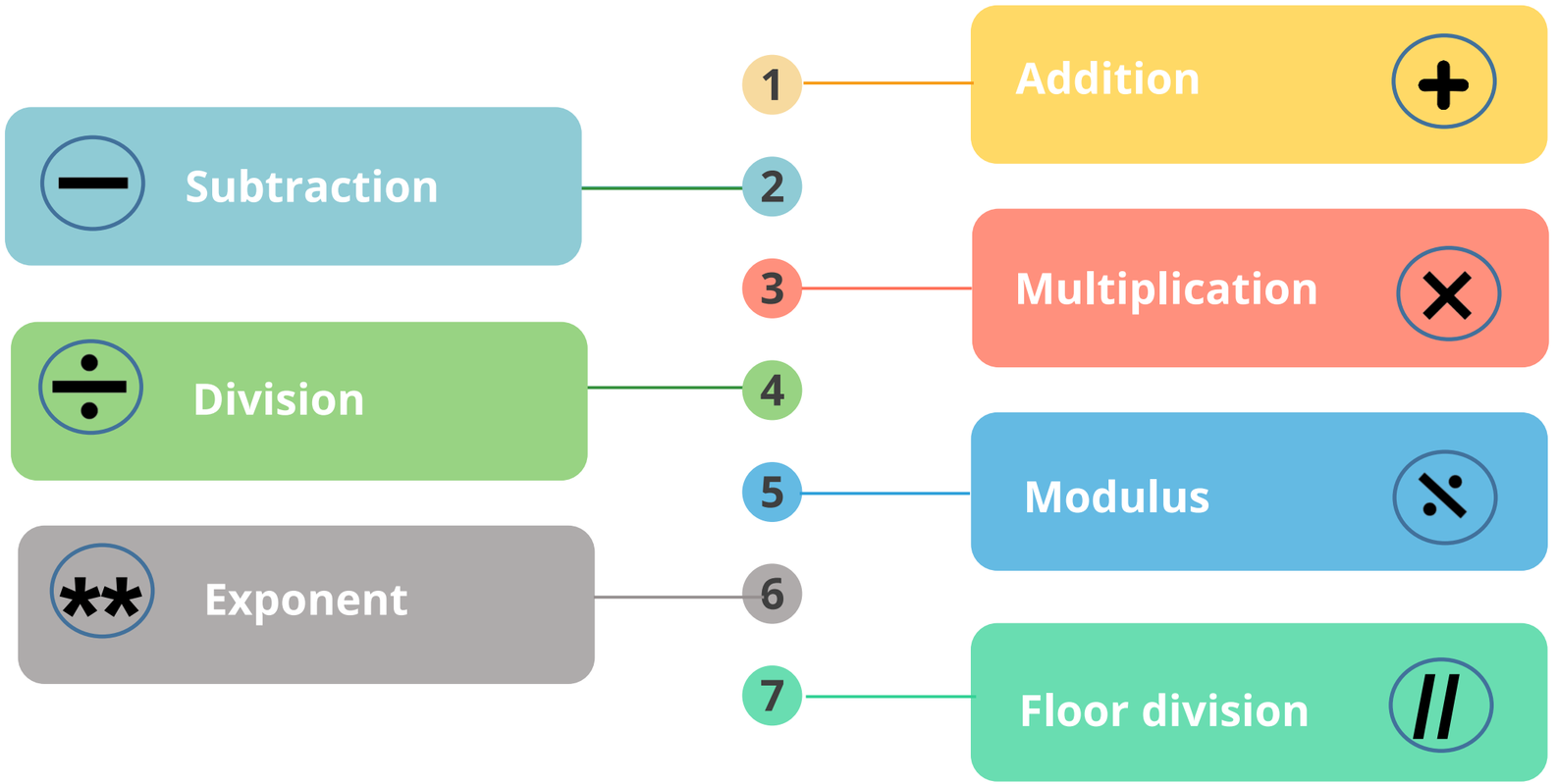
```
phone_dict={}

phone_dict['john']
={987654}

print(phone_dict)

output :

 {'john': set([987654])}
```

Operators in Python

Operators in Python

Operators are special symbols that represent computation. The values that an operator acts on are called operands.

The different operators are:



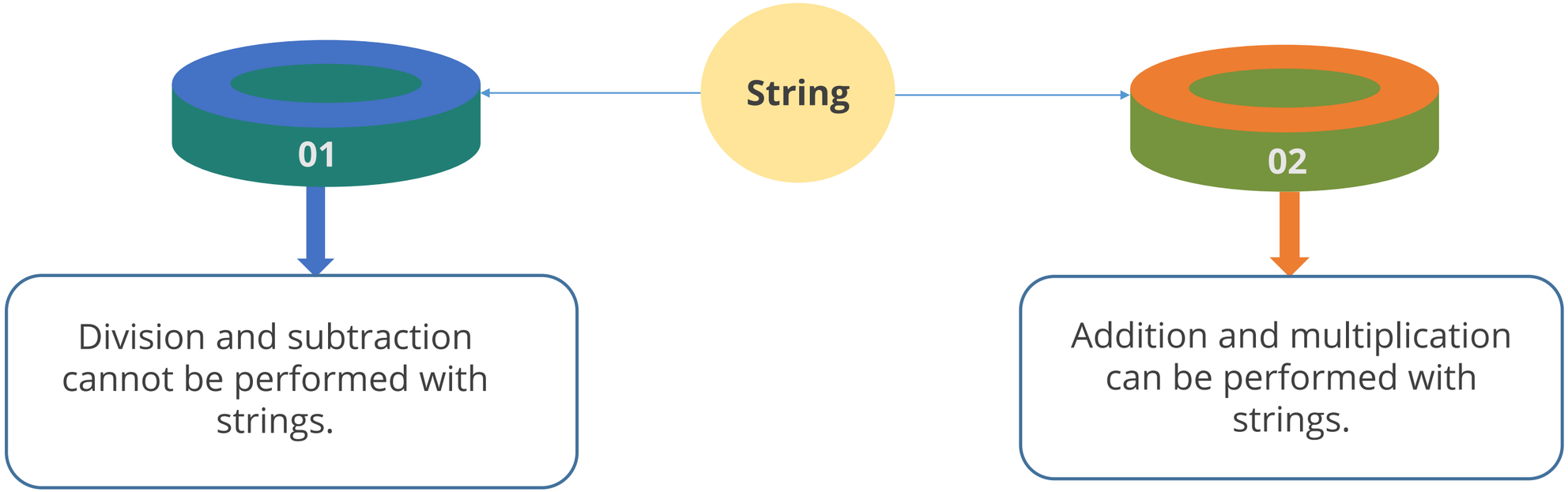
Arithmetic Operators

The following table contains the list of operators and their descriptions with examples.

Operator	Description	Example
+ Addition	Adds multiple operands and generates a result	a+b
- Subtraction	Subtracts the right-hand operand from the left-hand operand	a–b
*** Multiplication**	Multiplies values on either side of the operator	a*b
/ Division	Divides the left-hand operand by the right-hand operand	a/b
% Modulus	Divides the left-hand operand by the right-hand operand and returns a reminder	a%b
**** Exponent**	Returns an exponential (power) calculation on operators	a**b
// Floor division	Represents the division of operands where the result is the quotient, that is, the digits after the decimal point are removed	a//b

Operations on Strings

All arithmetic operations cannot be performed on strings.



Operations on Strings

The operations that can be performed on strings are explained below.

“+” is used to concatenate strings

Example: concat.py:

```
a = “Hello”
b = “World!!”
a+b
Output: “Hello World!!”
```

“*” performs repetition on strings

Example: repeat.py:

```
a = “Hello”
a*3
Output: “HelloHelloHello”
```

Order of Operands

Problem

What should be done when an expression has multiple operands?

Example:

2*3+1

Solution: 7

Precedence order is followed

Solution

1. **Parentheses:** Top precedence ((1+1)*8 = 16)
2. **Exponentiation:** Next highest precedence

   ((1+1) ** (5-2) = 8)
3. **Multiplication and Division:** Same precedence

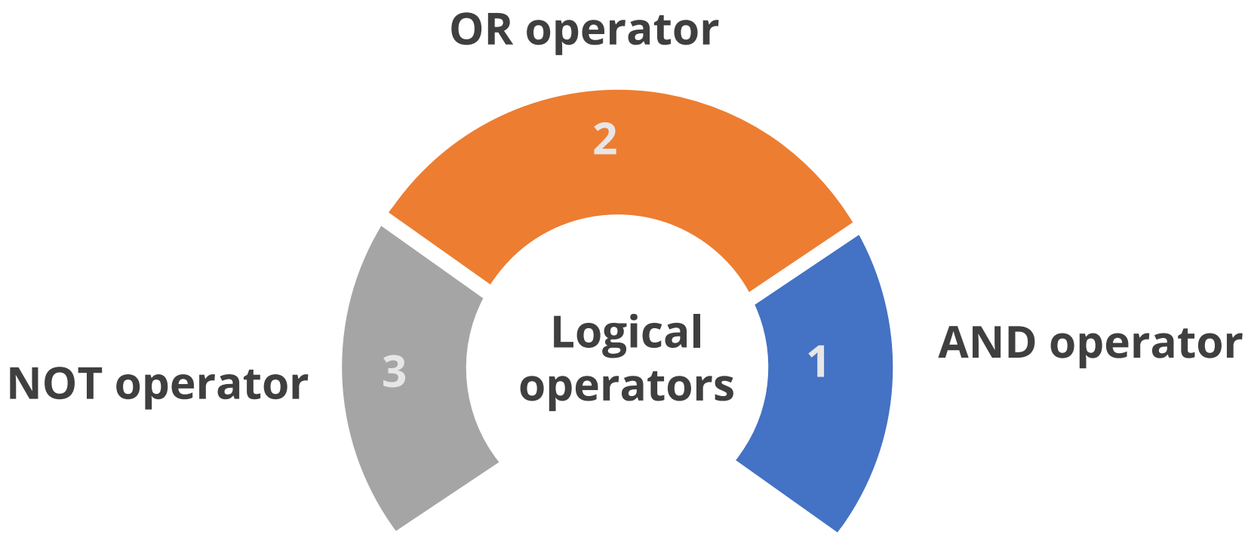
   (25/ 2 * 3.14 = 39.25)

Note:

- Operators with the same precedence are evaluated from left to right
- **PEMDAS**: Parentheses, exponents, multiplication, division addition, and subtraction

Logical Operators in Python

Logical operators are used with Boolean expressions or variables.



Logical Operators: AND Operator

When all variables are true, the logical *AND* operator is employed to provide a true result.

Example: bool.py:

```
a = True
b = True
c = False

a and b
//Output//
 True

a and c
//Output//
 False
```

Logical Operators: OR Operator

When one of the variables is true, the logical *OR* operator is employed to provide a true result.

Example: bool.py

```
a = True
b = True
c = False

a or b
//Output//
True

a or c
//Output//
 True
```

Logical Operators: NOT Operator

A *NOT* operator returns the opposite of a value.

Example: bool.py:

```
a = True
not a

//Output//
False
```

Variable Comparison in Python

It is possible to check equality and inequality between two variables in Python.

Equality (==)

Example: equality.py:

```
a = 10
b = 10
a==b
Output:
True


c  = 5
a == c
Output:
False
```

Inequality (!= or <>)

Example: inequality.py:

```
a = 10
b = 10
a!=b
Output:
False

c = 5
a != c
Output:
True
```

Control Statements in Python

Pass in Control Statements

A pass statement is a null operation with no code to execute and can be used as a place holder for a code.

When can *pass* be used?

1. It can be used when the user does not know what code to write inside the loop.
2. It can be used when the user does not wish to write any code for execution.
3. It can be used when empty code is not allowed, such as in loops, function definitions, and class definitions.
4. It can be used with conditional statements.

Pass in Control Statements

01. Pass can be used when the user does not know what code to write inside the loop.

Example: pass1.py:

```
n = 7
for i in range(n):

  #Pass can be used as a placeholder
  pass
```

Pass in Control Statements

02. Pass can be used when the user does not wish to write any code for execution.

Example: pass2.py:

```
if 1 + 1 == 2:
    print("Coding is fun")
    pass
    print("Hello")

Output:
Coding is fun
Hello
```

Pass in Control Statements

03. The user can use pass where empty code is not allowed, such as loops, function definitions, and class definitions.

Example: pass3.py:

```
def simplilearnFunction:
      pass

#This function will not give any output and
will pass this function
```

Pass in Control Statements

04. Pass statement can be used with conditional statements.

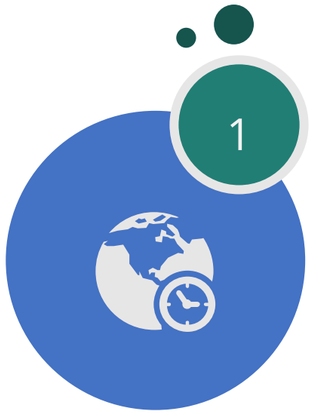
Example: pass4.py:

```
my_list=['a', 'b', 'c', 'd']

for i in my_list:
    if(i =='a'):
        pass
    else:
        print(i)

Output:
b
c
d
```

Conditional Statements

Control or conditional statements allow one to check the conditions and change the behavior of a program.

Conditional statements have these abilities:



It can run a selected section of the program.

Conditions can control the flow of a program.

The different conditions can cover different scenarios.

Conditional Statements: Types

The different types of conditional statements are:

“if” clause

Example:

```
x=5
if x > 0:
 print “x is positive”

Output:
x is positive
```

“if-else” clause

Example:

```
age= 20
if age > 18:
  print “Person is an adult.”
else:
  print “Person is not an adult.”

Output:
Person is an adult.
```

Conditional Statements: Types

The different types of conditional statements are:

“if-elif” clause

Example:

```
marks= 95
if marks > 90:
        print “A grade”
elif marks >= 80:
        print “B grade”
elif marks >= 60:
        print “C grade”

Output:
A grade
```

“if-elif else” clause

Example:

```
marks= 2
if marks > 90:
        print “A grade”
elif marks >= 80:
        print “B grade”
elif marks >= 6:
        print “C grade”
else:
        print “Fail”

Output:
Fail
```

Conditional Statements: Types

The different types of conditional statements are:

Nested if

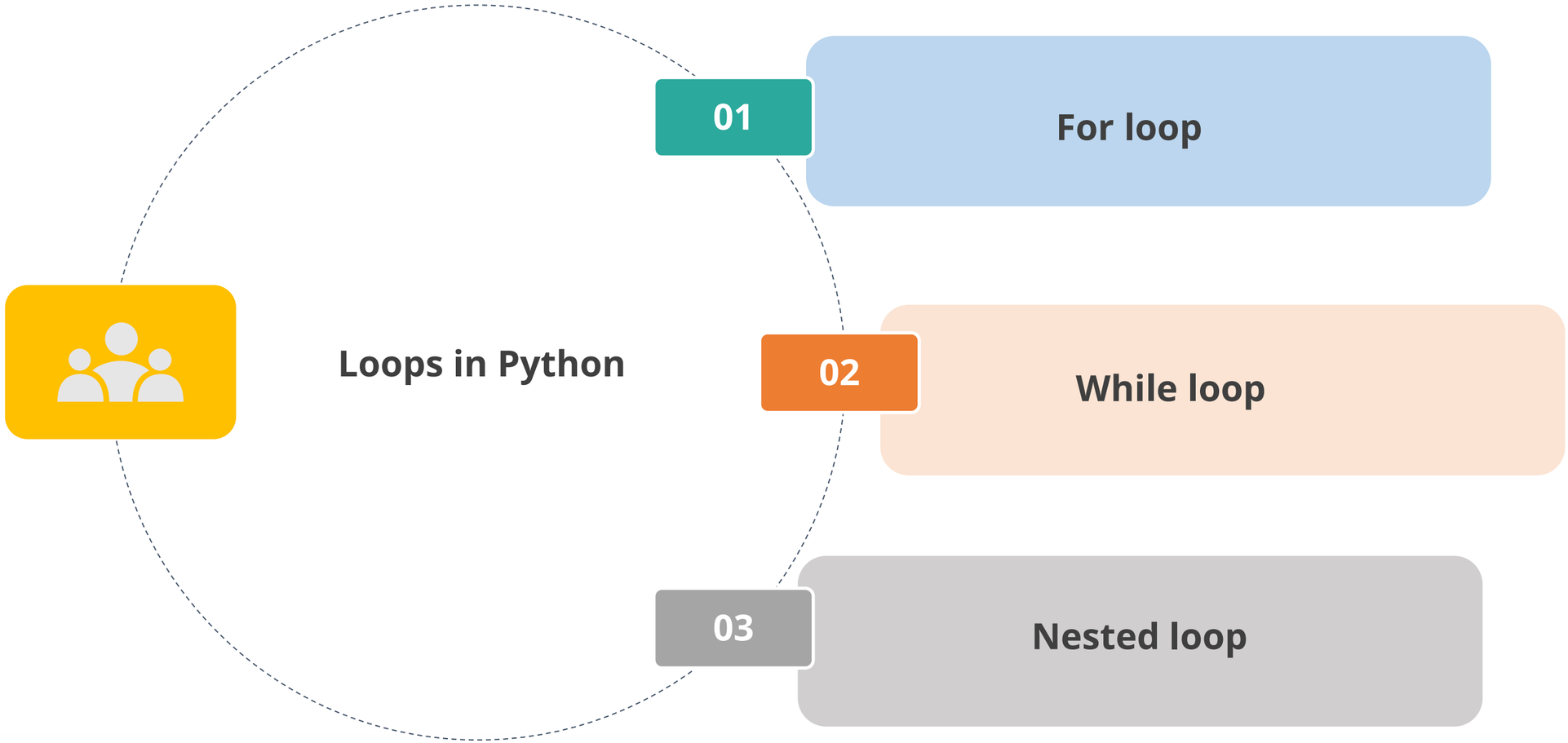
Example:

```
location ='US', age= 20
if location == 'US':
  if age > 18:
     print "Person is an adult from US"
  else:
     print "Person is not an adult from US"

Output:
Person is an adult from US
```

Loop Statements in Python

Loops in Python

A loop statement can execute a statement or group of statements many times.



Range Function

The range function is used with the loop statements and provides a result as a list of numbers that start from zero to a given number minus one.

Example: Range

```
range(5)

Output:
[0, 1, 2, 3, 4]
```

Loops In Python: For Loop

A *for* loop runs an action repeatedly. It helps to iterate over a sequence, such as a list or a string.

Example: For Loop

```
fruits = ['apple' , 'mango' , 'orange' , 'banana’]
for fruit in fruits:
      print fruit

Output:
apple
mango
orange
banana
```

Loops In Python: While Loop

A *while* loop runs an action repeatedly until the base condition is satisfied.

Example: While Loop

```
x = 0
while x <= 5:
        print x
        x = x+1
Output
0
1
2
3
4
5
```

Loops In Python: Nested Loop

It is a loop that is defined inside another loop.

Features of nested loops are:

Features

1. A *while* loop can be nested in a *for* loop, and vice-versa.
2. A *while* loop can also be nested in a *while* loop.
3. A *for* loop can also be nested in a *for* loop.

Loops In Python: Nested Loop

An example of a nested loop is given below:

Example: Nested Loop

```
adjective = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adjective:
  for y in fruits:
    print(x, y)
```

Output:

```
('red', 'apple')
('red', 'banana')
('red', 'cherry')
('big', 'apple')
('big', 'banana')
('big', 'cherry')
('tasty', 'apple')
('tasty', 'banana')
('tasty', 'cherry')
```

Assisted Practice 11.1: List Operations

Duration: 15 minutes

Problem Scenario: Write a program to illustrate different ways operations handle a list

Objective: In this demonstration, you will learn how to work with lists.

Steps Overview:

1. Remove empty strings from the list given below:

List1 = ["Mike", "", "Emma", "Kelly", "", "Brad"]

2. Add a new item to the list after the specified item

Write a program to insert 7000 after 6000 in the following Python list:

List2 = [10, 20, [300, 400, [5000, 6000], 500], 30, 40]

ASSISTED PRACTICE

Assisted Practice 11.1: List Operations

Duration: 15 minutes

3. Write a program to extend the nested list by inserting a sub_list that looks like the expected result provided below:

List3 = ["a", "b", ["c", ["d", "e", ["f", "g"], "k"], "l"], "m", "n"]

sub_list = ["h", "i", "j"]

Expected result: ['a', 'b', ['c', ['d', 'e', ['f', 'g', 'h', 'i', 'j'], 'k'], 'l'], 'm', 'n']

Task 1: Steps

Remove null values from a list

1. Create a list

2. Remove the null values from *List1* and convert the result into a list

3. Print the result

Assisted Practice 11.1: List Operations

Duration: 15 minutes

ASSISTED PRACTICE

Task 2: Steps

Write a program to insert 7000 after 6000 in the following Python list

1. Create a list

2. Append 7000 after 6000 using the append function

3. Print the result

Task 3: Steps

Write a program to extend the nested list by inserting a sub_list:

1. Create a list and the sub_list

2. Extend the inner-most list by appending the created sub_list using the extend function

Assisted Practice 11.2: Swap Operations

Duration: 10 minutes

Problem Scenario: Write a Python program that takes two string inputs from the user and performs the swapping operation on the strings

Objective: In this demonstration, you will learn to work with strings.

Sample string input:

String_1 = “Hello”

String_2 = “Simplilearn”

Expected swapped output:

String_1 = “Simplilearn”

String_2 = “Hello”

Assisted Practice 11.2: Swap Operations

Duration: 10 minutes

Steps Overview:

1. Define two string variables, String_1 and String_2, and assign values to String_1 and String_2

2. Perform the swap operation: Assign the value of String_1 to a temporary variable

3. Assign the value of String_2 to String_1

4. Assign the value of the temporary variable to String_2

Assisted Practice 11.3: Merge Operations

Duration: 10 minutes

Problem Scenario: Write a Python program that takes two dictionaries as the input and merges both inputs
Objective: To work with dictionaries will be the focus of this demonstration

Sample input:

Enter 1 item for the first dictionary:

Enter the key:

“Name”

Enter the value:

“Ted“

Enter 1 item for the second dictionary:

Enter the key:

“Occupation"

Assisted Practice 11.3: Merge Operations

Duration: 10 minutes

ASSISTED PRACTICE

Enter the value:

"Architect“

Expected output:

The two dictionaries merged successfully!

The New Dictionary is:

{'Name': 'Ted', 'Occupation': 'Architect’}

Steps Overview:

1. Prompt user to enter 1 item for the first dictionary: Input key and value
2. Prompt user to enter 1 item for the second dictionary: Input key and value
3. Merge both dictionaries
4. Display the success message and the new merged dictionary

Key Takeaways

- Python is a high-level, interpreted, and object-oriented programming language.
- The frequently used libraries in Python are NumPy, pandas, SciPy, Matplotlib, and scikit-learn.
- Jupyter is an open-source and interactive web-based Python interface for data science and scientific computing.
- Python script can be in batch mode or interpreter mode.

Knowledge Check

Knowledge Check 1

Which of the following commands helps to print the version of Anaconda in Python?

A. --version

B. sys.version

C. print version

D. print sys.version

Knowledge Check 1

Which of the following commands helps to print the version of Anaconda in Python?

A. --version

B. sys.version

C. print version

D. print sys.version

The correct answer is **D**

The command print sys.version helps to print the version of Anaconda in Python.

Knowledge Check

2

Which of the following is NOT an appropriate way to print in Python?

A. print (‘Hi’)

B. print (“Hi”)

C. print Hi

D. All of the above

Knowledge Check 2

Which of the following is NOT an appropriate way to print in Python?

A. print (‘Hi’)

B. print (“Hi”)

C. print Hi

D. All of the above

The correct answer is **C**

print Hi is not an appropriate way to print in Python.

Thank You