**LECTURE-05 (Saturday 10-May-2025)**

* <https://github.com/panaversity/learn-modern-ai-python/blob/main/00_python_colab/05_control_flow/Agentic_AI_Python_Lesson_05_Control_Flow_%26_Loops.ipynb>
* Loop: It is used to repeat a block of code multiple times. Python supports two types of loops:

1. FOR: It is used to iterate over a sequence (e.g., lists, strings, or ranges).
2. WHILE: It is used to repeat a block of code as long as a condition is True.

* In Python, there is no “foreach” and “do-while” loop
* “foreach” loop in other programming languages, is used to read fixed length data.
* “do-while” loop in other programming languages, is used when at least code should be executed once whether condition is true or false.
* When to use which loop:

1. FOR Loop: You know in advance how many times you want to loop.

Example (Grading a class of students): you have a list of 30 students and you want to calculate the average score for each student. You use for loop to iterate over the list of students and calculate the average score for each one.

1. WHILE Loop: You don’t know in advance how many times to loop. You loop based on a condition that is checked every iteration.

Example (Filling up a gas tank): You want to fill up your gas tank until it's full. You don't know how many times you'll need to fill up the tank, but you'll keep filling it up until it's full. You use a while loop to fill up the tank until it's full.

1. If we forget to add increment or decrement in while loop then infinite loop

Example 2:

counter : int = 2025

while counter > 1998:

  print("counter value : ", counter)

  counter == 1998

Output:

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

counter value : 2025

Here, same line will be printed infinite times because in while loop, “counter” value is not changed and in last line of while loop, comparison is done not resetting “counter” value.

We can fix issue by code:

counter : int = 2025

while counter > 1998:

  print("counter value : ", counter)

  counter = 1998

Output:

counter value : 2025

Here, we set “counter” value to “1998”

* Examples of FOR Loop

Example 1 (Iterate over a list):

fruits: list = ["apple", "banana", "cherry"]

for fruit in fruits:

    print(fruit)

Output:

apple

banana

cherry

Example 2 (Iterate over a string):

word: str = "Python"

for letter in word:

  print(letter)

Output:

P

y

t

h

o

n

* “FOR” with “ELSE”: In Python, a for loop can have an else block. The else block runs only if the loop completes without a break statement.

Example 1 (without break statement):

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

for num in numbers:

  print(num)

else:

  print("Loop completed successfully!")

Output:

1

2

3

4

5

Loop completed successfully!

Example 2 (with break statement):

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

for num in numbers:

  if num == 3:

    break

  print(num)

else:

  print("Loop completed successfully!")

Output:

1

2

Example 3:

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

for num in numbers:

  if num == 6:

    print("Number found!")

    break

else:

  print("Number not found!")

Output:

Number not found!

* Generator function: It is a function which will not run itself, we need to execute it. (e.g.: range function). Generator function is memory-efficient because we check first value if correct then load value in memory.
* “range” function: This function take 3 parameters

1. Start: It is required parameter
2. End: range function executes till “-1” of value provided in this parameter
3. Step: move forwarded to value provided in this parameter, from “start” parameter value.

Example 1:

for number in range(1,10):

  print(number)

Output:

1

2

3

4

5

6

7

8

9

Example 2: print table of 2

for n in range(1,11):

  print(f"2 x {n} = {2 \* n}")

Output:

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

2 x 6 = 12

2 x 7 = 14

2 x 8 = 16

2 x 9 = 18

2 x 10 = 20

Example 2: print table of provided number

number : int = int(input("Enter your number : "))

for n in range(1,11):

  print(f"{number} x {n} = {number \* n}")

Output:

Enter your number : 7

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

7 x 4 = 28

7 x 5 = 35

7 x 6 = 42

7 x 7 = 49

7 x 8 = 56

7 x 9 = 63

7 x 10 = 70

OR

Enter your number : 35

35 x 1 = 35

35 x 2 = 70

35 x 3 = 105

35 x 4 = 140

35 x 5 = 175

35 x 6 = 210

35 x 7 = 245

35 x 8 = 280

35 x 9 = 315

35 x 10 = 350

Example 3: print even numbers

for i in range(2, 11, 2):

  print(i)

Output:

2

4

6

8

10

* \_ (underscore): It is a throwaway variable, which is a common Python convention for a variable that you don't plan to use.

Example:

name, \_, age = ("Alice", "Engineer", 30)

print(name)

print(age)

Output:

Alice

30

Here, we will not use “Engineer” value, so it's assigned to “\_”(underscore) variable.

* Example of WHILE Loop:

Example 1:

count: int = 1

while count <= 5:

  print(count)

  count += 1

Output:

1

2

3

4

5

Example 2:

names : list[str] = ["abc", "def","ghi","pqr"]

counter : int = 0

while counter < len(names):

    print("Username :", names[counter])

    counter += 1

Output:

Username : abc

Username : def

Username : ghi

Username : pqr

OR above code can be written with FOR loop as:

names : list[str] = ["abc", "def","ghi","pqr"]

for name in names:

  print("Username :", name)

Output:

Username : abc

Username : def

Username : ghi

Username : pqr

* Controlling Loops: In Python, we can use two keywords “break” and “continue” to control loops:

1. break: It exits the loop immediately.

Example:

for i in range(10):

  if i == 5:

    break

  print(i)

Output:

0

1

2

3

4

Here, “for” loop will be ended when value of variable “i” reached to 5.

1. continue: It skips the rest of the code in the current iteration and moves to the next iteration.

Example:

for i in range(5):

  if i == 3:

    continue

  print(i)

Output:

0

1

2

4

Here, “3” is not printed because when value of “i” is “3” then “continue” statement executed and code execute return back to for loop and below print statement ignored.

* Nested loop: Loop inside Loop

Example 1:

names : list[str] = ["abc", "def","ghi","pqr"]

for name in names:

  print("Username : ", name)

for i in range(len(name)):

  print("Username : ", name[i])

Output:

Username : abc

Username : def

Username : ghi

Username : pqr

Username : p

Username : q

Username : r

Here, even first loop ended but still “name” variable exist for second loop and “name” variable contains last value “pqr” of first loop.

Example 2:

names : list[str] = ["abc", "def","ghi","pqr"]

for name in names:

  print("Username :", name)

  for letter in name:

    print("Letters :", letter)

Output:

Username : abc

Letters : a

Letters : b

Letters : c

Username : def

Letters : d

Letters : e

Letters : f

Username : ghi

Letters : g

Letters : h

Letters : i

Username : pqr

Letters : p

Letters : q

Letters : r

Example 3:

posts : list[tuple[str, list]] = [

    ("post 1", ["comment 11", "comment 12"]),

    ("post 2", ["comment 21", "comment 22"]),

]

for post in posts:

  card = f"""LinkedIn Post

  Post Title : {post[0]}

  All Comments : {post[1]}

  """

  print(card)

Output

LinkedIn Post

Post Title : post 1

All Comments : ['comment 11', 'comment 12']

LinkedIn Post

Post Title : post 2

All Comments : ['comment 21', 'comment 22']

OR

posts : list[tuple[str, list]] = [

    ("post 1", ["comment 11", "comment 12"]),

    ("post 2", ["comment 21", "comment 22"]),

]

for post in posts:

  card = f"""LinkedIn Post

  Post Title : {post[0]}

  All Comments : {"--".join(post[1])}

  """

  print(card)

Output:

LinkedIn Post

Post Title : post 1

All Comments : comment 11--comment 12

LinkedIn Post

Post Title : post 2

All Comments : comment 21--comment 22

* “:=” assignment operator: It creates and initialize variable at the time of assignment.

Example:

x = 3

print(x)

print(y := 9)

print(y)

if z := 10:

  print(z)

print(x :=20)

Output:

3

9

9

10

20

* <https://github.com/panaversity/learn-modern-ai-python/blob/main/00_python_colab/08_modules_functions/Agentic_AI_Python_Lesson_08_Module_%26_Functions.ipynb>
* Function: In programming languages, function or method is used for code reusability and to organize code which increases code readability.
* There are three types of functions:

1. Default or pre-defined functions (e.g.: Len, id, print, range)
2. User defined or custom functions

Example 1:

def print\_something():

  print("something")

print\_something()

Output:

something

Example 2 (function with parameter):

def say\_hello(user\_name : str):

  print("Hello ", user\_name)

user\_name : str = input("Enter you name : ")

say\_hello(user\_name)

output :

Enter you name : abc

Hello abc

Here, “def say\_hello(user\_name : str)” shows that “say\_hello” is a function which have one parameter “user\_name” whose data type is string.

While “say\_hello(user\_name)” shows that function “say\_hello” is called and “user\_name” is passed as function argument.

1. Return or non-return functions
2. Non-return functions: They do not return anything.

Example:

def say\_hello(user\_name : str) -> None:

  print("Hello ", user\_name)

user\_name : str = input("Enter you name : ")

say\_hello(user\_name)

Output:

Enter you name : abc

Hello abc

1. Return functions: They return any value.

Example:

def sum\_of(num1 : int, num2 : int):

  result = num1 + num2

  return result

sum\_of(10,20)

Output:

30

* Example of function:

def print\_tables\_to\_provided\_number(last\_table\_number : int, multiply\_table\_number : int) -> None:

  for i in range(1, last\_table\_number + 1):

    print(f"\n--- Table of {i} ----")

    for j in range(1,multiply\_table\_number + 1):

      print(f"{i} x {j} = {i\*j}")

last\_table\_number = int(input("Please tell us, till which number you want to print table start from 1 :"))

multiply\_table\_number = int(input("Please tell us, how many numbers you want to be mulitplied start from 1 :"))

print\_tables\_to\_provided\_number(last\_table\_number, multiply\_table\_number)

Output:

Please tell us, till which number you want to print table start from 1 :3

Please tell us, how many numbers you want to be mulitplied start from 1 :4

--- Table of 1 ----

1 x 1 = 1

1 x 2 = 2

1 x 3 = 3

1 x 4 = 4

--- Table of 2 ----

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

--- Table of 3 ----

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12