Python Important Notes & Commands for DevOps

Why Python?

Python is one of the easiest programming language to learn and knowns for its versatility and user-friendly syntax, Also python is an open source programming language and have extensive libraries to make programming easy. It has massive use across different industries.

Iska use kai jagah hota hai: Web development, Automation, Devops, Game development and Data science.

💡 **Variables in Python:**

Variable matlab ek naam jisme tum data store karte ho. Jaise apni jeb mein paise rakhne ka naam hai jeb, waise hi memory me data rakhne ka naam hai Variable.

✅ **Python me Variable kaise banta hai?**

```
x = 10

naam = "Anuj"

pi = 3.14
```

Yahan:

X ek integer variable hai (Value 10)

Naam ek staring variable hai (Anuj)

Pi ek float variable hai (3.14)

🧠 **Important Rules:**

1. Variable ka naam alphabet ya underscore (_) se shuru ho sakta hai
2. Naam me space allowed nahi
3. Case-sensitive hai → naam aur Naam alag hain
4. Keywords jaise for, while, if variable naam nahi ban sakte

🔥 **Python me type batana zaroori nahi hota**: Python **dynamically typed** language hai, matlab type automatically samajh leta hai.

```
a = 5      # int

a = "hello"  # ab string ban gaya
```

Data type in Python:

In python, a data type is a classification that specifies the type of value a variable can hold. We can check data type using type() function.

Examples:

1. my_name = "Madhav"

>>> type(my_name)

O/P: **<class 'str'>**

2. value = 101

>>> type(value)

O/P: **<class 'int'>**

🔥 Basic Data Types in Python:

Data Type	Example	Explanation
int	10, -5, 0	Integer (poore number)
float	3.14, -7.5, 0.0	Decimal number
str (string)	"Hello", 'Anuj'	Text ya words
bool	True, False	True ya False (0 ya 1 type)
list	[1, 2, 3]	Items ka collection (ordered)
tuple	(4, 5, 6)	List jaisa hi but immutable (badal nahi sakte)
dict (dictionary)	{"name": "Anuj", "age": 22}	Key-Value pair ka set
set	{1, 2, 3}	Unique elements ka group

Note: Python mein sb kuch object hota hai even a number bhi

Typecasting in Python:

Type casting means ek data type ko dusre type mein badalna.

Python mein 2 types of type casting hoti hai:

1**. Implicit type casting (python khud krta hai):**

```
a = 5     # int
b = 3.2    # float
c = a + b  # int + float = float
print(c)   # Output: 8.2
print(type(c))  # <class 'float'>
```

2. **Explicit type casting (Hum khud krte hai)**

Function	Use for
int()	Kisi cheez ko integer me badalna
float()	Kisi cheez ko float me badalna
str()	Kisi cheez ko string me badalna
list(), tuple(), set()	Collections me badalne ke liye

Example:

```
# string to int
x = "100"
y = int(x)
print(y)      # Output: 100
print(type(y))  # <class 'int'>

# float to int
a = 5.7
b = int(a)
print(b)      # Output: 5

# int to string
c = 10
d = str(c)
print(d)      # Output: "10"
```

⚡ **Quick Note:**

Jab string ko number me badal rahe ho, dhyan rahe string me sahi number likha ho.
Warna error dega.

```
int("123")  # OK
int("hello")  # ❌ Error
```

Input function in Python - Taking user input in Python

Example:

```
name = input("Bhai tera naam kya hai? ")      # Jo user, input dega vo 'name' variable mein store ho jaega.
print("Arey wah " + name + ", badiya naam hai!")
```

🔢 **Number lena hai to?**

Input se hamesha string milta hai. Agarnumber chahie toh int() yaa float() se convert karo:

```
age = int(input("Bhai teri umar kitni hai? "))
print("Ohh, toh tu " + str(age) + " saal ka hai.")
```

Question: Can integers be strings?

Yes, easily convertible:

```
num = 123
str_num = str(num)  # Ab yeh string ban gaya
print(str_num)     # Output: '123'
print(type(str_num))  # <class 'str'>
```

Question: Can String be integers?

Haan, lekin sirf jab string ke ander sirf number hi ho.

```
s = "456"
num = int(s)
print(num)      # Output: 456
print(type(num))  # <class 'int'>
```

🎯 Python String Operations

🔹 **1. Concatenation (Jodna)**

```
a = "Hello"
b = "Bhai"
print(a + " " + b)   # Output: Hello Bhai
```

🔹 **2. Repetition (Repeat karna)**

```
laugh = "Ha"
print(laugh * 3)  # Output: HaHaHa
```

🔹 **3. Length Check (Kitne character hain?)**

```
msg = "Bhai ka code"
print(len(msg))   # Output: 12
```

🔹 **4. Indexing (Character nikalna)**

```
name = "Python"
print(name[0])   # P
print(name[-1])  # n
```

🔹 **5. Slicing (Tukda nikalna)**

```
text = "HelloDost"
print(text[0:5])  # Hello
print(text[5:])   # Dost
```

🔹 **6. Case Changing**

```
msg = "hello bhai"
print(msg.upper())  # HELLO BHAI
print(msg.capitalize())  # Hello bhai
print(msg.lower())  # hello bhai
```

🔹 **7. Replace (Badalna)**

```
line = "Hello Bhai"
print(line.replace("Bhai", "Dost"))  # Hello Dost
```

🔹 **8. Split (Todna word mein)**

```
data = "chai sutta coding"
print(data.split())  # ['chai', 'sutta', 'coding']
```

🔹 **9. Join (List ko string banana)**

```
words = ['Code', 'kar', 'bhai']
print(" ".join(words))  # Code kar bhai
```

🔹 **10. Check Start/End**

```
msg = "hello bhai"
print(msg.startswith("hello"))  # True
print(msg.endswith("bhai"))     # True
```

🔹 **11. Find / Index (Kaha hai ye word?)**

```
msg = "kya haal bhai"
print(msg.find("haal"))   # 4
```

🔹 **12. Remove Space (Strip)**

```
msg = "  hello bhai  "
print(msg.strip())   # "hello bhai"
```

🔹 **13. Check if numeric / alpha**

```
print("123".isdigit())    # True
print("bhai123".isalnum()) # True
print("bhai".isalpha())    # True
```

🔥 **Bhai ki tip:**

Strings Python mein immutable hote hain, matlab ek baar ban gaya to badlaav kaam se hota hai, naya string banta hai.

🔪 String Slicing in Python

Format: **string [start : end : step]**

start → kaha se shuru karna hai (index)

end → kaha tak lena hai (but end waala **include nahi hota** 😅)

step → kitne step mein aage badhna (default: 1)

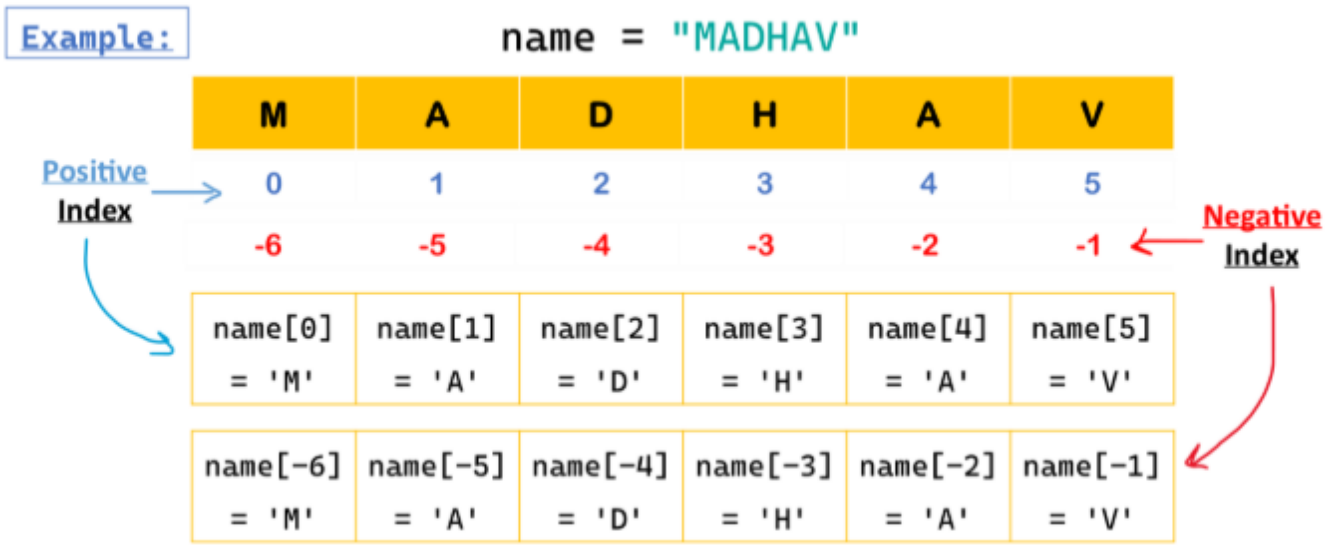
🔥 **Examples:**

```
msg = "HelloBhai"
print(msg[0:5])    # Hello → 0 se 4 tak
print(msg[5:])     # Bhai  → 5 se end tak
print(msg[:5])     # Hello → starting se 4 tak
print(msg[-4:])    # Bhai  → last ke 4 char
print(msg[::2])    # HloBi → har dusra character
print (msg[:]) = msg[::]  #"HelloBhai"
```

🔁 **Reverse karna string:**

```
print(msg [::-1])  # iahBo lleH → full ulta
```

📌 **Indexing ka cheat sheet:**



Notes: msg[2:5] means → index 2 se le, lekin 5 **tak nahi**, 4 tak.

Step se tu skip kar sakta hai, jaise msg[::3] → har 3rd character.

count operations: .count()

```
msg = "Bhai kya kar raha hai bhai?"
print(msg.count("bhai"))    # Output: 2 (chhoti 'b' waale)
print(msg.count("Bhai"))    # Output: 1 (Bade 'B' waale)
```

🎯 **Count in a specific range:**

```
text = "hello bhai"
print(text.count("bhai", 6, 20))  # 2 (count karega index 6 to 19)
```

Comparison Operations:

```
x = 5
y = 10
print(x == y)  # ❓ Barabar hai kya? False
print(x != y)  # ❌ Barabar nahi? True
print(x > y)   # 🔼 Bada hai kya? False
print(x < y)   # 🔽 Chhota hai kya? True
print(x >= y)  # 🔼 ya == ? False
print(x <= y)  # 🔽 ya == ? True
```

Assignment Operations:

```
x = 5    # assign
x += 2   # x = x + 2 → 7
x -= 1   # x = x - 1 → 6
x *= 3   # x = x * 3 → 18
x /= 2   # x = x / 2 → 9.0
```

Conditional statement in Python

🔹 **1. if**

Bas ek condition check karta hai.

```
age = 20
if age >= 18:
   print("Bhai tu vote de sakta hai!")
```

🔹 **2. if...else:** Condition true ho toh kuch, warna kuch aur.

```
age = int(input("Umar likh bhai: "))
if age >= 18:
  print("Entry mil gayi bhai! 🕺 ")
else:
  print("Chhote abhi tu wait kar 😅 ")
```

🔹 **3. if...elif...else:** Multiple condition check karne ke liye.

```
marks = 75
if marks >= 90:
  print("Bhai topper hai tu! 🔥 ")
elif marks >= 60:
  print("Theek hai bhai, pass ho gaya! 😳 ")
else:
  print("Padh le bhai, warna back lag jayegi! 📚 ")
```

🔹 **Match-case Statement**

📌 **Syntax:**

```
match variable:
  case value1:
    # kaam1
  case value2:
    # kaam2
  case _:
    # default kaam (else jaise)
```

Example:

```
# User se input lo
name = input("Naam bata bhai: ")
age = int(input("Umar kitni hai bhai: "))
# Tuple banao
user = (name, age)
# match-case use karo
match user:
  case (name, age) if age >= 18:
    print(f"{name} bhai adult hai! 🔥 ")
  case (name, age):
    print(f"{name} abhi chhota hai bhai... 📚 ")
```

Loops in Python

1. For loop in Python:

```
# List ke elements pe loop chalane ka example
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
  print(fruit)      #Output: apple banana cheery
```

Agar tu range() use krega to ek sequence of numbers banake unpe bhi loop chalana possible hai:

```
for i in range(5):
  print(i)          #Output: 0 1 2 3 4
                              OR
for i in range(0,5)
print(i)          #Output: 0 1 2 3 4
```

Write a program to print even and odd number till 100?

```
# Even numbers print karna
print("Even numbers till 100:")
for i in range(101):
  if i % 2 == 0:
    print(i)

# Odd numbers print karna
print("Odd numbers till 100:")
for i in range(101):
  if i % 2 != 0:
    print(i)
```

Alternate ways:

```
# Even numbers print karna
print("Even numbers till 100:")
for i in range(2,101,2):
    print(i)

# Odd numbers print karna
print("Odd numbers till 100:")
for i in range(1, 100, 2):
  print(i)
```

Note:

Syntax	Meaning
range(2, 101, 1)	2 se 100 tak, 1-1 number badhe
range(2, 101, 2)	2 se 100 tak, 2-2 number badhe (even only)
range(2, 101, 0)	❌ Error — step zero allowed nahi hai python mein

2. While loop:

✅ **Even numbers from 1 to 100 using while loop:**

```
print("Even numbers till 100:")
i = 2
while i <= 100:
  print(i)
  i += 2
```

✅ **Odd numbers from 1 to 100 using while loop:**

```
print("Odd numbers till 100:")
i = 1
while i <= 100:
  print(i)
  i += 2
```

Flow control statements

✅ **break — loop ko turant tod deta hai**

```
i = 1
while i <= 10:
  if i == 5:
    break
  print(i)
  i += 1
#Output: 1 2 3 4
```

Jab i == 5 hota hai, break laga ke loop ko tod diya jata hai.

✅ **continue — current iteration ko skip karta hai**

```
i = 0
while i < 10:
  i += 1
  if i == 5:
    continue
  print(i)
#Output 1 2 3 4 6 7 8 9 10
```

Jab i == 5 hota hai, continue us iteration ko skip kar deta hai — 5 print nahi hota.

✅ **pass — khali placeholder (kuch nahi karta)**

```
for i in range(5):
  if i == 3:
    pass
  print(i)
```

pass ka use tab hota hai jab tu syntax complete rakhna chahta hai but koi kaam nahi karna and runs the entire code without causing any syntax error.

Functions in Python

Function, Pyhton mein ek resuable block hota hai jo specific kaam karta hai. Bar same code likhne se bachne ke liye functions ka use hota hai.

- Function ek independent block hota hai.
- Direct call karte hai.

✅ **Basic syntax of a function:**

```
def function_name():
  # function body
  print("Hello bhai!")
```

🔁 **Call the function:**

```
function_name()
```

Output:

```
Hello bhai!
```

✅ **Function with parameters:**

```
def greet(name):
   print("Hello", name)
greet("Raju")            #Output: Hello Raju
```

✅ **Function with return value:**

```
def add(a, b):
   return a + b
result1 = add(10, 5)
result2 = add(15,6)
print("First_sum:", result1,  "Second_sum", result2)
#Output:  First_sum: 15, Second_sum: 21
```

Types of function in Python:

1. Build-in function (jo Python ke ander already bane hote hai)

```
print("Hello")      # Output print karta hai
len("hello")        # String ki length deta hai
range(5)          # Range banata hai
```

2. User-defined Function (jo tum khud banate hai)

```
def greet():
   print("Hello bhai!")
greet()
```

Method bhi function jaisi hi hota hai, lekin kissi object ya class ke ander hota hai. Objkect ke sath call karte hai.

Example:

```
name = "raju"
print(name.upper())   # upper() ek method hai
```

Yaha upper() ek **method** hai jo string object (name) pe kaam kar raha hai.

---

f-string : f-string ka full naam hai: Formatted string literal

jab tum ek string ke ander direct variable ghusanna chahte ho simple aur clean way mein, tab f-string use karte hai.

```
Syntax: f"Hello {variable}"
```

🔥 **Example se dekh bhai:**

- **Normal (bina f-string ke):**

```
name = "Raju"
age = 25
print("My name is " + name + " and I am " + str(age) + " years old.")
```

Isme dekh kitna **+ str()** conversion karna padh rah ai – code ganda lag raha hai isme.

- **F-string ke sath:**

```
name = "Raju"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

🧠 **Quick Points about f-strings:**

Without f-string	With f-string
"Hello " + name + "!"	f"Hello {name}!"
str() lagana padta hai	Auto string bana deta hai
Zyada confusing	Simple aur readable
Slow thoda	Fast

📌 **Tujhe sirf ek cheez yaad rakhni hai bhai:**

Jab bhi string ke andar variable, number, ya calculation dalna ho — **f-string ka use karo!** Bas likhne se pehle f lagana hai aur {} ke andar variable ya expression daal dena hai.

Aur example le:

Bina f-string:

```
a = 5
b = 10
print("Sum of " + str(a) + " and " + str(b) + " is " + str(a+b))
```

f-string ke sath:

```
a = 5
b = 10
print(f"Sum of {a} and {b} is {a+b}")
```

Output dono ka:

```
Sum of 5 and 10 is 15
```

What is a list in Python?

List ek ordered collection hota hai jisme tum multiple items stores kar sakte ho.

List ke ander numbers, strings, dusri list, kuch bhi daal skte hai

List is mutable – can be edited, added, deleted

Example of list:

```
fruits = ["apple", "banana", "cherry"]
print(fruits)
```

Output:

```
['apple', 'banana', 'cherry']
```

List operations:

```
# Main list
fruits = ['apple', 'banana', 'cherry']
print("Starting list:", fruits)
```

Bilkul! Main ek hi list bana ke sab important list methods ke **example** achhe tareeke se dikhaata hoon.

Chalo ek list lete hain:

Main list

fruits = ['apple', 'banana', 'cherry']

print("Starting list:", fruits)

---

1. append(x)

List ke end mein item add karta hai

fruits.append('orange')

print("After append:", fruits)

Output: ['apple', 'banana', 'cherry', 'orange']

---

2. insert(i, x)

Kisi particular index pe item insert karta hai

fruits.insert(1, 'mango')

print("After insert at index 1:", fruits)

Output: ['apple', 'mango', 'banana', 'cherry', 'orange']

---

3. extend(l2)

Ek list ke andar dusri list jod deta hai

more_fruits = ['kiwi', 'pineapple']

fruits.extend(more_fruits)

print("After extend:", fruits)

Output: ['apple', 'mango', 'banana', 'cherry', 'orange', 'kiwi', 'pineapple']

---

4. remove(x)

First occurrence of item delete karta hai

fruits.remove('banana')

print("After remove 'banana':", fruits)

Output: ['apple', 'mango', 'cherry', 'orange', 'kiwi', 'pineapple']

---

5. pop(i)

Index ke basis pe item hataata hai

fruits.pop(2) # index 2 ka item hataayenge

print("After pop index 2:", fruits)

Output: ['apple', 'mango', 'orange', 'kiwi', 'pineapple']

---

6. clear()

Puri list empty kar deta hai

(Ye thoda end mein dikhaunga nahi toh list khali ho jayegi 😅)

---

7. index(x)

Item ka index return karta hai

print("Index of 'kiwi':", fruits.index('kiwi'))

Output: 3

---

8. count(x)

Item kitni baar aaya hai list mein batata hai

fruits.append('apple') # ek aur apple daal dete hain

print("After adding another 'apple':", fruits)

print("Count of 'apple':", fruits.count('apple'))

Output:
After adding another 'apple': ['apple', 'mango', 'orange', 'kiwi', 'pineapple', 'apple']
Count of 'apple': 2

9. sort()

List ko ascending order mein arrange karta hai

fruits.sort()

print("After sort:", fruits)

Output: ['apple', 'apple', 'kiwi', 'mango', 'orange', 'pineapple']

---

10. reverse()

List ko ulta kar deta hai

fruits.reverse()

print("After reverse:", fruits)

Output: ['pineapple', 'orange', 'mango', 'kiwi', 'apple', 'apple']

---

11. concatenate (manual)

Do list ko jodne ka kaam manually + se karte hain

veggies = ['tomato', 'potato']

combined = fruits + veggies

print("After concatenation:", combined)

Output: ['pineapple', 'orange', 'mango', 'kiwi', 'apple', 'apple', 'tomato', 'potato']

Note: Concatenation ka koi special method nahi hai, direct + operator se hota hai.

---

12. update

List mein direct update aise hota hai

fruits[0] = 'grapes'

print("After updating index 0 to 'grapes':", fruits)

Output: ['grapes', 'orange', 'mango', 'kiwi', 'apple', 'apple']

Bonus: clear()

List ko pura empty karne ka example

fruits.clear()

print("After clear:", fruits)

Output: []

🔷 Tuple in Python

- **Definition:** Tuple ek *ordered* collection hai items ka.
- **Important:**
  - Elements ka order fix rehta hai.
  - Tuple **immutable** hota hai (banne ke baad change nahi hota).
  - It also allows duplicates
- **Syntax:**
  - Round brackets ( ) ka use hota hai.

Example:

```
# Tuple banana
my_tuple = ('apple', 'banana', 'cherry')
print(my_tuple)

# Accessing elements
print(my_tuple[1])  # 'banana'

# Slicing
print(my_tuple[0:2])  # ('apple', 'banana')

# Tuple length
print(len(my_tuple))  # 3

# Nested Tuple
nested = (1, 2, (3, 4))
print(nested[2])  # (3, 4)
print(nested[2][1])  # 4
```

🔷 Set in Python

- **Definition:** Set ek *unordered* collection hai unique items ka.
- **Important:**
  - No duplicate elements allowed.
  - Unordered (order ka guarantee nahi hota).
  - **Mutable** hota hai (new elements add kar sakte ho).

- **Syntax:**
    - Curly braces { } ka use hota hai.

```
# Set banana
my_set = {'apple', 'banana', 'cherry'}
print(my_set)

# Duplicate add karoge toh ignore hoga
my_set.add('apple')
print(my_set)  # 'apple' ek hi baar rahega

# Naya item add karna
my_set.add('orange')
print(my_set)

# Removing item
my_set.remove('banana')
print(my_set)

# Check item present hai ya nahi
print('apple' in my_set)  # True

# Set length
print(len(my_set))  # 3 (after removal)

# Union of two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1.union(set2))  # {1, 2, 3, 4, 5}

# Intersection of two sets
print(set1.intersection(set2))  # {3}
```

🔷 Dictionary in Python

- **Definition**:
  Dictionary ek *key-value pair* ka collection hai.
- **Important Points**:
  - Keys **unique** hote hain.
  - Values kuch bhi ho sakti hain (duplicate bhi allowed hai).
  - **Mutable** hai (badal sakte ho).
  - **Unordered** hai (Python 3.7+ mein insertion order preserve hota hai).

Syntax: `my_dict = {key1: value1, key2: value2}`

Dictionary Example:

```
# Ek simple dictionary
student = {
   "name": "Rahul",
   "age": 22,
   "city": "Mumbai"
}
print(student)
```

Output:

```
{'name': 'Rahul', 'age': 22, 'city': 'Mumbai'}
```

🔷 Kya hai if __name__ == "__main__": ?

- Jab Python ka koi file **direct run** karte ho, to uska __name__ variable "__main__" ban jaata hai.
- Jab wohi file **dusre file mein import** hoti hai, to __name__ ban jaata hai **file ka naam**.

⚡ **Isliye:**

if __name__ == "__main__": ka matlab hota hai — *"Sirf tabhi code run karo jab file ko directly chalaya jaaye, import hone par nahi."*

🔥 Simple Example:

```
def greet():
   print("Hello from function!")
print("Top level code chal gaya.")
if __name__ == "__main__":
   greet()
```

🔷 return kya hai?

- Jab function ka kaam khatam hota hai aur **kuch result wapas bhejna** hota hai, tab return use karte hain.
- return function ke andar likha jaata hai.
- Jaise koi "order complete" hone ke baad "bill amount" milta hai, waise hi!
- Jab hum function banaate hain, wo kuch **kaam (task)** karta hai.
- **return function** ka final output hota hai — jo batata hai ki function ne kya kaam kiya aur kya result diya.

Example:

```
def greet():
  return "Hello"
def give_func():
  return greet  # function ka reference de diya, call nahi kiya
x = give_func()  # x mein ab greet function aa gaya
print(x())      # Ab x() karke greet ko call kiya
```

🔷 Constructor kya hota hai?

- Jab **object create** karte ho kisi class ka, tab **constructor automatically** call hota hai.
- Python mein constructor ka naam **__init__** hota hai.
- Iska kaam hota hai:
    - **Object ke variables initialize** karna.
    - **Starting setup** karna jab object ban raha ho.

```
class Student:
  def __init__(self, name, age):
    self.name = name  # Object ka naam set karo
    self.age = age    # Object ka age set karo

# Ab jab object banayenge to __init__ apne aap chalega
s1 = Student("Rahul", 21)

print(s1.name)  # Rahul
print(s1.age)   # 21
```

✅ Dekha? : Student("Rahul", 21) likhte hi __init__ apne aap chala aur name aur age set ho gaya!

OOPS in Python

1. Class:

Class ek **blueprint** hai. Jaise car banani hai, toh design banana padega pehle — wahi design **class** hai.

```
class Car:
  def __init__(self, brand, color):
    self.brand = brand
    self.color = color
  def drive(self):
    print(f"The {self.color} {self.brand} is driving.")
```

2. Object:

Object class ka **real world use** hai. Class ko use karke hum object banate hain.

```
my_car = Car("Toyota", "Red")
my_car.drive()
```

Output:

```
The Red Toyota is driving.
```

---

3. Inheritance:

Ek class doosri class ki **properties** aur **functions** use kar sakti hai. Jaise beta apne baap ki properties inherit karta hai.

```
class ElectricCar(Car):
  def __init__(self, brand, color, battery_capacity):
    super().__init__(brand, color)
    self.battery_capacity = battery_capacity

  def charge(self):
    print(f"Charging {self.brand} with {self.battery_capacity} kWh battery.")
```

python

```
tesla = ElectricCar("Tesla", "White", 100)
tesla.drive()
tesla.charge()
```

4. Polymorphism:

Same function name, different behavior.
Jaise "speak" sab animals ke liye hota hai, par dog bhaukta hai aur cat meow karti hai.

```
class Dog:
    def sound(self):
        print("Bark")

class Cat:
    def sound(self):
        print("Meow")

def animal_sound(animal):
    animal.sound()

dog = Dog()
cat = Cat()

animal_sound(dog)   # Bark
animal_sound(cat)   # Meow
```

1. Method Overloading (Not native in Python like Java)

- In languages like Java/C++, you can have multiple methods with the same name but different parameters.
- **Python does NOT support method overloading** directly.

```
class Calculator:
    def add(self, a, b=0, c=0):
        return a + b + c


calc = Calculator()
print(calc.add(2, 3))     # 5
print(calc.add(2, 3, 4))  # 9
```

2. Method Overriding

- Happens in **Inheritance**.
- A **child class** redefines a **method** that was already defined in its **parent class**.
- Python **natively supports** method overriding.

```
class Animal:
    def sound(self):
        print("Animal makes a sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")

dog = Dog()
dog.sound()
 # Output: Dog barks
```

👉 Dog overrides the sound method of Animal.

---

5. Encapsulation:

Data ko **protect** karte hain class ke andar, direct access nahi dete.

```
class BankAccount:
    def __init__(self):
        self.__balance = 0  # private variable

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance

acc = BankAccount()
acc.deposit(5000)
print(acc.get_balance())  # 5000
```

6. Abstraction: User ko sirf important cheezein dikhani, details chhupani.
(Example: Tum car drive karte ho par engine kaise kaam karta hai sab nahi dekhte.)
Python mein abstraction karne ke liye **abc** module use hota hai (advanced cheez).

```
from abc import ABC, abstractmethod


class Shape(ABC):
    @abstractmethod
    def area(self):
        pass


class Square(Shape):
    def __init__(self, side):
        self.side = side


    def area(self):
        return self.side * self.side

sq = Square(5)
print(sq.area())  # 25
```

Write a code for addition of number using Class and object in python

Method 1:

```
class Calculator:
def __init__(self, a, b):
self.a = a
self.b = b
def add (self):
return self.a + self.b
#Create two objects of the calculator class with initial values
cal1 = Calculator (10,5)
cal2 = Calculator (20,15)
#Use the add method to add two numbers
result1 = calc1.add()
result2 = calc2.add()
print (result1)
print (result2)
```

Method 2:

```
class Calculator:

def __init__(self):
    pass

def add (self, a, b):
return a + b

#Create two objects of the calculator class

cal1 = Calculator ()
cal2 = Calculator ()

#Use the add method to add two numbers

result1 = calc1.add(10, 5)
result2 = calc2.add(20, 15)

print (result1)
print (result2)
```

Method 3: USER INPUT

```
class Calculator:

def __init__(self):
    pass

def add (self, a, b):
return a + b

#Create two objects of the calculator class

cal = Calculator ()

#Get the user input

Num1 = int (“enter the first number”)
Num2 = int(“enter the second number”)

#Use the add method to add two numbers

result1 = calc1.add(Num1, Num2)

print (result1)
```

---

super ka matlab kya hai?

➡️ **super** ka use **parent class** (ya **base class**) ke methods aur properties ko **child class** (ya **derived class**) ke andar **access karne** ke liye hota hai.

Jab child class ke andar, parent class ke method ko bhi chalana ho, ya initialize karna ho → tab super() ka use karte hain.

```
class Parent:
    def __init__(self):
        print("Parent class ka constructor")

class Child(Parent):
    def __init__(self):
        super().__init__()  # Parent ka constructor bhi chalega
        print("Child class ka constructor")

# Object create karte hain
c = Child()
```

Output:

```
Parent class ka constructor
Child class ka constructor
```

Super ka use kahaan hota hai?

Situation	Use of super()
Parent class ke **constructor** ko call karne me	super().__init__()
Parent class ke **method** ko call karne me	super().method_name()
Parent class ke **property** ko access karne me	super().property_name

Ek method call ka example bhi dekh lo:

```
class Parent:
    def show(self):
        print("Parent class ka show method")
class Child(Parent):
    def show(self):
        super().show()  # Parent ka show() pehle chalega
        print("Child class ka show method")
c = Child()
c.show()
```

Output:

```
Parent class ka show method
Child class ka show method
```

👉 **Private Variables aur Private Methods Python me kya hote hain?**

- Jab hum **variable** ya **method** ka naam ke aage **double underscore** __ laga dete hain,
  toh wo **private** ban jaata hai.
- **Private** ka matlab: Class ke **bahar se** directly access **nahi** kar sakte.
  Sirf **class ke andar** hi use kar sakte ho.

```
class Example:
  def __init__(self):
    self.__private_var = 10  # private variable

  def __private_method(self):
    print("Yeh ek private method hai")

  def access_private(self):
    print(self.__private_var)
    self.__private_method()

obj = Example()

# Galat: Direct access nahi kar sakte
# print(obj.__private_var)  # ❌ Error aayega
# obj.__private_method()    # ❌ Error aayega

# Sahi: Access through public method
obj.access_private()  # ✅
```

Output:

```
10
Yeh ek private method hai
```

Python me total private nahi hote variables/methods — ek trick se access kar sakte ho:

```
print(obj._Example__private_var)  # ✅
obj._Example__private_method()    # ✅
```

*args aur **kwargs

***args:** Jab function mein kitne arguments aayenge ye fix nahi hai, tab *args use karte hain.

*args ek tuple banata hai.

Example:

```
def add_numbers(*args):
    total = 0
    for num in args:
        total += num
    print("Total:", total)


add_numbers(1, 2, 3)        # Output: Total: 6
add_numbers(10, 20, 30, 40)  # Output: Total: 100
```

Samjho:

- 3, 4 argument dene ke place mein *args likh ke kaam ho gya
- *args sab numbers ko ek **tuple** mein daal deta hai: (1, 2, 3)
- Fir loop chala ke sabka total nikal diya.

****kwargs:** Jab function mein named arguments (key-value pair) lene ho, tab **kwargs use karte hain.

**kwargs ek dictionary banata hai.

```
def show_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")


show_info(name="Anuj", age=25, city="Gurgaon")
```

Output:

```
name: Anuj
age: 25
city: Gurgaon
```

Samjho:

- **kwargs sab ko ek dictionary me convert kar deta hai: { "name": "Anuj", "age": 25, "city": "Gurgaon" }

🔥 map() aur filter() in Python

Function	Kaam
map()	Sabhi items pe **same operation apply** karta hai
filter()	Sirf **condition true** waale items ko **nikalta** hai

map() function: Map() function applies on a given function to each and every element of a list and returns the map object.

Ye har item pe **function apply** karta hai. Jo result aata hai uska ek **naya object** banata hai.

- **Syntax:** map(function, iterable)

Hamesha Map function mein do arguments aate hai, ek function khud hota hai

```
# Har number ka square lena hai
numbers = [1, 2, 3, 4, 5]
def square(n):
  return n * n
squared_numbers = map(square, numbers)
print(list(squared_numbers))
```

Output:

```
[1, 4, 9, 16, 25]
```

Samjho:

- map() har number ke upar square() function apply karega.

---

filter() function: filter() function filters elements from a list based on whether they meet a certain conditions or not (i.e., the function return true)

- Ye har item pe **function apply** karta hai **jo True return kare** unhi items ko **select** karta hai.
- Jo nahi match kare, usko hata deta hai.

```
# List of numbers
numbers = range (1, 101)
# Function to check even
def is_even(num):
  return num % 2 == 0
# Filter ke through sirf even numbers nikal rahe
even_numbers = filter(is_even, numbers)
# Filter object ko list mein convert kar rahe
print(list(even_numbers))
```

👉 **Lambda function:** lambda arguments : expression

- **arguments —** woh input jo tum dete ho (jaise x, y).
- **expression —** jo return karna hai.

Example:

Normal function:

```
def square(x):
  return x * x
print(square(5))  # Output: 25
```

Same cheez lambda se:

```
square = lambda x: x * x
print(square(5))  # Output: 25
```

Dekha?

- def nahi likhna
- Function ka naam dena optional hai (direct bhi call kar sakte ho)

Another Example:

```
numbers = range(1, 101)
even_numbers = filter(lambda num: num % 2 == 0, numbers)
print(list(even_numbers))
```

🔥 **Lambda ke bina code:**

```
# 1 se 100 tak saare numbers
numbers = range(1, 101)
# Normal function banaya to check even
def is_even(num):
  return num % 2 == 0
# filter ke andar normal function diya
even_numbers = filter(is_even, numbers)
print(list(even_numbers))
```

Add two numbers:

```
add = lambda a, b: a + b
print(add(10, 15))  # Output: 25
```

File Handling in Python

File handling in Python allows you to read from and write to files. This is important when you want to store data permanently or work with large datasets. Python provides built-in functions and methods to interact with files.

Steps for File Handling in Python:

• Opening a file

• Reading from a file

• Writing to a file

• Closing the file

1. Open a file

```
file = open('example.txt', 'r')  # 'r' = read mode
```

- 'r' → read
- 'w' → write (overwrites)
- 'a' → append (adds at the end)

2. Read a file:

```
with open('file.txt', 'r') as file:
    content = file.read()
    print(content)
```

3. Write to a file

```
file = open('example.txt', 'w')
file.write('Hello, world!\n')
file.close()
```

4. **Append to a file**

```
file = open('example.txt', 'a')
file.write('Another line.\n')
file.close()
```

OS module in Python: os ka full form hai Operating System. Python ka os module humko apne computer ke system ke sath kaam karne deta hai jaise ki:

- File aur folder banana, delete karna
- Files ke path ko sambhalna
- System ke environment variables ko access karna
- Directory change karna, dekhna, list karna
- Aur kai system-level kaam karna

Question) Use OS module to create 100 folders (folder1, folder2, ..., folder100) aur har folder ke andar ek file ho (jaise folder1 ke andar 1.txt, folder2 ke andar 2.txt)

```
import os

# Base path jahan tu folders banana chahta hai

base_path = "C:/Users/Anuj/Desktop/TestFolders"  # apne path ke hisaab se change kar lena

# Agar base folder nahi hai to bana do

if not os.path.exists(base_path):

    os.makedirs(base_path)

# 1 se 100 tak folder aur file banana

for i in range(1, 101):

    folder_name = f"folder{i}"

    folder_path = os.path.join(base_path, folder_name)

    os.makedirs(folder_path, exist_ok=True)  # exist_ok=True: agar folder hai to error nahi dega

    file_name = f"{i}.txt"

    file_path = os.path.join(folder_path, file_name)

    # File create karte hain

    with open(file_path, "w") as f:

        f.write(f"This is file {file_name} inside {folder_name}")

print("Folders and files created successfully!")
```

Important points:

- os.makedirs() ka use kiya folder banane ke liye.
- open(file_path, "w") ka use kiya file create aur likhne ke liye.
- exist_ok=True lagane se agar folder already bana hai to program rukega nahi.
- base_path apne computer ka sahi path dena (warna wo kahin aur ban jayenge).

Main boto3 Python code likh ke deta hoon jo:

- Saare IAM users list karega
- Saare S3 buckets list karega
- Ek naya S3 bucket banayega jiska naam s3_storage1 hoga

```
import boto3

# AWS ke clients

iam_client = boto3.client('iam')

s3_client = boto3.client('s3')


# 1. IAM Users list karna

print("IAM Users:")

users = iam_client.list_users()

for user in users['Users']:

   print(f" - {user['UserName']}")


# 2. S3 Buckets list karna

print("\nS3 Buckets:")

buckets = s3_client.list_buckets()

for bucket in buckets['Buckets']:

   print(f" - {bucket['Name']}")


# 3. Naya S3 bucket banana

bucket_name = 's3-storage1'  # Bucket name me underscore (_) allowed nahi hota, isiliye hyphen (-) use kiya hai

s3_client.create_bucket(

   Bucket=bucket_name,

   CreateBucketConfiguration={

      'LocationConstraint': boto3.session.Session().region_name

   }

)

print(f"\nBucket '{bucket_name}' created successfully!")
```

1. File Operations:

🔹 **Read a file:**

```
with open('file.txt', 'r') as file:
    content = file.read()
    print(content)
```

🔹 **Write to a file:**

```
with open('output.txt', 'w') as file:
    file.write('Hello, DevOps!')
```

2. API Request:

Requests is a very popular **third-party module** in Python, specially used for making **HTTP requests** — like GET, POST, PUT, DELETE etc.

🔹 **Important points about requests module:**

Feature	Details
🔧 Install Required?	**Yes** (Not built-in)
🔍 Purpose	Used for interacting with web APIs, websites, etc.
📦 Install with	pip install requests
🌐 Supports	GET, POST, headers, auth, params, JSON, etc.
💥 Common Error	ModuleNotFoundError: No module named 'requests' if not installed

✅ **To install requests:**

Run this in your terminal:

```
pip install requests
```

Or, if you're using Python 3 specifically:

```
pip3 install requests
```

✅ **Example use:**

```
import requests

response = requests.get("https://api.github.com")

print(response.status_code)

print(response.json())
```

3. JSON Handling:

json ek built-in Python module hai. Iska use hum JSON (JavaScript Object Notation) data ko Python objects mein convert karne ke liye karte hain, ya phir Python objects ko JSON format mein convert karne ke liye.

Iske kuch common functions hain:

- json.load(): File se JSON data read karne ke liye.
- json.loads(): String se JSON data ko Python object mein convert karne ke liye.
- json.dump(): Python object ko file mein JSON format mein likhne ke liye.
- json.dumps(): Python object ko JSON string mein convert karne ke liye.

Read JSON data:

```
import json
with open('data.json', 'r') as file:
  data = json.load(file)
  print(data)
```

Write JSON to a file:

```
import json
data = {'name': 'DevOps', 'type': 'Workflow'}
with open('output.json', 'w') as file:
  json.dump(data, file, indent=4)      //OUTPUT:    "name": "DevOps", "type": "Workflow"
```

4. Working with Databases:

```
import sqlite3
# Connect to SQLite database (it will create the database if it doesn't exist)
conn = sqlite3.connect('example.db')
# Create a cursor object using the connection
cursor = conn.cursor()
# Execute a query to create a table 'users' if it doesn't exist already
cursor.execute('CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)')
# Commit the changes to the database (save the changes)
conn.commit()
# Close the connection
conn.close()
```

Explanation:

1. **sqlite3.connect('example.db')**: Isse SQLite database file example.db ke saath connection establish hota hai. Agar file pehle se exist nahi karti, toh yeh automatically create ho jaata hai.
2. **conn.cursor()**: Yeh ek cursor object create karta hai jo SQL queries execute karne ke liye use hota hai.
3. **cursor.execute('CREATE TABLE IF NOT EXISTS users ...')**: Yeh query database mein users naam ka table create karti hai, agar woh pehle se nahi hai.
4. **conn.commit()**: Yeh changes ko database mein save karne ke liye use hota hai.
5. **conn.close()**: Yeh connection ko close karta hai jab kaam ho jaata hai.

Is code se **example.db** database mein ek users table create ho jaayega, jisme id (primary key) aur name columns honge.

Agar data insert karna ho ya query run karni ho, toh tu cursor ke through aur conn.commit() ke baad us data ko save kar sakta hai.

5. Error Handling in python:

Python mein error handling ke liye hum try, except, else, aur finally ka use karte hain. Yeh errors ko handle karne mein madad karta hai, taaki program crash na ho.

```
try:
    # Code that may raise an exception
    x = 1 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError as e:
    # Code to handle the exception
    print(f"Error: {e}")
else:
    # Code to run if there is no exception
    print("No error occurred.")
finally:
    # Code that will always run, no matter what
    print("This will always execute.")
```

Explanation:

1. **try**: Is block mein wo code likhte hain jo exception raise kar sakta hai.
2. **except**: Agar try block mein koi error aata hai, toh yeh block execute hota hai. Aap specific errors ko handle kar sakte ho, jaise ZeroDivisionError, FileNotFoundError, etc.
3. **else**: Agar try block mein koi exception nahi aata, toh yeh block execute hota hai.
4. **finally**: Yeh block hamesha execute hota hai, chahe error aaye ya na aaye. Yeh typically cleanup tasks ke liye use hota hai (e.g., closing files or database connections).

6. Docker Integration: Docker ko Python se integrate karne ke liye tum docker package use kar sakte ho. Isse tum Docker containers ko manage kar sakte ho, images ko build, run aur stop kar sakte ho, aur bhi bohot kuch.

Running a Container

```
import docker

client = docker.from_env()

# 'hello-world' image se ek container run karna

container = client.containers.run("hello-world", detach=True)

# Container ka ID print karna

print(container.id)
```

List Running Containers

```
import docker

# Docker client initialize karna

client = docker.from_env()

# Running containers ki list lena

containers = client.containers.list()

# Containers ke IDs print karna

for container in containers:

    print(container.id)
```

Stopping a Container

```
import docker

# Docker client initialize karna

client = docker.from_env()

# Running container ka ID jaise ki 'container_id' dena

container = client.containers.get("container_id")

# Container ko stop karna

container.stop()

print("Container stopped.")
```

Building a Docker Image from a Dockerfile

```
import docker

# Docker client initialize karna

client = docker.from_env()

# Dockerfile ke path ka mention karna aur image build karna

image = client.images.build(path=".", tag="myimage:latest")

print("Image built successfully.")
```

Removing a Container

```
import docker

# Docker client initialize karna

client = docker.from_env()

# Running container ko remove karna

container = client.containers.get("container_id")

container.remove()

print("Container removed.")
```

Container Health Checks: Check the health of a running Docker container

```
import docker

# Create a Docker client from environment variables

client = docker.from_env()

# Get the container by its ID (replace 'container_id' with your container's actual ID)

container = client.containers.get('container_id')

# Print the health status of the container

health_status = container.attrs['State']['Health']['Status']

print(f"Health status of the container: {health_status}")
```

7. Monitoring System Resources: psutil ek powerful library hai jo system resources jaise CPU, memory, disk usage, aur network stats ko monitor karne mein help karti hai.

Agar tum psutil ka use karna chahte ho, toh sabse pehle tumhe is library ko install karna padega:

```
pip install psutil
```

Using psutil to monitor system resources:

```
import psutil

# CPU usage ka percentage nikaalna

print(f"CPU Usage: {psutil.cpu_percent()}%")

# Virtual memory (RAM) usage ka percentage nikaalna

print(f"Memory Usage: {psutil.virtual_memory().percent}%")
```

Explanation:

1. psutil.cpu_percent(): Is function ka use karke tum current CPU usage ka percentage le sakte ho. Yeh time interval ko track karta hai aur phir CPU usage ka average return karta hai.
2. psutil.virtual_memory().percent: Yeh system ki virtual memory (RAM) ka usage percentage return karta hai. Agar tum system ke memory resources ka status dekhna chahte ho, toh yeh useful hai.

Additional Examples:

1. CPU Core Usage:

Agar tum chahte ho ki har ek CPU core ka usage dekhna ho:

```
cpu_usage_per_core = psutil.cpu_percent(percpu=True)

print(f"CPU Usage per core: {cpu_usage_per_core}")
```

2. Disk Usage:

System ke disk ka usage dekhne ke liye:

```
disk_usage = psutil.disk_usage('/')

print(f"Disk Usage: {disk_usage.percent}%")
```

3. Network Stats:

Network statistics bhi monitor kar sakte ho:

```
net_io = psutil.net_io_counters()

print(f"Bytes Sent: {net_io.bytes_sent}, Bytes Received: {net_io.bytes_recv}")
```

8. Scheduling Tasks:

```
import schedule

import time

# Job function jo scheduled task ko perform karega

def job():

   print("Running scheduled job...")

# 1 minute ke interval pe job ko schedule karna
```

```
schedule.every(1).minutes.do(job)

# Infinite loop to keep checking and running the scheduled jobs

while True:

  schedule.run_pending()  # Run any pending jobs

  time.sleep(1)  # Wait for 1 second before checking again
```

Explanation:

1. **schedule.every(1).minutes.do(job)**: Is line se **job()** function ko har 1 minute ke interval pe run karne ke liye schedule karte hain.
2. **schedule.run_pending()**: Yeh function pending tasks ko execute karta hai, jo scheduled hain.
3. **time.sleep(1)**: Yeh line ensures karti hai ki program har second ke baad check kare ki koi task pending hai ya nahi.

9. Integrating with CI/CD Tools:

```
import requests

# Jenkins job trigger karne ka URL

url = 'http://your-jenkins-url/job/your-job-name/build'

# POST request bhejna Jenkins job ko trigger karne ke liye

response = requests.post(url, auth=('user', 'token'))

# Response ka status code print karna

print(response.status_code)
```

10. Using Python for Infrastructure as Code:

```
import boto3

# Create an EC2 resource object

ec2 = boto3.resource('ec2')

# Filter for running EC2 instances

instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])

# Loop through the instances and print their ID and state

for instance in instances:

  print(instance.id, instance.state['Name'])
```

Explanation:

- boto3.resource('ec2'): Creates a resource object for EC2.
- ec2.instances.filter(...): Filters instances with the state "running".
- for instance in instances: Iterates through each running instance and prints its ID and current state.

11. Automating AWS S3 Operations

- **Upload a File to S3:**

```
import boto3

# Create an S3 client

s3 = boto3.client('s3')

# Specify the file you want to upload

file_name = 'path/to/your/local/file.txt'

bucket_name = 'your-bucket-name'

object_name = 'uploaded-file.txt'


# Upload the file to S3

s3.upload_file(file_name, bucket_name, object_name)

print(f"File {file_name} uploaded to {bucket_name}/{object_name}.")
```

- **Download a File from S3**

```
import boto3

# Create an S3 client

s3 = boto3.client('s3')

# Specify the S3 bucket and the object to download

bucket_name = 'your-bucket-name'

object_name = 'uploaded-file.txt'

file_name = 'path/to/your/local/destination/file.txt'

# Download the file from S3

s3.download_file(bucket_name, object_name, file_name)

print(f"File {object_name} downloaded from {bucket_name} to {file_name}.")
```

Explanation:

1. **Uploading a file**:
    - s3.upload_file(file_name, bucket_name, object_name) uploads the file from your local system to the specified S3 bucket.
2. **Downloading a file**:
    - s3.download_file(bucket_name, object_name, file_name) downloads the file from the S3 bucket to your local system.

12. Terraform Execution: Executing Terraform commands with subprocess:

```
import subprocess

# Initialize Terraform

subprocess.run(['terraform', 'init'], check=True)

# Apply configuration (with auto-approval)

subprocess.run(['terraform', 'apply', '-auto-approve'], check=True)
```

Explanation:

1. **subprocess.run()**: This function runs a specified command in a subprocess. The arguments are passed as a list of strings, where each string represents part of the command.
    - 'terraform init': Initializes your Terraform environment, downloads necessary plugins, and sets up the working directory.
    - 'terraform apply': Applies the Terraform configuration to the infrastructure.
    - The -auto-approve flag automatically approves the execution without requiring user input.
2. **check=True**: Ensures that if the Terraform command fails (returns a non-zero exit status), it raises a subprocess.CalledProcessError. This helps handle errors gracefully.

13. Working with Prometheus Metrics

```
import requests

# Send a GET request to Prometheus metrics endpoint

response = requests.get('http://localhost:9090/metrics')

# Split the response text by lines

metrics = response.text.splitlines()

# Iterate over each metric and print it

for metric in metrics:

  print(metric)
```

Explanation:

1. **requests.get('http://localhost:9090/metrics')**: Sends an HTTP GET request to the Prometheus metrics endpoint, where it exposes metrics in plain text format.
2. **response.text.splitlines()**: Splits the raw text response by newlines to get each individual metric.
3. **for metric in metrics:**: Loops through each metric line and prints it.

14. Serverless Applications with AWS Lambda: Deploying a simple AWS Lambda function:

Here's a simple AWS Lambda function that returns a "Hello from Lambda!" message in the response. This function can be deployed to AWS Lambda to be triggered by events like HTTP requests, S3 events, or more.

```
import json

def lambda_handler(event, context):
    # Lambda function logic
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

15. Creating and Sending Emails with smtplib: Send an email using Python

```
import smtplib
from email.mime.text import MIMEText
# Email details
sender = 'you@example.com'
recipient = 'recipient@example.com'
subject = 'Test Email'
body = 'This is a test email.'
# Create the MIMEText message
msg = MIMEText(body)
msg['Subject'] = subject
msg['From'] = sender
msg['To'] = recipient


# Send the email
with smtplib.SMTP('smtp.example.com') as server:
    # Log in to the SMTP server
    server.login('username', 'password')
    # Send the email
    server.send_message(msg)
print("Email sent successfully!")
```

16. Managing Kubernetes Resources with kubectl: Using subprocess to interact with Kubernetes

```
import subprocess

# Get list of pods

subprocess.run(['kubectl', 'get', 'pods'])

# Apply a Kubernetes configuration (e.g., a deployment)

subprocess.run(['kubectl', 'apply', '-f', 'deployment.yaml'])
```