Key words

Keywords are the pre-defined words given by the interpreter and each keyword as their own identity and performs a specific task.

we cannot use keywords for variable names or any other functions because the interpreter identifies and treated it like a keyword

**AND:**

AND keyword is a logical operator and it is used to perform logical AND operations the AND keyword is used to check the two conditions if both the statements are true then the condition is true otherwise false.

a = 50  
b = 40  
c = 10  
if a > b and b > c :  
 print("Both conditions are true")

**#Output :** Both conditions are true

**OR :**

The or keyword is a logical operator in Python that returns True if at least one of the operands is True. It's often used to create compound conditions in if statements.

Example:

x = 10  
y = 5  
  
if x > 5 or y > 10:  
 print("At least one condition is True")  
else:  
 print("Both conditions are False")

output: At least one condition is True

**NOT :**

The not operator in Python is a logical operator that inverts the truth value of an expression. In simpler terms, it flips the boolean value from True to False and viceversa.

**Example:**

x = True  
  
if not x:  
 print("x is False")  
else:  
 print("x is True")

output: x is True

**IF:**

The if statement is a fundamental control flow statement in Python used to execute a block of code only if a certain condition is True

**Syntax:**

Python

if condition:

# code to be executed if condition is True

**Example:**

age = 18  
  
if age >= 18:  
 print("You are an adult.")

#output:You are an adult.

**ELSE:**

The else statement is often used in conjunction with the if statement to execute a block of code when the if condition is False.

**Syntax:**

if condition:

# code to be executed if condition is True

else:

# code to be executed if condition is False

**Example:**

age = 15  
  
if age >= 18:  
 print("You are an adult.")  
else:  
 print("You are a minor.")  
#output:  
#You are an adult.  
#You are a minor.

**ELIF:**

The elif statement, short for "else if," is used to check multiple conditions sequentially. It allows you to create more complex decision-making structures.

**Syntax:**

Python

if condition1:

# code to be executed if condition1 is True

elif condition2:

# code to be executed if condition1 is False and condition2 is True

elif condition3:

# code to be executed if condition1 and condition2 are False, but condition3 is True

else:

# code to be executed if none of the above conditions are True

**Example:**

number = 10  
  
if number > 0:  
 print("Positive number")  
elif number == 0:  
 print("Zero")  
else:  
 print("Negative number")  
 #outpt: positive number

**WHILE :**

The while loop is a control flow statement that repeatedly executes a block of code as long as a given condition 1 remains True

**Syntax:**

while condition:

# code to be executed while the condition is True

Example:

count = 1  
while count <= 5:  
 print(count)  
 count += 1

output:

1

2

3

4

5

**FOR:**

A for loop in Python is a control flow statement that iterates over a sequence (like a list, tuple, string, or range) and executes a block of code for each item in the sequence.

**Syntax:**

for item in sequence:

# code to be executed for each item

**Example:**

Names = ["Archana", "Rachana", "Chaithu"]  
for Name in Names:  
 print(Name)

Output:

Archana

Rachana

Chaithu

IN:

To use a for loop in Python, you'll typically iterate over a sequence of items. Here are some common use cases.

**Example:**

Names = ['Archana', 'Rachana', 'Chaithu', 'Thrishan']  
if 'Madhu' in Names :  
 print("Madhu is present")  
else :  
 print("not present")

**Outpu**t: not present

**TRY:**

In Python, the try-except block is a powerful tool for handling exceptions gracefully. It allows you to anticipate potential errors in your code and execute alternative actions or provide informative messages to the user.

**Syntax:**

try:

# Code that might raise an exception

except ExceptionType:

# Code to handle the specific exception

**Example:**

a="10"  
b= -10  
try :  
 c=a+b  
 print("add :",c)  
except Exception as Archana :  
 print("Archana")  
 print("add by 0 NOT-ok in python")

**output:**

Archana

add by 0 NOT-ok in python

**EXCEPT:**

In Python, the except keyword is used within a try-except block to handle exceptions that may arise during the execution of code.

#Except:  
a="10"  
b= -10  
try :  
 c=a+b  
 print("add :",c)  
except Exception as Archana :  
 print("Archana")  
 print("add by 0 NOT-ok in python")

**. FINALLY :**

The finally keyword is used in try, except blocks. It defines a block of code to run when the try, except, else block is final. The finally block will be executed no matter if the try block raises an error or not.

marks = 120

marks = 90

try :

if(marks>100):

raise Exception("marks are greater than 100")

print("given marks:",marks)

except Exception as e :

print(e)

finally :

print("Finally block is executed")

Output :

given marks: 90

Finally block is executed

**. DEF :**

It is used to define a function it is reusable block of code, functions takes input and perform actions and they return outputs

def sum (a, b) :

return a + b

print(sum(1,6))

**Output:7**

**RETURN:**

1. **RETURN:**

* The RETURN keyword is used to exit a function and send a value(or values) back to the function’s call.
* To provide multiple outputs from a single function.

**Syntax: return [expression]**

Ex:def square(num):

return num \* num

result = square(5)

print(result)

Output: 25

1. **IMPORT**

* Import keyword is used to include modules in your python program.
* Import pandas as pd
* Import numpy as np
* Import math as m

**Syntax: import module\_name**

Ex: import numpy as np

import math  
  
result = math.sqrt(16)  
print(result) # Output: 4.0

**CLASS**

* The keyword CLASS is used to define a class.
* It will give the blue print of an object.
* Classes are fundamental to object-oriented programming (OOP) in Python.

Syntax: class ClassName:

def \_\_init\_\_(self, parameter1, parameter2, ...):

# Constructor to initialize attributes

self.attribute1 = parameter1

self.attribute2 = parameter2

...

Exclass Flower:  
 def \_\_init\_\_(self, color, type):  
 self.color = color  
 self.type = type  
  
 def describe\_flower(self):  
 print(f"This is a {self.color} {self.type} flower.")  
  
# Creating flower instances  
rose = Flower("Red", "Rose")  
lily = Flower("White", "Lily")  
  
# Describing the flowers  
rose.describe\_flower() # Output: This is a Red Rose flower.  
lily.describe\_flower() # Output: This is a White Lily flower.

FROM

* From keyword is used to import the module or library.
* It allows you to import only the functions, classes, or variables.

Syntax:

* From module\_name import specific\_func

We can also import multiple items:

* From modul\_name import func1, func2

Ex: from math import sqrt

from math import sqrt  
  
result = sqrt(25)  
print("The square root of 25 is", result)

**AS**

* The as keyword is used to create an alias for a module, function, or object.
* The as will give a module name as shorter name(alias) to make code easier to write.

Syntax: import module\_name as alias\_name

Ex

import numpy as np

arr = np.array([1, 2, 3])

print(arr)

Output: [1 2 3]

1. **TRUE**

In python, True is a Boolean value

Ex: x = 20  
  
if x > 15:  
 print("True")  
else:  
 print("False")

Output:True

**FALSE**

In python, False is a Boolean value.

Ex: x = 20

if x > 15:

print("True")

else:

print("False")

output: True

**NONE**

The keyword NONE represents the absence of a value or a null value.

Syntax: None

**Ex:**

x = None  
  
if x is None:  
 print("x is None")  
else:  
 print("x is not None")  
  
#output: x is None

**IS**

* The IS keyword is used to check if two objects refer to the same memory location.
* Is: It checks whether two variables point to the same object.

Syntax: object1 is object2

Ex: a = 10

b = 10

print(a is b)

O/P: True (both refer to the same memory location for 10)

a=10

b=20

Print(a is b)

O/P: False

**LAMBDA**

The LAMBDA keyword in Python is used to create anonymous (nameless) functions. These functions are also called lambda functions.

Syntax: lambda arguments: expression

Ex: add=lambda x : x+10

print(add(5))

O/P: 15

# adding two numbers to the function

Add= lambda x , y : X + Y

Print(add(5,4))

O/P: 9

**WITH**

* The WITH keyword in Python is used to simplify the management of resources, such as file handling.
* This ensures that the file is properly closed after it is used, even if an error occurs during the read or write operation.

Syntax: with expression as variable:

class MyContextManager:  
 def \_\_enter\_\_(self):  
 print("Entering the context")  
 return self  
  
 def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):  
 print("Exiting the context")  
  
# Using 'with' with a custom context manager  
with MyContextManager() as manager:  
 print("Inside the context")

output:

Entering the context

Inside the context

Exiting the context

**GLOBAL**

* In Python, the GLOBAL keyword allows us to modify the variable outside of the current scope.
* It is used to create a global variable and make changes to the variable in a local context.

Syntax: global variable\_name

c = 10 # Initialize c globally  
  
def add():  
 global c # Using the global keyword to modify the global variable c  
 c = c + 2  
 print(c)  
  
add() # Calling the add function  
#output: 12

**NON-LOCAL**

* The NON-LOCAL keyword in Python is used to access and modify a variable in an outer function from an inner function.
* It is used to work with nested functions.

Syntax: nonlocal variable\_name

Ex: x = 5 # Global x  
  
def name():  
 x = 10 # Local x inside the function  
 print("Non-local x:", x) # This prints the local x inside the function  
  
name() # Calling the function  
print("global x:", x) # This prints the global x  
#output:   
#Non-local x: 10  
#global x: 5