**LECTURE-04 (Sunday 03-May-2025)**

* Linear programming: code is written in a straight, sequential flow means if you want to print any text 1000 times then copy code “print(text)” and paste 1000 times and then run.
* Structured programming: control blocks. It emphasizes the use of functions, loops, and conditional statements to improve code clarity and structure. (if, else, while, for)
* Object Oriented Programming: concept of "objects" which contains data and behavior (function). (classes, objects)
* Symbolic AI (first type of AI) was achieved by if-else conditions.
* <https://github.com/panaversity/learn-modern-ai-python/tree/main/00_python_colab/05_control_flow>
* Control block (if-else): whenever and whatever we want, can be executed.
* In Python, every controlled block have reserve keyword (if, else, elif, for, while, try, catch) and we cannot create variable names using reserve keyword names.
* Identify control block:

1. We place colon at the end of line of control block like

if logic :

else :

for logic :

1. at next line, indentation is present like

if logic :

block\_body

else :

block\_body

for logic :

block\_body

Here, for all control blocks, logic always return boolean value True/False.

Example 1:

print("line1")

print("line2")

print("line3")

print("line4")

if False:

  print("line5")

  print("line6")

print("line7")

Output:

line1

line2

line3

line4

line7

Here, text “line5” and “line6” are not printed because they are inside “if” control block and “if” condition is “False”.

OR

print("line1")

print("line2")

print("line3")

print("line4")

if True:

  print("line5")

  print("line6")

print("line7")

Output:

line1

line2

line3

line4

line5

line6

line7

Now, text “line5” and “line6” are printed because “if” condition is “True”.

* In VSCode, 4 spaces are used for indentation while in Google Colab, 2 spaces.
* If-else block:

if Logic :

True block

else:

False block

Example 1:

if False:

  print("ABC")

else:

  print("DEF")

Output:

DEF

OR

if True:

  print("ABC")

else:

  print("DEF")

Output:

ABC

* If-elif-else : Chain of blocks, based on priorities. It is very important to take of priorities while implementing if-elif-else blocks because we have to keep sequence of blocks on the basis of priorities as if any condition found true then that condition’s block executed and remaining conditions are not checked.

Example 1:

if False:

  print("line number 1")

elif True:

  print("line number 2")

elif False:

  print("line number 3")

elif True:

  print("line number 4")

else:

  print("line number 5")

Output:

line number 2

* Lines with same indentation after colon will be considered as control block.

if False:

  print("line number 1")

elif False:

  print("line number 2")

elif True:

  print("line number 3")

elif False:

  print("line number 4")

print("line number 5")

Output:

line number 3

line number 5

Here, “line number 5” print statement is considered out of if-elif block

* “if” block is starting block which can be ended by “else” or “elif” block. Also, “if” block not needed any block to end.

Example 1 (if-elif-else):

if True:

  print("line number 1")

elif False:

  print("line number 2")

elif True:

  print("line number 3")

elif False:

  print("line number 4")

else:

  print("line number 5")

if True:

  print("line number 6")

Output:

line number 1

line number 6

Here, last “if” is not related to above “if-elif-else” blocks.

Example 2 (if-elif):

if False:

  print("line number 1")

elif False:

  print("line number 4")

if True:

  print("line number 6")

Ouput:

line number 6

Example 3 (if-else):

if False:

  print("line number 1")

else:

  print("line number 4")

if True:

  print("line number 6")

Output:

line number 4

line number 6

Example 4 (if):

if True:

  print("line number 1")

if True:

  print("line number 6")

Output:

line number 1

line number 6

* Comprehensive Style:

Example:

"abc" if False else "def"

Output:

def

OR

"abc" if True else "def"

Output:

abc

* Nested “if-else” block means dependents.

Example:

if True:

  if True:

    print("abc")

  else:

    print("def")

else:

  print("ghi")

Output:

abc

OR

if True:

  if False:

    print("abc")

  else:

    print("def")

else:

  print("ghi")

Output:

def

Example:

percentage : int = int(input("Enter the percentage : "))

print(percentage, type(percentage))

"Pass" if percentage >= 80 else "Fail"

#or

if percentage >= 80:

    print("Pass")

else:

    print("Fail")

if percentage >= 80:

    print("A1")

elif percentage >= 70:

    print("A")

elif percentage >= 60:

    print("B")

elif percentage >= 50:

    print("C")

elif percentage >= 40:

    print("D")

elif percentage >= 33:

    print("E")

else:

    print("F")

Output:

Enter the percentage : 77

77 <class 'int'>

Fail

A

OR

Output:

Enter the percentage : 95

95 <class 'int'>

Pass

A1

OR

Above code, can be rewrite using “and” operator:

percentage : int = int(input("Enter the percentage : "))

if percentage >= 33 and percentage < 40:

    print("E")

elif percentage >= 40 and percentage < 50:

    print("D")

elif percentage >= 50 and percentage < 60:

    print("C")

elif percentage >= 60 and percentage < 70:

    print("B")

elif percentage >= 70 and percentage < 80:

    print("A")

elif percentage >= 80:

    print("A1")

else:

    print("F")

Output:

Enter the percentage : 65

B

* At Google Colab, “Ctrl” + “K” + “C” – shortcut key is used to comment code while “Ctrl” + “K” + “C” – shortcut key is used to comment code
* Class Assignment

Validate value of below fields from user input.

user = "admin"

password = "admin"

age = 22

otp = "123"

Code 1 using nested if-else conditions:

user : str = "admin"

password : str = "admin"

age : int = 22

otp : str = "123"

provided\_user : str = ""

provided\_password : str = ""

provided\_age : int = 0

provided\_otp : str = ""

provided\_user = input("Enter user : ")

if provided\_user == user:

  provided\_password = input("Enter password : ")

  if provided\_password == password:

    provided\_age = int(input("Enter age : "))

    if provided\_age == age:

      provided\_otp = input("Enter otp : ")

      if provided\_otp == otp:

        print("Login successful.")

      else:

        print("Login failed.")

    else:

      print("Invalid age")

      provided\_age = int(input("Last attempt to enter valid age. Enter age : "))

      if provided\_age == age:

        provided\_otp = input("Enter otp : ")

        if provided\_otp == otp:

          print("Login successful.")

        else:

          print("Login failed.")

      else:

          print("Login failed.")

  else:

    print("Invalid password")

    provided\_password = input("Last attempt to enter valid password. Enter password : ")

    if provided\_password == password:

      provided\_age = int(input("Enter age : "))

      if provided\_age == age:

        provided\_otp = input("Enter otp : ")

        if provided\_otp == otp:

          print("Login successful.")

        else:

            print("Login failed.")

      else:

        print("Invalid age")

        provided\_age = int(input("Last attempt to enter valid age. Enter age : "))

        if provided\_age == age:

          provided\_otp = input("Enter otp : ")

          if provided\_otp == otp:

            print("Login successful.")

          else:

            print("Login failed.")

        else:

          print("Login failed.")

    else:

      print("Login failed.")

else:

  print("Invalid user found")

  provided\_user = input("Last attempt to enter valid user. Enter user : ")

  if provided\_user == user:

    provided\_password = input("Enter password : ")

    if provided\_password == password:

      provided\_age = int(input("Enter age : "))

      if provided\_age == age:

        provided\_otp = input("Enter otp : ")

        if provided\_otp == otp:

          print("Login successful.")

        else:

          print("Login failed.")

      else:

        print("Invalid age")

        provided\_age = int(input("Last attempt to enter valid age. Enter age : "))

        if provided\_age == age:

          provided\_otp = input("Enter otp : ")

          if provided\_otp == otp:

            print("Login successful.")

          else:

            print("Login failed.")

  else:

    print("Login failed.")

Output (Once user can pass wrong value of fields “user”, “password” and “age” while “otp” should be correct in first attempt):

Enter user : a

Invalid user found

Last attempt to enter valid user. Enter user : a

Login failed.

OR

Enter user : admin

Enter password : a

Invalid password

Last attempt to enter valid password. Enter password : a

Login failed.

OR

Enter user : admin

Enter password : admin

Enter age : 5

Invalid age

Last attempt to enter valid age. Enter age : 8

Login failed.

OR

Enter user : admin

Enter password : admin

Enter age : 22

Enter otp : 4

Login failed.

OR

Enter user : admin

Enter password : admin

Enter age : 22

Enter otp : 123

Login successful.

Code 2 using method recursion (method calls itself):

user : str = "admin"

password : str = "admin"

age : int = 22

otp : str = "123"

def get\_value\_from\_user(field\_name : str, expected\_value : str) :

  provided\_value : str = input(f"Enter {field\_name} : ")

  if(provided\_value == expected\_value):

    return provided\_value

  else:

    print(f"Invalid {field\_name}. Please provide valid {field\_name}.")

    return get\_value\_from\_user(field\_name, expected\_value)

print(f"Provided user : {get\_value\_from\_user('user', user)}")

print(f"Provided password : {get\_value\_from\_user('password', password)}")

print(f"Provided age : {get\_value\_from\_user('age', str(age))}")

print(f"Provided otp : {get\_value\_from\_user('otp', str(otp))}")

print("Login Successfully.")

Output:

Enter user : a

Invalid user. Please provide valid user.

Enter user : a

Invalid user. Please provide valid user.

Enter user : admin

Provided user : admin

Enter password : g

Invalid password. Please provide valid password.

Enter password : h

Invalid password. Please provide valid password.

Enter password : admin

Provided password : admin

Enter age : e

Invalid age. Please provide valid age.

Enter age : 6

Invalid age. Please provide valid age.

Enter age : 5

Invalid age. Please provide valid age.

Enter age : 22

Provided age : 22

Enter otp : 5

Invalid otp. Please provide valid otp.

Enter otp : 6

Invalid otp. Please provide valid otp.

Enter otp : 123

Provided otp : 123

Login Successfully.

* Validate Student ID:

Code using nested if:

provided\_student\_id : int = int(input("Enter your student id : "))

provided\_student\_cnic : int = int(input("Enter your student cnic : "))

valid\_student\_id : int = 123

valid\_student\_cnic : int = 456

if provided\_student\_id == valid\_student\_id:

  if provided\_student\_cnic == valid\_student\_cnic:

    print("Access granted")

else:

  print("Access denied")

Output:

Enter your student id : 123

Enter your student cnic : 456

Access granted

OR

Enter your student id : 23

Enter your student cnic : 45

Access denied

Code using and operator:

#id check

provided\_student\_id : int = int(input("Enter your student id : "))

provided\_student\_cnic : int = int(input("Enter your student cnic : "))

valid\_student\_id : int = 123

valid\_student\_cnic : int = 456

if provided\_student\_id == valid\_student\_id and provided\_student\_cnic == valid\_student\_cnic:

  print("Access granted")

else:

  print("Access denied")

Output:

Enter your student id : 3

Enter your student cnic : 3

Access denied

OR

Enter your student id : 123

Enter your student cnic : 456

Access granted

* “in” operator: It is used to check if a value exists within an iterable (like a list, string, tuple, set, or dictionary). It returns True if the value found, and False otherwise.

Example check if provided student is present in blacklisted students:

#id check

provided\_student\_id : int = int(input("Enter your student id : "))

provided\_student\_cnic : int = int(input("Enter your student cnic : "))

valid\_student\_id : int = 123

valid\_student\_cnic : int = 456

blacklisted\_id : int = [10,34,56, 123]

if provided\_student\_id == valid\_student\_id and provided\_student\_cnic == valid\_student\_cnic:

  if provided\_student\_id in blacklisted\_id:

    print("Student Id is Blacklisted")

  else:

    print("Access granted")

else:

    print("Access denied")

Output:

Enter your student id : 123

Enter your student cnic : 456

Student Id is Blacklisted

* “not in” operator: It is used to check whether a value is not present in a sequence (like a list, tuple, string, or dictionary keys). It returns True if the value is not found, and False otherwise.

Example 1:

fruits\_available\_in\_store : list = ["apple", "banana", "cherry"]

required\_fruit : str = input("Please provide fruit name which you need : ")

if required\_fruit not in fruits\_available\_in\_store:

  print("Your required fruit is not available in store")

else:

  print("Your required fruit is available in store")

Output:

Please provide fruit name which you need : apple

Your required fruit is available in store

OR

Please provide fruit name which you need : grapes

Your required fruit is not available in store

OR

Please provide fruit name which you need : Apple

Your required fruit is not available in store

But here, “apple” is present in store but we have provided input “Apple”. To fix this issue add “lower” method with “required\_fruit” while comparing:

fruits\_available\_in\_store : list = ["apple", "banana", "cherry"]

required\_fruit : str = input("Please provide fruit name which you need : ")

if required\_fruit.lower() not in fruits\_available\_in\_store:

  print("Your required fruit is not available in store")

else:

  print("Your required fruit is available in store")

Output:

Please provide fruit name which you need : APPLE

Your required fruit is available in store

* Prompt : Build a calculator which takes input from the user, beside basic functionality include modulus , floor division, exponentiation

Code:

# prompt: Build a calculator which takes input from the user, beside basic functionality include modulus , floor division, exponentiation

def calculator():

    """Performs basic arithmetic operations and modulus, floor division, exponentiation."""

    try:

        num1 = float(input("Enter the first number: "))

        op = input("Enter the operator (+, -, \*, /, %, //, \*\*): ")

        num2 = float(input("Enter the second number: "))

        if op == "+":

            result = num1 + num2

        elif op == "-":

            result = num1 - num2

        elif op == "\*":

            result = num1 \* num2

        elif op == "/":

            if num2 == 0:

                raise ZeroDivisionError("Cannot divide by zero")

            result = num1 / num2

        elif op == "%":

            result = num1 % num2

        elif op == "//":

            if num2 == 0:

              raise ZeroDivisionError("Cannot perform floor division by zero")

            result = num1 // num2

        elif op == "\*\*":

            result = num1 \*\* num2

        else:

            raise ValueError("Invalid operator")

        print("Result:", result)

    except ValueError as ve:

        print(f"Error: {ve}")

    except ZeroDivisionError as zde:

        print(f"Error: {zde}")

    except Exception as e:

        print(f"An unexpected error occurred: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    calculator()

Output:

Enter the first number: 10

Enter the operator (+, -, \*, /, %, //, \*\*): /

Enter the second number: 3

Result: 3.3333333333333335

* Scope: It refers to the region of a program where a particular variable is recognized and accessible. It determines the visibility and lifetime of variables.

Example 1:

global\_variable = 55

def wow():

  local\_variable = 30

print(global\_variable)

print(local\_variable)

Output: global printed but local error because “local\_variable” scope is within the method “wow”

55

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

NameError Traceback (most recent call last)

[<ipython-input-4-3b526389ace2>](https://localhost:8080/) in <cell line: 0>()

**6**

**7** print(global\_variable)

----> 8 print(local\_variable)

NameError: name 'local\_variable' is not defined

* Types of Scope:

1. Local Scope (L): variables defined inside a function. They are accessible only within that function.

Example:

def my\_function():

    x = 10  # local scope

    print("Calling x from my\_function:", x)

my\_function()

Output:

Calling x from my\_function: 10

1. Enclosing Scope (E): This refers to the scope of any enclosing functions, which is relevant in nested functions.

Example:

def outer():

    y = 20  # enclosing scope for inner()

    def inner():

        print("Calling y from inner : ", y)

    inner()

outer()

Output:

Calling y from inner : 20

1. Global Scope (G): variables defined at the top level of a script or module. Accessible anywhere in the module unless shadowed.

Example:

z = 30  # global scope

def show():

    print("Calling z from show : ", z)

show()

Output:

Calling z from show : 30

* Shadowing: It happens when a variable in a local or inner scope has the same name as a variable in an outer scope. The inner variable "shadows" or hides the outer one within its scope.

Example:

x = "global"  # Global scope

def outer():

    x = "enclosing"  # Enclosing scope (shadows the global x)

    def inner():

        x = "local"  # Local scope (shadows enclosing x)

        print("Calling x from inner : " , x)     # --> "local"

    inner()

    print("Calling x from outer : " , x)         # --> "enclosing"

outer()

print("Calling x from main : " , x)             # --> "global"

Output:

Calling x from inner : local

Calling x from outer : enclosing

Calling x from main : global

Here, "local" shadows "enclosing" inside inner() while "enclosing" shadows "global" inside outer()

* Prompt : generate a grading system for school which takes marks as an input show grade according to marks

Code:

# prompt:  generate a grading system for school which takes marks as an input show grade according to marks

def get\_grade(marks):

  """

  This function takes marks as input and returns the corresponding grade.

  """

  if marks >= 90:

    return "A+"

  elif marks >= 80:

    return "A"

  elif marks >= 70:

    return "B"

  elif marks >= 60:

    return "C"

  elif marks >= 50:

    return "D"

  else:

    return "F"

# Get marks from the user

try:

  marks = float(input("Enter the marks: "))

  if 0 <= marks <= 100:

    grade = get\_grade(marks)

    print("The grade is:", grade)

  else:

    print("Invalid input. Marks should be between 0 and 100.")

except ValueError:

  print("Invalid input. Please enter a numerical value for marks.")

Output:

Enter the marks: 91

The grade is: A+

* “match” operator: It refers to the structural pattern matching feature. It is similar to a “switch” statement in other languages.

Example:

weather : str = input("Please enter weather condition : ")

match weather:

    case "sunny":

      print("It's a beautiful sunny day!")

    case "rainy":

      print("Grab your umbrella, it's raining.")

    case "cloudy":

      print("It's a bit cloudy today.")

    case "snowy":

      print("Let's build a snowman!")

    case \_: # Default case (if none of the above match)

      print("I'm not sure about the weather.")

Output:

Please enter weather condition : sunny

It's a beautiful sunny day!

OR

Please enter weather condition : abc

I'm not sure about the weather.

Here,

“match” statement is used to start the pattern matching.

“case” clauses defines different patterns to match.

\_ (underscore) case acts like a "default" or wildcard pattern.

* Loop:

1. FOR
2. WHILE

Both have different syntax and usage. When matching conditions are limited then use “while” loop else use “for” loop.

Example of “for” loop to read elements of a “list”:

numbers : list = [1,2,3,4,5]

for number in numbers:

  print(number)

Output:

1

2

3

4

5

Here, best practice is to use variable name “number” in “for” loop instead of variable name “i”

Example of “for” loop to read elements of a “str” (string):

letters : str = "abcdefghijklmnopqrstyvwxyz"

for letter in letters:

    print(letter)

Output:

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

y

v

w

x

y

z