Agenda

- Conditional Logic
- Loop(for,while)
- Control Statement
- List Comprehension

Conditional Logic

What are Conditional Statements?

> Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

> Python if Statement is used for decision-making operations

IF

In [ ]:
```
#python uses indentation to identify a block
# ''' Syntex:
# if condition:
#   statement1
# statement2  '''

i = 10
if (i > 15):
   print ("10 is less than 15")

print("Outside IF Block. ")

#OR Short Hand If
# if i > 5: print(i,"is greater than 5")
```

Elif

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

In [ ]:
```
a = 15
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

if- else

In [ ]:
```
'''Syntax:

if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false '''

i = 20;
if (i < 15):
    print ("i is smaller than 15")
    print ("i'm in if Block")
else:
    print ("i is greater than 15")
    print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

Nested If

In [ ]:
```
x = 30

if x > 10:
  print("Above ten,")
  if x > 20:
    print("Also above 20!")
else:
    print("Not above 20.")
```

Loop

For in loops

> For loops are used for sequential traversal. For example: traversing a list or string or array etc. for loops only implements the collection-based iteration.

In [ ]:
```
'''Syntax:

for var in iterable:
    # statements '''

name = ["mini", "sini", "rini"]

for x in name:
  print(x)
```

In [ ]:
```
#Looping Through a String
for x in "Computerscience":
  print(x)
```

range() Function

> The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number

In [ ]:
```
for x in range(2, 6):
  print(x)
```

In [ ]:
```
sum = 0
for i in range(1, 11):
    sum = sum + i
print("Sum of first 10 natural number :", sum)
```

while Loop

> while loop we can execute a set of statements as long as a condition is true.

In [ ]:
```
'''Syntax:

while expression:
    statement(s) '''

count = 0
while (count < 3):
    count = count + 1
    print("Hello World")
```

In [ ]:
```
# Single statement while block
count = 0
while (count < 5): count += 1; print("Hello all")
```

Control Statement

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed

- Continue Statement- It returns the control to the beginning of the loop.
- Break Statement- It brings control out of the loop.
- Pass Statement- We use pass statement to write empty loops

In [1]:
```
#Continue Statement
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for x in numbers:
    if x == 5:
        continue
    print(x)

print("Out of Loop")
```

```
1
2
3
4
6
7
8
9
10
Out of Loop
```

In [ ]:
```
#Continue Statement
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for x in numbers:
    if x == 5:
        break
    print(x)

print("Out of Loop")
```

In [ ]:
```
#Pass Statement: We use pass statement to write empty loops.
#Pass is also used for empty control statements, functions and classes.
# An empty loop

i = 10
j = 15

if i > j:
    pass

def calculate():
    pass

class Person:
    pass
```

List Comprehension

> Create powerful functionality within a single line of code

Syntax of List Comprehension

[expression for item in list]

In [ ]:
```
num=[1,2,3,4,5,6]
a=[x*x for x in num]
print(a)
```

In [ ]:
```
sentence = 'the rocket came back from mars'
vowels = [i for i in sentence if i in 'aeiou']
vowels
```

In [ ]:
```
list1 = [3,4,5]

multiplied = [item*3 for item in list1]

print(multiplied)
```

In [ ]:
```
#Conditionals in List Comprehension

number_list = [ x for x in range(20) if x % 2 == 0]
print(number_list)
```

In [ ]:
```
#range function use in list comprehension
S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
print(S)
print(V)
```

In [ ]:
```
#List Comprehension with nested loop
[x + y for x in [1, 2, 3] for y in [3, 4, 5]]
```

In [ ]:
```
#Iterate two or more list simultaneously within list comprehension
list_1 = [1, 2, 3 , 4]
list_2 = ['a', 'b', 'c', 'd']

# Two lists
[(i, j) for i, j in zip(list_1, list_2)]
```

List Comprehension vs For Loop in Python

> Suppose, we want to separate the letters of the word human and add the letters as items of a list. The first thing that comes in mind would be using for loop.

In [ ]:
```
# Iterating through a string Using for Loop


h_letters = []

for letter in 'human':
    h_letters.append(letter)

print(h_letters)
```

In [ ]:
```
#Iterating through a string Using List Comprehension

h_letters = [ letter for letter in 'human' ]
print( h_letters)
```

Thank you