**PYTHON FOR DEVOPS**

**Python VS Shell Scripting:**

Mostly Devops engineers deploy their applications in the Linux machines and they are more secured than windows.

To get the details of file or to create a file, CPU utilization, free space and memory usage we use the Shell commands.

Instead of using the commands one by one to get the details. We can simply create a file with extension as **Test.sh** called Shell scripting file we can get all these data at one place.

**Two reasons to use Python instead of Shell scripting:**

1. Not all the deployments are done in Linux, some can deploy in windows also. Although we have ansible for that where we can simply write a playbook but the Ansible is also written in Python.
2. When we are working with complex tasks, API’s and Data manipulation. Python makes it easy for us to achieve all these tasks than Shell scripting.

**Data Types:**

1. Numeric Data Types:
   * int: Represents integers (whole numbers). Example: x = 5
   * float: Represents floating-point numbers (numbers with decimal points). Example: y = 3.14
   * complex: Represents complex numbers. Example: z = 2 + 3j
2. Sequence Types:
   * str: Represents strings (sequences of characters). Example: text = "Hello, World"
   * list: Represents lists (ordered, mutable sequences). Example: my\_list = [1, 2, 3]
   * tuple: Represents tuples (ordered, immutable sequences). Example: my\_tuple = (1, 2, 3)
3. Mapping Type:
   * dict: Represents dictionaries (key-value pairs). Example: my\_dict = {'name': 'John', 'age': 30}
4. Set Types:
   * set: Represents sets (unordered collections of unique elements). Example: my\_set = {1, 2, 3}
   * frozenset: Represents immutable sets. Example: my\_frozenset = frozenset([1, 2, 3])
5. Boolean Type:
   * bool: Represents Boolean values (True or False). Example: is\_valid = True
6. Binary Types:
   * bytes: Represents immutable sequences of bytes. Example: data = b'Hello'
   * bytearray: Represents mutable sequences of bytes. Example: data = bytearray(b'Hello')
7. None Type:
   * NoneType: Represents the None object, which is used to indicate the absence of a value or a null value.
8. Custom Data Types:
   * You can also define your custom data types using classes and objects.

**String Data Type in Python:**

* In Python, a string is a sequence of characters, enclosed within single (' '), double (" ")
* You can access individual characters in a string using indexing, e.g., my\_string[0] will give you the first character.
* Python provides many built-in methods for string manipulation, such as split(), join(), and startswith()

**Split():**

arn = "arn:aws:iam::123456789012:user/johndoe"

print (arn.split("/")[1)

Output:

Johndoe

--------------------------------------------------------------------------------------------------------------------------------------

**Upper()/Lower():**

str1="Hello"  
str2="WORLD"

print(str1.upper()+"! "+ str2.lower())

Output:

HELLO! world

--------------------------------------------------------------------------------------------------------------------------------------

**Concatination:**

str1="Hello"  
str2="World"  
result=str1+"! "+str2  
print(result)

Output:

Hello! World

--------------------------------------------------------------------------------------------------------------------------------------

**Len():**

arn = "arn:aws:iam::123456789012:user/johndoe"

length=len(arn)  
print("length of the string is :",length)

Output:

length of the string is : 38

--------------------------------------------------------------------------------------------------------------------------------------

**Replace():**

text = "Python is awesome"  
new\_text = text.replace("awesome", "great")  
print("Modified text:", new\_text)

Output:

Modified text: Python is great

--------------------------------------------------------------------------------------------------------------------------------------

**REGEX:**

**Regular Expressions for Text Processing:**

* Regular expressions (regex or regexp) are a powerful tool for pattern matching and text processing.
* The re module in Python is used for working with regular expressions.
* Common metacharacters: . (any character), \* (zero or more), + (one or more), ? (zero or one), [] (character class), | (OR), ^ (start of a line), $ (end of a line), etc.
* Examples of regex usage: matching emails, phone numbers, or extracting data from text.
* re module functions include re.match(), re.search(), re.findall(), and re.sub() for pattern matching and replacement.

import re

text = "The quick brown fox"

pattern = r"brown"

search = re.search(pattern, text)

if search:

print("Pattern found:", search.group())

else:

print("Pattern not found")

--------------------------------------------------------------------------------------------------------------------------------------

**KeyWords:**

Keywords are reserved words in Python that have predefined meanings and cannot be used as variable names or identifiers. These words are used to define the structure and logic of the program. They are an integral part of the Python language and are case-sensitive, which means you must use them exactly as specified.

Here are some important Python keywords:

1. **and**: It is a logical operator that returns True if both operands are true.
2. **or**: It is a logical operator that returns True if at least one of the operands is true.
3. **not**: It is a logical operator that returns the opposite of the operand's truth value.
4. **if**: It is used to start a conditional statement and is followed by a condition that determines whether the code block is executed.
5. **else**: It is used in conjunction with if to define an alternative code block to execute when the if condition is False.
6. **elif**: Short for "else if," it is used to check additional conditions after an if statement and is used in combination with if and else.
7. **while**: It is used to create a loop that repeatedly executes a block of code as long as a specified condition is true.
8. **for**: It is used to create a loop that iterates over a sequence (such as a list, tuple, or string) and executes a block of code for each item in the sequence.
9. **in**: Used with for, it checks if a value is present in a sequence.
10. **try**: It is the beginning of a block of code that is subject to exception handling. It is followed by except to catch and handle exceptions.
11. **except**: Used with try, it defines a block of code to execute when an exception is raised in the corresponding try block.
12. **finally**: Used with try, it defines a block of code that is always executed, whether an exception is raised or not.
13. **def**: It is used to define a function in Python.
14. **return**: It is used within a function to specify the value that the function should return.
15. **class**: It is used to define a class, which is a blueprint for creating objects in object-oriented programming.
16. **import**: It is used to import modules or libraries to access their functions, classes, or variables.
17. **from**: Used with import to specify which specific components from a module should be imported.
18. **as**: Used with import to create an alias for a module, making it easier to reference in the code.
19. **True**: It represents a boolean value for "true."
20. **False**: It represents a boolean value for "false."
21. **None**: It represents a special null value or absence of value.
22. **is**: It is used for identity comparison, checking if two variables refer to the same object in memory.
23. **lambda**: It is used to create small, anonymous functions (lambda functions).
24. **with**: It is used for context management, ensuring that certain operations are performed before and after a block of code.
25. **global**: It is used to declare a global variable within a function's scope.
26. **nonlocal**: It is used to declare a variable as nonlocal, which allows modifying a variable in an enclosing (but non-global) scope.

--------------------------------------------------------------------------------------------------------------------------------------

**Variables:**

**Variable Scope:** In Python, variables have different scopes, which determine where in the code the variable can be accessed. There are mainly two types of variable scopes

**Local Scope:** Variables defined within a function have local scope and are only accessible inside that function.

**Global Scope:** Variables defined outside of any function have global scope and can be accessed throughout the entire code.

**Code:**

a=5 #global variable  
b=6 #global variable  
def addition():  
 a=8 #local variable  
 print(“Addition is ”,a+b)   
def substraction():  
 a=4 #local variable  
 print(“Substraction is”,a-b)  
addition() #calling the functions  
substraction()

Output:

Addition is 14 (Here addition of one global and local variable, preference given to local variable)

Substraction is -2 (Here sub of one global and local variable, preference given to local variable)

**Modules:** Multiple functions in a program called modules

**Packages:** Multiple modules is called Packages

**Functions:**

A function in Python is a block of code that performs a specific task. Functions are defined using the def keyword and can take inputs, called arguments. They are a way to encapsulate and reuse code.

**Code:**

a=5  
b=6  
add=a+b  
print("addition is",add)  
def addition():  
 a=8  
 print("Addition is",a+b)  
def substraction():  
 a=4  
 print("Substraction is",a-b)  
addition()  
substraction()

Once a function is declared we have to call the function to use it if it has a print statement. If we use the return then the functions output will be stored and we have to call the function in print statement

**Advantages of Functions:**

* Readability
* Reuseability
* Debugging

A function will have a prefix def before the function name. The content of the function is bounded by the spaces or indentations.

While creating resources in the AWS with SDK functions play a major role. The functions will help in easy identification of the errors.

We can create resources through AWS CLI, CFT and Terraform. But they have some limitations.

This uses a module called boto3 in python for API calls in cloud.

def S3():

------------

------------

def EC2():

------------

------------

**def add(num1, num2):  
 a=num1+num2  
 return a  
def sub(num1, num2):  
 s=num1-num2  
 return s  
def mul(num1, num2):  
 m=num1\*num2  
 return m**

**print(add(5,6))  
print(sub(6,6))  
print(mul(6,5))**

Once a function is declared we have to call the function to use it if it has a print statement. If we use the return then the functions output will be stored and we have to call the function in print statement

**Modules:**

A module is a Python script containing Python code. It can define functions, classes, and variables that can be used in other Python scripts. Modules help organize and modularize your code, making it more maintainable.

A module is a group of functions and can be reuseable for multiple projects.

We can create a module and use it by simply importing it with right path.

Code:1

a=1  
b=7  
def addition():  
 print("Addition is",a+b)  
def substraction():  
 print("Substraction is",a-b)  
addition()  
substraction()

saved as function\_example.py

Code:2

import function\_example as cal  
cal

##cal.addition() this is to invoke a particular function in the module.

Here we created a module and imported it in another code to reuse it.

**Packages:**

A package is a collection of modules organized in directories. Packages help you organize related modules into a hierarchy. They contain a special file named \_\_init\_\_.py, which indicates that the directory should be treated as a package.

**Suppose you have a package structure as follows:**

my\_package/

\_\_init\_\_.py

module1.py

module2.py

You can use modules from this package as follows:

from my\_package import module1

result = module1.function\_from\_module1()

**How to Import a Package**

# Import the entire module

import math

# Use functions/variables from the module

result = math.sqrt(16)

print(result)

# Import specific function/variable from a module

from math import pi

print(pi)

In this example, we import the math module and then use functions and variables from it. You can also import specific elements from modules using the from module import element syntax.

**Python workspaces:**

Python workspaces refer to the environment in which you develop and run your Python code. They include the Python interpreter, installed libraries, and the current working directory. Understanding workspaces is essential for managing dependencies and code organization.

Python workspaces can be local or virtual environments. A local environment is the system-wide Python installation, while a virtual environment is an isolated environment for a specific project. You can create virtual environments using tools like virtualenv or venv.

**Example:**

# Create a virtual environment

python -m venv myenv

# Activate the virtual environment (on Windows)

myenv\Scripts\activate

# Activate the virtual environment (on macOS/Linux)

source myenv/bin/activate

Here in this we can create virtual environments where we can create different virtual environments to work for each project in the same machine.

Each virtual environment can have different versions of a software like one env can have 8.1.2 and other can have 8.3.2 versions.

**Command Line Arguments:**

In this we have to import a module called sys. This takes the command line arguments.

**import sys  
  
def sum(num1,num2):  
 a=num1+num2  
 return a  
def sub(num1,num2):  
 s=num1-num2  
 return s  
def mul(num1,num2):  
 m=num1\*num2  
 return m  
  
num1= int(sys.argv[1])  
operation= sys.argv[2]  
num2= int(sys.argv[3])  
  
if operation=="+":  
 output=sum(num1,num2)  
 print(output)  
elif operation=="-":  
 output=sub(num1,num2)  
 print(output)  
else:  
 print("Provide the operator + or -")**

**Input**: python file.py 2 + 10

This will be run only in terminal not as a file. We have to provide the values and also the operation in the same line.

**Output**:10

**Environmental Variables:**

Sensitive information like Passwords, API keys, etc are to be hidden. We can store the passwords as environmental variables and can use then when ever necessary.

export password=”kummy” (in terminal)

**To view in code:**

import os

print(os.getenv(“password”)

**Lists & Tuples:**

These are a sequence data types where to mitigate the problem of creating variables for each data we can simply put the data in a list or tuple to get them when required.

If we have hundred S3 bucket names and want to declare them in the code we need hundred variables which is a difficult task. For this we can use the lists to declare the data in it.

**S3\_bucket\_list=[**“first\_bucket”,” second\_bucket”,”third\_bucket”,” fourth\_bucket**”]**

**Lists** are mutable means we can edit the contents in the list. For lists the memory allocation is dynamic as the size may vary.

**Vegies\_list = [“tomato”,”potato”,”brinjal”]**

Vegies\_list.append(“carrot”)

Vegies\_list.remove(“potato”)

print (vegies\_list[0]) (Here the tomato gets printed as it is the first element) 0 is index number.

print (vegies\_list[2])

print (len(vegies\_list)) (gives length of the lists)

**Tuples** are immutable means we cannot edit the contents. For tuples the memory allocation is static as the size cannot be changed.

**Vegies = (“tomato”,”potato”,”brinjal”)**

**Loops:**

A repetitive execution of a block of code is called looping.

**Exp**:

for i in range (10):

print(“I’m the God”)

This gives the output statement 10 times.

**For Loop:** A definite execution of code.

colors=["yellow","green","blue"]  
  
for i in colors:  
 print(i)

**Output:**

Yellow

Green

Blue

**Code 2:**

**for i in range(10):  
 print(i)  
  
colors=["yellow","green","blue","orange","red"]  
for color in colors:  
 if color=="blue":  
 break  
 print(color)  
for color in colors:  
 if color=="blue":  
 continue  
 print(color)**

**Output:**

Yellow, green yellow, green, orange, red ---these gets printed

**While Loop**: An indefinite execution of code as long as the condition is true.

Loop manipulations are break and continue.

**Break**: This breaks the loop when the condition is met.

In the code above if the blue comes then loop stops

**Continue**: This will skip the condition and continues further

In the code above if blue comes then it skips the blue and continues further.

We can use continues and break in loops only.

**Exception Handling:**

**try:**

----code-----

**except (errortype):**

print statement for the exception

continue/break (to stop the code or skip the part)

Example:

a=int(input("enter the first value: "))  
b=int(input("enter the second value: "))  
c=input("enter the operation symbol : ")  
if c=="+":  
 print(a+b)  
elif c=="-":  
 print(a-b)  
elif c=="/":  
 try:  
 print(a/b)  
 except ZeroDivisionError:  
 print("provide a value greater than 0")  
elif c=="\*":  
 print(a\*b)

**Python Real Time UseCase with Lists & Exceptional Handling**

List all the files in the list of folders that user provides.

**import os  
##split() is used to seperate the inputs  
folders= input("Enter the folders with spaces separating ").split()  
#For loop to fetch the folder from the folders list  
for folder in folders:  
#The os module will help in listing the files in each folder  
#Execption block here as the folder names checked and files are assigning starts here into the files list  
 try: #To handle the file not found errors and permission errors  
 files=os.listdir(folder)  
 except FileNotFoundError:  
 print("Entered folder is not available : " + folder)  
 continue ##Continue to the next folder check and skips the error folder  
 except PermissionError:  
 print("No permission to access the folder : " + folder)  
 print("----list of files in the folder: "+ folder)**

**#For loop to fetch the files from the files list  
 for file in files: #prints the files in the folder one by one  
 print(file)**

**Output:**

This has to be run in the terminal.

**Dictionaries:**

Dictionaries are similar to lists but they can store the properties of the variables.

These uses a concept called key value pair. The key is the reference variable and it has an value, so if we call the key then the value of that particular key is called for output.

“name” : “Sai” = key : value , each pair is separated by commas

student\_inf={  
 "name":"Sai",  
 "age":"21",  
 "class":"11"  
 }

print(student\_inf["age"])

Here the dictionary is represent by flower brackets “{}”.

**Problem:** Suppose there is a list of student details like below

These details have to be made into a list and can be printed as output.

Using list we can do this as below

#Name: Sai  
#Age: 21  
#Class: 11  
student\_info=["Sai","21","11"]  
print(student\_info[0])

But we have to remember the correct index number and also it is difficult to get the data for more students in a single list. We can overcome this with dictionaries.

student\_inf={  
 "name":"Sai",  
 "age":"21",  
 "class":"11"  
 }  
print(student\_inf["age"])

we will get the output as the age = 21. But this is for one student data and if we want more students data we need the use the combination of the dictionaries and the lists to get the desired output from the given data.

##list and dictionary  
students\_info=[  
 {  
 "name":"Sai",  
 "age":"21",  
 "class":"11"  
 },  
 {  
 "name":"Kumar",  
 "age":"20",  
 "class":"10"  
 }  
]  
print(students\_info[1]["age"])

We get the output as 20 (index value is 1 and the details of second student)

**Accessing the file, reads and writes the data inside the file with python program.**

**Problem:**

**Access a file and update the data inside the file so that the server allows more requests into the application.**

**File:**

**##SERVER CONFIGURATION  
IP\_ADDRESS = 172.58.162.30  
MAX= 566  
PORT= 8080**

**Program:**

file\_path=input("enter the file name: ")  
key=input("Enter the key to be changed: ")  
value=input("Enter the value: ")  
  
def server\_config (file\_path, key, value):  
 with open(file\_path, "r") as file:  
 lines= file.readlines()  
 with open(file\_path, "w") as file:  
 for line in lines:  
 if key in line:  
 file.write(key + "= " + value + "\n" )  
 else:  
 file.write(line)  
server\_config(file\_path, key, value)

This program will change the required filed in the file mentioned above