LAB-4

➢ pip Installation

- For windows users:

```
py -m pip install pandas
py -m pip install emoji
```

- For Mac users:

```
python3 -m pip install pandas
python3 -m pip install emoji
```

- After Installation, Run the following:

```
from emoji import emojize
print(emojize(":thumbs_up:"))
```

- with this you all can see the name of all the packages you have installed till now.

```
pip list
```

➢ for loops in Python

For this week's lab we would like to start with briefly revising the concept of for loops in Python. This will serve as a nice refresher for those who understand the concept well and will also give the chance of clarifying any doubts to those who may have felt a bit challenged with for loops in the assignments.

In a nutshell, we will look at a few examples of when the different ways of writing the for loop are useful.

***Use case 1*:** When you simply need to visit each element in a collection of things for ex. in a list - Use the simple for loops with the in keyword. You don't need to use range function.

```
example_list_1 = [3, 4, "nice work", 90.9876, 0, '*', '_']

# Printing every element of the example_list_1
for item in example_list_1:
  print(item)
```

Use case 2: more often, we want to do some processing on the elements in our list rather than just printing them. For example, you want to answer questions like - print the element if it's a string or an integer. In this case, you use the IF/ELSE structure inside the loop to test the type of the element.

Notice that the below loop will visit every element but will only print the strings.

```
for item in example_list_1:
  if type(item) == str:
    print(item)
```

Use case 3: What if you get a list of lists! Let's see what happens in this case. Tell the students how the item now prints the sublists.

example_list_2 = [[0,1,2], ["A", "B"], ['$', '#', '*'], "Hello", [], []]

The following loop will simply visit each element inside the list - no different than previous example only that our list now contains sublists!

```
for item in example_list_2:
 print(item)
```

Use case 4: What if you now want to do something with the sublists? Let's say we want to print the second element from our sublists. We will need to two things now,

1. We will need to check if the item that we get is indeed a sublist (and not an integer or a string) because only a sublist (which is also a list) will have the 2nd element that we want to print. If it's an integer or a string then basically you don't have any second element because the integer is the only thing that you have. Also, we will need to check that the sublist is not empty because we can't access anything inside an empty sublist.
2. If item is indeed a list, then we will need to index the second element from our sublist and then print it.

- This requires us to use if conditional and list indexing inside the for loop.

```
for item in example_list_2:
 if len(item)>1 and type(item) == list: # <-- non-empty list (not an integer/string/float) and that the list
```

has atleast 2 elements.
  print(item[1]) # <-- Index the 2nd element and print it.

Use case 5: nested loops: having a loop inside a loop. In nested loop we have "inner" loop and "outer" loop. The "inner" loop will be executed one time for each iteration of the "outer" loop.

"for loop" using "range"

```
animal_list = ["cat", "dog", "hamster", "goat", "turtle"]
for i in range(len(animal_list)):
    print("The animal at index " + str(i), " is " + animal_list[i])
```

- The followings are examples for nested loop:

```
for i in range(3)
    for j in range(4):
        print("outer loop: " + i + "and inner loop: " + j)

lst = ['a', 'b', 'c']
for i in range(5):
    for j in range(3)
        print(i , lst(j))
```

- The output will be:

```
0 a
0 b
0 c
1 a
1 b
1 c
2 a
2 b
2 c
3 a
3 b
3 c
4 a
4 b
4 c
```

➢ While loops in Python

In While loop we have a condition and the loop will execute as long as the condition is true.

```
sum = 0
i = 0
while i<5:
    sum += i
    i += 1

print("we are out of the while!")
```

- we can reimplement the above example using not equal sign

```
sum = 0
i = 0
while i!=5:
    sum += i
    i += 1

print("we are out of the while!")
```

- the conditions can use string too.

```
password = ''
i = 0
lst = ['p', 'a', 's', 's', 'w', 'o', 'r', 'd', 'a', 'b', 'c']
while password != 'password':
    password += lst[i]
    i += 1

print("we are out of the while!")
```

➢ Python Keywords (break, continue)

There are a set of keywords That are reserved in python and we cannot use them as a variable name, function names, or any other identifiers.

- In Python, break and continue statements can alter the flow of a normal loop.
- Loops iterate over a block of code until the test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
- The break and continue statements are used in these cases.

Python break statement

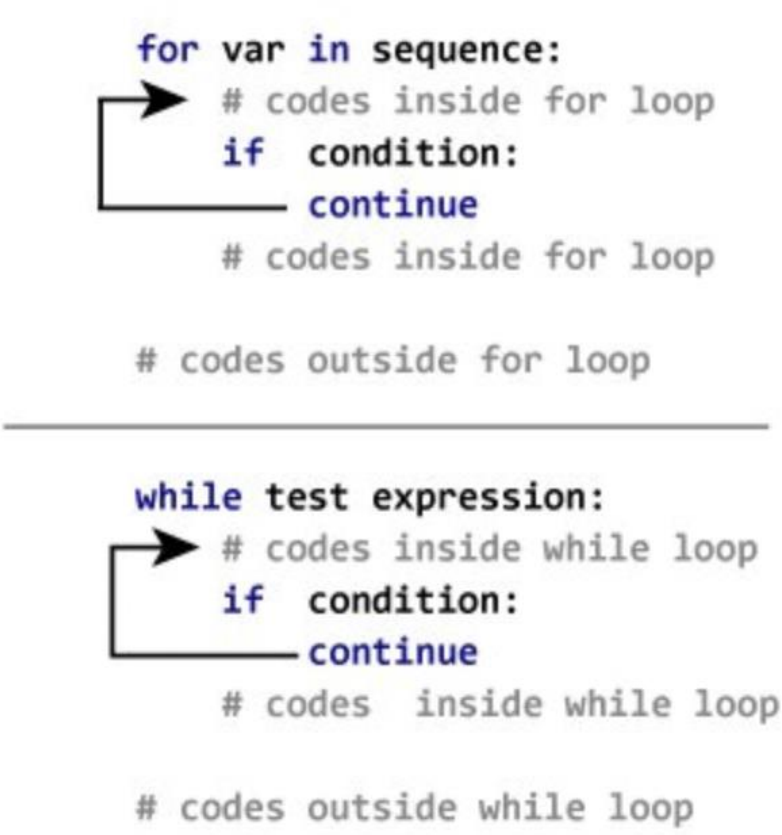
- The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
- If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.

- Example

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

➢ Python continue statement

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.
- The working of the continue statement in for and while loop is shown below.



- Example:

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

➢ Dictionary in Python

- Dictionary is a container in python just like list, string and etc. Python dictionary is a collection of items. Each item of a dictionary has a key: value pair. Dictionaries defined using the {} (curly brackets) and has 'keys' and 'values' (key: value) .
- Keys can be of any data type but they should be unique.
- We can change, add or remove items after the dictionary has been created. In dictionary we are allowed to have duplicate values but we are not allowed to have the duplicate keys.
- We are going to show how we can create a dictionary, how we can add elements, update a value and, how we can get access to each element in it

Example of dictionary.

```
classes = {
  'reptiles': [ 'turtles', 'snakes', 'lizards'],
  'mamals': ['bats' 'bear', 'squirrels', 'deer'],
  'fish': ['gold fish', 'neon tetra'],
  'birds': ['owls', 'swift']
}

student = {
  'student_name': 'Leo',
  'course': 'c200',
  'age': 20
  'grade': 'A'
}
```

.

There is a dictionary that has the items id with their price.

```
dict = {
    'item1': 55,
    'item2': 24,
    'item3': 100,
    'item4': 55
}
```

add elements to dictionary

```
dict['item6'] = 33
dict['item9'] = 101
print(dict)
```

Updating existing Key's Value

```
dict[item1] = 50
```

accessing an element using key

```
print("Accessing a element using key:")
print(dict['item3'])
```

Delete the dictionary

del dict['item1']

print(dict)

how to get all keys

```
dict.keys()
```

how to get all values

```
dict.values()
```

dictionary with for loop

```
for k in dict:
  print(k) # print keys in dict
```

```
for k in dict:
  print(dict[k]) # print values in dict
```