ONE WEEK STTP ON "PYTHON PROGRAMMING"

DAY 2 – 23 JUNE 2020

By

R.CHITHRA DEVI

ASSOCIATE PROFESSOR

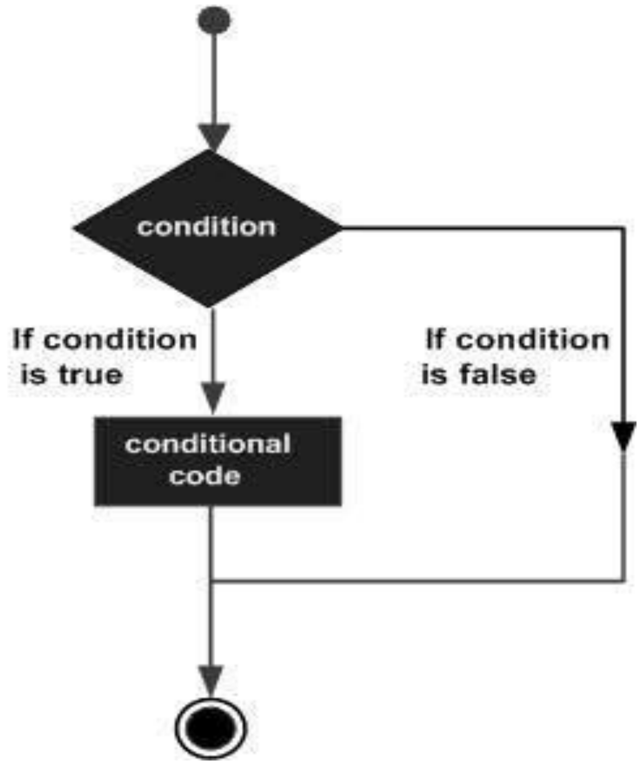
DEPARTMENT OF INFORMATION TECHNOLOGY

DR.SIVANTHI ADITANAR COLLEGE OF ENGINEERING

TIRUCHENDUR - 628205

PYTHON - DECISION MAKING

- Decision Making statements are used to control the flow of execution of a program depending upon condition.
- Python programming language assumes any **non-zero** and **non-null** values as TRUE, and any **zero** or **null values** as FALSE value.



PYTHON - DECISION MAKING

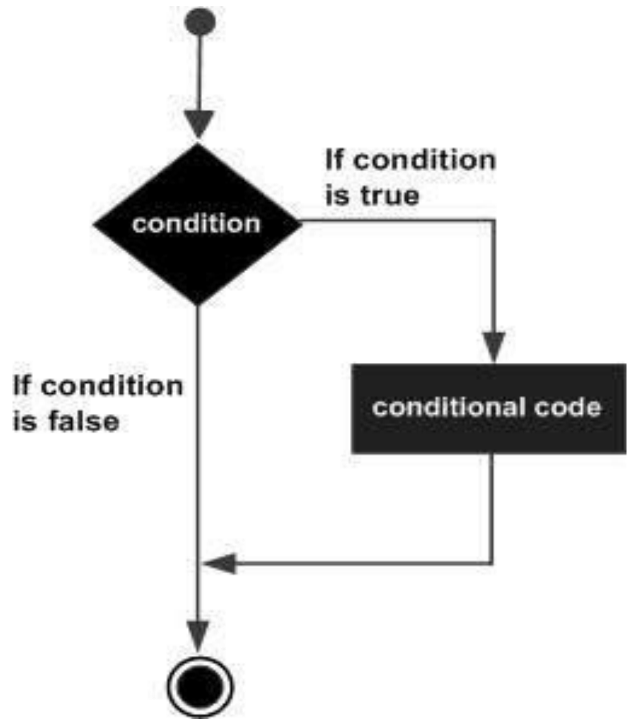
- Python programming language provides the following types of decision-making statements.
    - if statements
    - if…else statements
    - If …elif statements
    - nested if statements

IF STATEMENTS

- The **if** statement contains a logical expression using which the data is compared and a decision is made based on the result of the comparison.
- Syntax:

```
        if expression:
            statement(s)
```

- If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.



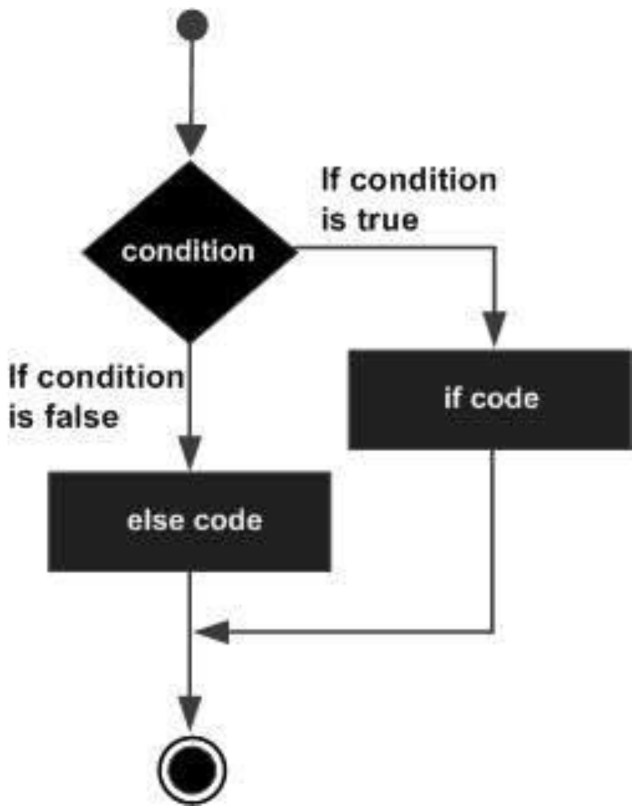
IF STATEMENTS - EXAMPLE

```
>>> a=10
>>> b=10
>>> if a==b:
...     print("Equal")
...     print("OK")
...
Equal
OK
>>>
```

IF…ELSE STATEMENTS

- An **else** statement can be combined with an **if** statement. An **else** statement contains a block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
- The else statement is an optional statement and there could be at the most only one **else** statement following **if**.
- Syntax:

```
if expression:
    statement(s)
else:
    statement(s)
```



IF …ELSE STATEMENTS - EXAMPLE

```
>>> a=10
>>> b=15
>>> if a==b:
...     print("Equal")
... else:
...     print("Not Equal")
...
Not Equal
>>>
```

```
>>> a=10
>>> b=10
>>> if a==b:
...     print("Equal")
... else:
...     print(Not Equal“)
...
Equal
>>>
```

IF…ELIF STATEMENTS

- The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at the most one statement, there can be an arbitrary number of **elif** statements following an **if**.

- Syntax:

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statememt(s)
```

IF …ELIF STATEMENTS - EXAMPLE

```
>>> a=100
>>> if a==100:
...     print("True")
... elif a>100:
...     print("Biggest")
... elif a<100:
...     print("Smallest")
... else:
...     print(a)
...
True
>>>
```

```
>>> a=100;b=100
>>> if a==b:
...     print(“Equal")
... elif a>b:
...     print(“a is Biggest")
... elif a<b:
...     print(“b is Biggest")
... else:
...     print(“Not Equal”)
...
Equal
>>>
```

NESTED IF STATEMENTS

- When you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct.
- In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

- Syntax:

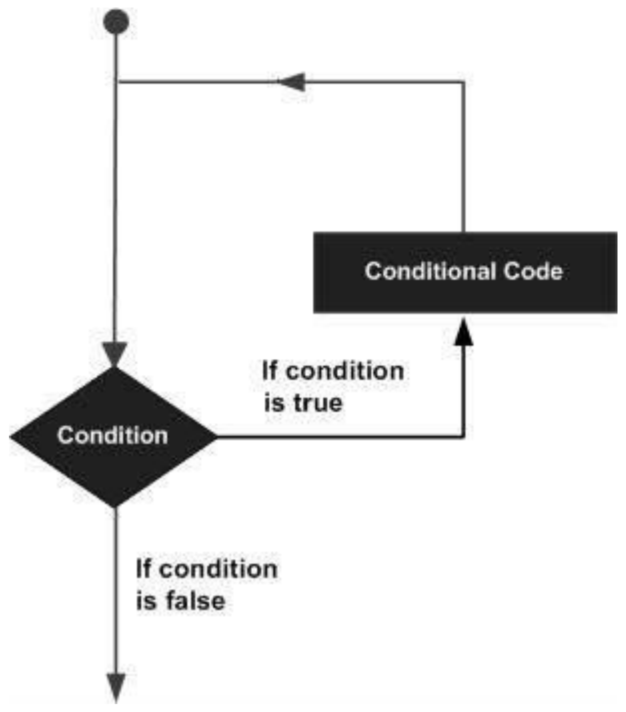
```
if expression1:
    statement(s)
    if expression2:
        statement(s)
    elif expression3:
        statement(s)
    else:
        statememt(s)
else:
    statement(s)
```

NESTED IF STATEMENTS - EXAMPLE

```
>>> a=100;b=100
>>> if a==b:
...     if a>b:
...             print("a is Big")
...     else:
...             print("Not")
... elif a<b:
...     print("a is small")
... else:
...     print("a is Big")
...
Not
>>>
```

PYTHON - LOOPING STATEMENTS

- A loop statement allows us to execute a statement or group of statements multiple times.
- Python programming language provides the following types of loops to handle looping requirements.
    - while loop
    - for loop
    - nested loop

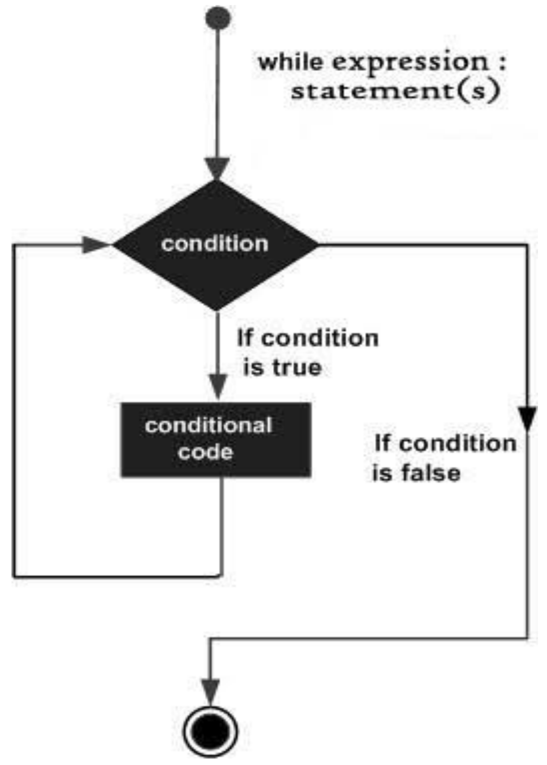


WHILE - LOOP

- A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.
- Syntax:

```
while expression:
    statement(s)
```

- Here, **statement(s)** may be a single statement or a block of statements with uniform indent.



WHILE LOOP - EXAMPLE

```
>>> a=1
>>> while a<10:
...     print("a=",a)
...     a+=1
...
a= 1
a= 2
a= 3
a= 4
a= 5
a= 6
a= 7
a= 8
a= 9
>>>
```

USING ELSE STATEMENT WITH WHILE LOOP

- Python supports having an **else** statement associated with a loop statement.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.
- Example,

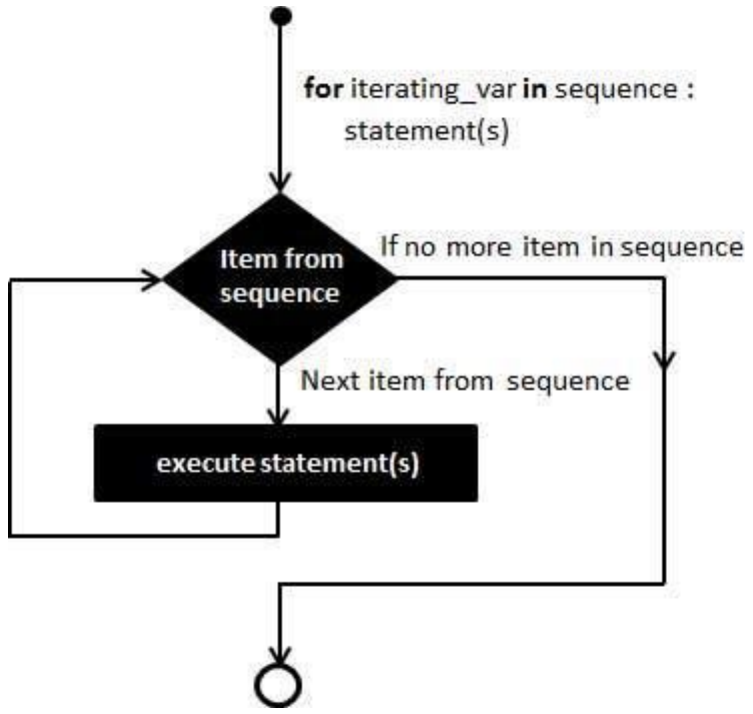
```
>>> count = 0
>>> while count < 5:
...    print (count, " is  less than 5")
...    count = count + 1
... else:
...    print (count, " is not less than 5")
...
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
>>>
```

FOR - LOOP

- The for statement in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- Syntax:

```
for iterating_var in sequence:
        statement(s)
```

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.



RANGE() FUNCTION

- The built-in function range() is the right function to iterate over a sequence of numbers.
- It generates an iterator of arithmetic progressions.
- Example,
  - >>> range(5)
  - range(0, 5)
  - >>> list(range(5))
  - [0, 1, 2, 3, 4]
  - >>> list(range(10))
  - [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

FOR LOOP - EXAMPLE

```
>>> for abc in list(range(4)):
...  print("ABC=",abc)
...
ABC= 0
ABC= 1
ABC= 2
ABC= 3
>>>
```

USING ELSE STATEMENT WITH FOR LOOP

- Python supports having an **else** statement associated with a loop statement.
- If the **else** statement is used with a **for** loop, the **else** block is executed only if for loops terminates normally (and not by encountering break statement).
- Example,

```
>>> numbers = [11,33,55,39,55,75,37,21,23,41,13]
>>> for num in numbers:
...   if num%2 == 0:
...     print ('the list contains an even number')
...     break
... else:
...   print ('the list does not contain even number')
...
the list does not contain even number
>>>
```

NESTED - LOOP

- Python programming language allows the usage of one loop inside another loop. The following section shows a few examples to illustrate the concept.
- Syntax:

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s)
```

- Syntax:

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

NESTED LOOP - EXAMPLE

```
>>> for i in range(1,11):
...     for j in range(1,11):
...        k = i*j
...        print (k, end=' ')
...     print()
...
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
>>>
```

PYTHON – LOOP CONTROL STATEMENTS

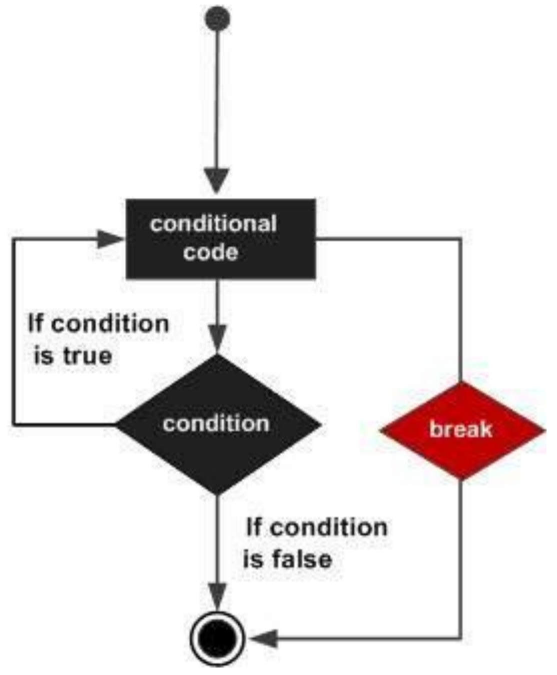
- The Loop control statements change the execution from its normal sequence.
- When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.

- Python supports the following control statements.
    - break statement
    - continue statement
    - pass statement

BREAK

- Terminates the loop statement and transfers execution to the statement immediately following the loop.
- The break statement stops the execution of the innermost loop and starts executing the next line of the code after the block.
- Syntax:

```
break
```



BREAK - EXAMPLE

ONE - FOR

```
>>> for letter in 'Python':
...     if letter == 'h':
...       break
...     print ('Current Letter :', letter)
...
Current Letter : P
Current Letter : y
Current Letter : t
>>>
```

TWO – WHILE

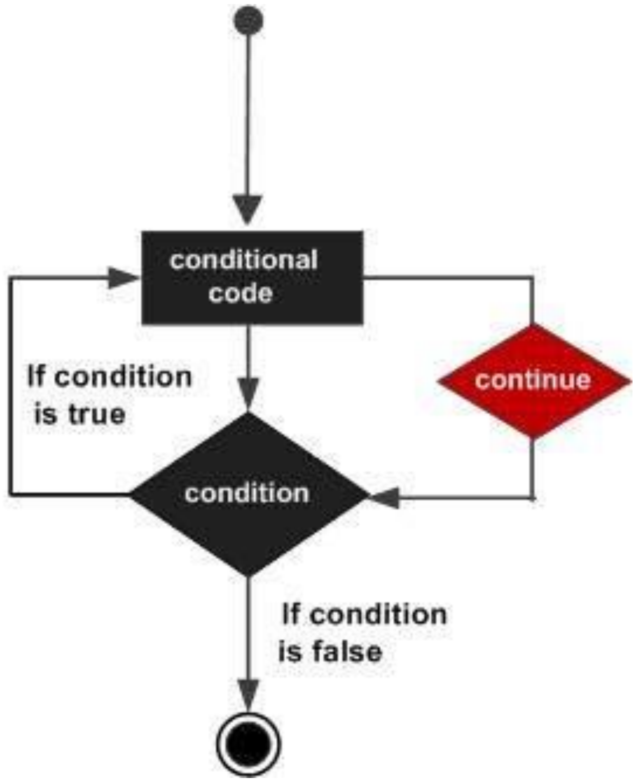
```
>>> var = 10
>>> while var > 0:
...     print ('Current variable value :', var)
...     var = var -1
...     if var == 5:
...       break
...
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
>>>
```

CONTINUE

- The **continue** statement in Python returns the control to the beginning of the current loop.
- When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.
-

  Syntax:

  ```
  continue
  ```



CONTINUE - EXAMPLE

ONE - FOR

```
>>> for letter in 'Python':
...    if letter == 'h':
...       continue
...    print ('Current Letter :', letter)
...
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
>>>
```

TWO – WHILE

```
>>> var = 10
>>> while var > 0:
...    var = var -1
...    if var == 5:
...       continue
...    print ('Current variable value :', var)
...
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
```

PASS

- The **pass** statement is a *null* operation; nothing happens when it executes.
- The **pass** statement is also useful in places where your code will eventually go, but has not been written yet i.e. in stubs.

Syntax:

```
pass
```

Example,

```
>>> for letter in 'Python':
...    if letter == 'h':
...       pass
...       print ('This is pass block')
...    print ('Current Letter :', letter)
...
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
```

THANK YOU FOR ATTENDING THE STTP!

Any Queries?