Flow control statements

Agenda

1. if else
2. elif
3. for
4. while
5. break
6. continue
7. pass

if else

if else

- **if** statements are used for decision making i.e whether a block of code needs to be executed or not.
- **if** block can be optionally followed by an **else** block.

All statements which belong to the ***if*** block are having same indentation.

```
Syntax:

if(condition):
    statement-1
    statement-2
    .          .
    .          .
    statement-n
```

if else continued..

All statements which belong to the ***if*** block are having same indentation.

All statements which belong to the ***else*** block are having same indentation.

```
Syntax:

if(condition):
    statement-1
    statement-2
    .          .
    statement-n

else:
    statement-1
    statement-2
    .          .
    statement-n
```

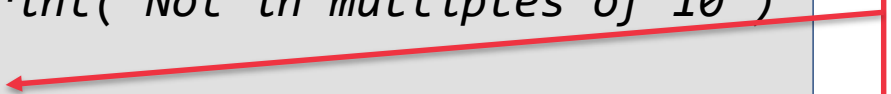
if else continued..

```
Predict the output:

a = 10
if(a%2 == 0):
    print("Even")
else:
    print("Odd")
```

```
You are right..!!

Output:

Even
```

if else continued..

```
Predict the output:

a = 500
if(a%10 == 0):
    print("In multiples of 10")
else:
    print("Not in multiples of 10")
else:
    print("End")
```



You are right..!!

SyntaxError because of the second else block.

Multiple if statements

```
Predict the output:

a = 15
if(a%2 == 0):
    print("Even")
if(a%2 != 0):
    print("Odd")
if(a >= 0):
    print("Positive")
```

```
Output:

Odd
Positive
```

Nested if statements

```
Predict the output:

gender = ‘female’
age = 25
if(gender == ‘female’):
    if(age > 18):
        print("Eligible")
    else:
        print("Not Eligible")
else:
 print("End")
```

```
Output:

Eligible
```

Nested if statements continued..

```
Predict the output:

gender = ‘male’
age = 25
if(gender == ‘female’):
    if(age > 18):
        print("Eligible")
    else:
        print("Not Eligible")
else:
 print("End")
```

```
Output:

End
```

elif

wipro

elif

- You have multiple **if** conditions and when any one if condition is satisfied, you may want other conditions not to be checked and to be simply skipped.
- **elif** is the solution. It is similar to **else if** in other languages like C,C++,Java.

```
Syntax:

if(condition-1):

elif(condition-2):
.                .
.                .
elif(condition-n):

else:
```

When none of the **if** conditions are matching, **else** will be executed.

elif continued..

```
Predict the output:

designation = ‘Engineer’
if(designation == 'Doctor'):
     print('Hi Doctor..!!')
elif(designation == 'Engineer'):
     print('Hi Engineer..!!')
elif(designation == 'Lawyer'):
     print('Hi Lawyer..!!')
else:
     print('Invalid designation')
```

```
Output:

Hi Engineer..!!
```

When this **else** will be executed ? Think..!!

for

for

- Two important uses of **for** loop:
  1. It is used when a block of code needs to be executed more than once.
  2. It is used to iterate over a collection of elements. Elements of List, Tuple, String, Dictionary are iterated using **for** loop.
- All the statements that belong to the **for** loop should have same indentation.
- Syntax of **for** loop is different in Python.
- Before proceeding to that, lets quickly learn what is range( ) function?

What is range( ) function?

- **range( )** is the built-in function which returns a range object, which is a sequence of integers.

```
Syntax:

range(start, stop, step)
```

- start : It specifies the starting value for the sequence. Its an optional argument.
  By default it is 0.
- stop : It specifies the ending value for the sequence and this value is excluded.
  It is a mandatory argument.
- step : It specifies the increment or decrement value for the next number in the sequence.
  Its an optional argument. By default it is 1.

Examples for range( ) function

Code	Values generated
range (1, 5, 1)	1, 2, 3, 4
range (10)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
range (1, 3)	1, 2
range (1,10,2)	1, 3, 5, 7, 9
range (10, 0, -1)	10, 9, 8, 7, 6, 5, 4, 3, 2, 1
range (20, 0, -5)	20, 15, 10, 5

Using range( ) function with for loop

```
Program:

for count in range(0, 5, 1):
    print('Hello..count is ',count)
```

```
Output:

Hello..count is  0
Hello..count is  1
Hello..count is  2
Hello..count is  3
Hello..count is  4
```

Calculating sum of numbers from 1 to 10

```
Program:

sum = 0;
for num in range(1, 11):
    sum = sum + num
print(sum)
```

```
Output:

55
```

Using for loop with string to print each character

```
Program:

name = ‘Tushar’
for letter in name:
    print(letter)
```

```
Output:

T
u
s
h
a
r
```

Using for loop with list to print only the even numbers

```
Program:

li = [2, 67, 44, 89]

for num in li:
    if(num%2 == 0):
        print(num)
```

```
Output:

2

44
```

Using for loop with tuple to calculate the sum of elements

```
Program:

t1 = (1, 3, 5, 7)
sum = 0

for num in t1:
    sum = sum + num
print(sum)
```

```
Output:

16
```

Using for loop with dictionary to print the values

```
Program:

d1 = {
    'name'  : 'Chandu',
    'age'   : 24,
    'gender'   : 'Male',
    'country'  : 'India'
 }

for key in d1:
    print(d1[key])
```

```
Output:

Chandu
24
Male
India
```

Using else with for loop

- **for** loop can be optionally followed by an **else** block.
- Statements in **else** block will be executed only when **for** loop is not executed even once or when the loop is completed without breaking in between.

```
Program:

s = '' #empty string

for letter in s: #not executed even once
    print(letter)
else:
    print("Empty string")
```

```
Output:

Empty string
```

Using else with for loop continued..

```
Program:

li=[1,2,3]
for x in li:
    print(x,end=" ")
else:
    print('\nLoop is completed without breaking')
```

```
Output:

1 2 3
Loop is completed without breaking
```

Nested for loops

```
Program:

names = [‘MARCEL’, ‘CHAD’]
for name in names:
    for letter in name:
        print(letter)
    print('***')
```

Statements that belong to the outer **for** loop.

Statements that belong to the inner **for** loop.

```
Output:

M
A
R
C
E
L
***
C
H
A
D
***
```

while

while

- **while** loop is used to repeatedly execute certain block of statements as long as the given condition is True.
- **while** loop can be optionally followed by an **else** block which will be executed only when while loop is never executed i.e condition is always false.

All statements which belong to the ***while*** loop are having same indentation.

```
Syntax:

while(condition):
    statement-1
    statement-2
    .         .
    .         .
    statement-n
```

Calculating sum of digits using while loop

```
Program:

x = 12345
sum = 0
while(x!=0):
    last_digit = x%10
    sum = sum + last_digit
    x = x//10
print(sum)
```

```
Output:

15
```

while with optional else block

All statements which belong to the ***while*** loop are having same indentation.

All statements which belong to the ***else*** *block* are having same indentation.

```
Syntax:

while(condition):
    statement-1
    statement-2
    .         .
    .         .
    statement-n

else:
    statement-1
    .         .
    statement-n
```

while with optional else block continued..

```
Program:

num = 0
while(num > 0): #condition is always false
    print('Hello..')
else:
    print('End')
```

```
Output:

End
```

while with optional else block continued..

```
Program:

i=10
while(i<20):
    print(i,end=" ")
    i=i+5
else:
    print('\nLoop completed without breaking')
```

```
Output:

10 15
Loop completed without breaking
```

Nested while loops

```
Program:

row = 1

while(row<=5):
    col = 1
    while(col<=row):
        print('*', end =" ")
        col = col+1
    print()
    row = row+1
```

```
Output:

*
* *
* * *
* * * *
* * * * *
```

break

break

- **break** keyword is used only within loops.
- It helps in terminating the loop in between based on some condition.
- When **break** statement is encountered, control goes out of the loop.
- Statements written after the **break** statement are never executed.

```
Example:

while(condition):
    if(condition):
        break
        statement-1 of if  #never executed
    statement-1 of while
```

break continued..

```
Program:

li = [1, 3, 5, 4, 7, 9]
for num in li:
    if(num%2 == 0):
        break
        print('After break')
    else:
        print(num)
print('Outside for loop')
```

```
Output:

1
3
5
Outside for loop
```

continue

continue

- **continue** keyword is used only within loops.
- It helps in skipping certain statements from being executed based on some condition.
- When **continue** statement is encountered, all the following statements are skipped and loop goes to the next iteration.

```
Example:

while(condition):
     if(condition):
        statement-1 of if
        continue
     else:
        statement-1 of else
     statement-1 of while
```

All these statements are skipped when if condition is True and **continue** keyword is encountered.

continue keyword continued..

```
Program:

alphabets = ['A', 'B', 'C', 'D', 'E']

for letter in alphabets:
    if(letter=='C'):
        continue
    else:
        print(letter)
```

```
Output:

A
B
D
E
```

continue keyword continued..

```
Program:

a = 5
while(a>0):
    if(a==3):
        a = a-1
        print('skipped')
        continue
    else:
        print(a)
    print('***')
    a = a-1
```

```
Output:

5
***
4
***
skipped
2
***
1
***
```

pass

pass

- **pass** statement is doing no operation.
- It is executed like a valid statement but it does nothing.
- We can use **pass** statement inside a function to denote it does nothing for time being.
- We will see more about pass statement when we learn functions.

```
Program:

t1 = (1, 2, 3, 4, 5)
for x in t1:
    if(x%2 == 0):
        pass #does nothing
    else:
        print(x)
```

```
Output:

1
3
5
```

Thank you