AUG
Faculty of Computer and Information Sciences
Ain Shams University

Session 2
Getting into python

Comparisons

operator	Example	Result
>	5 > 3 > 2	True
<	'A' < 'B'	True
>=	'C' >= 'Z'	False
<=	15 <= 15	True
==	3 == 4	False
!=	2 != 3	True
is	5 is 5	True
in	"Hi" in ["bye", "Hi"]	False

Logical Operator

Operator	Example	Result
and	5>2 and 8< 1	False
or	5>2 or 8< 1	True
not	5>2 and not 8< 1	True

Conditions

Uses to control the flow in the program. In some cases, different actions will be executed based on some conditions. So we use the if statement.

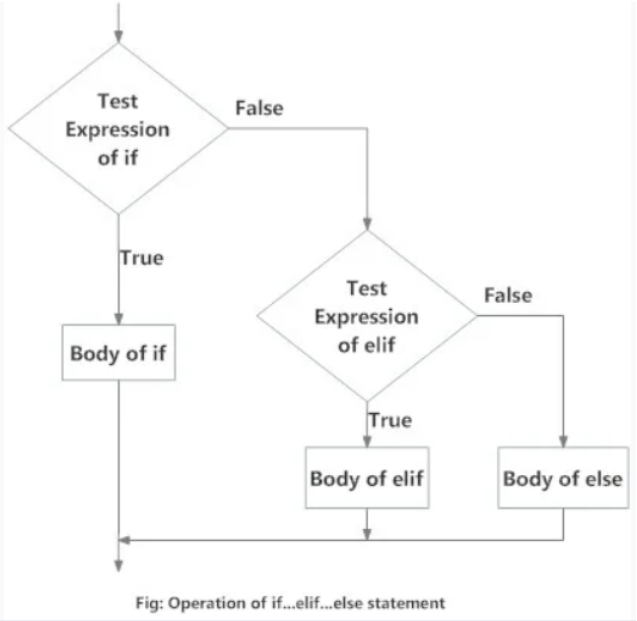


Fig: Operation of if...elif...else statement

If statement

```
result = 50
if num >= 50:
  print(“succeeded”)
```

indentation (whitespace at the beginning of a line) is used to define if statement scope in the code.

else

if the whole previous conditions were False “else” is used to execute some actions

```
job = “Programmer”
if job in ["Programmer", "Engineer"]:
   salary = 8000
else:
   salary = 5000
```

Elif

Uses to determine another path (actions) to be executed based on new conditions if the previous conditions were False.

```
result = 30
if result <= 15:
    print(“cold”)
elif result <= 30:
      print(“sunny”)
 else:
       print(“hot”)
```

Short if

If you have only one statement to execute, you can put it on the same line as the if statement.

```
if 5 > 3 : print("Hi")
```

Short if else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line

```
print("hallo") if 5 > 3 else print("bye")
```

Ternary Operators

if you have multiple else statements have only one statement to execute you can put on the same line

```
num1 = 50
num2 = 60
print("num1 ") if num1 > num2 else print("=") if num1 == num2 else print("num2 ")
```

Nested if

You can have any number of if...elif..else statement in if...elif...else scop.

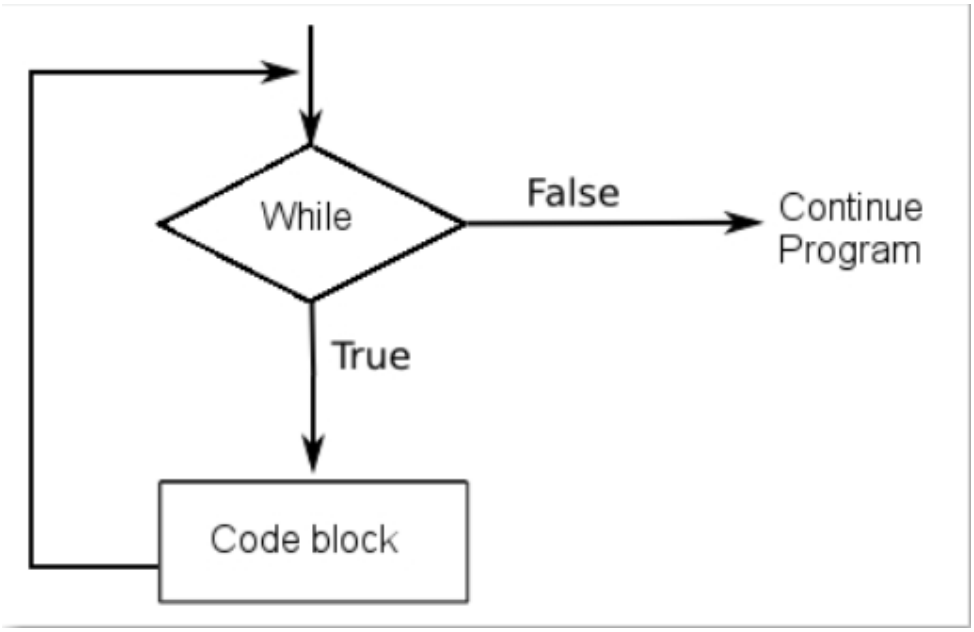
```
result = 80

  if result >= 50:
      if result >= 85:
          print("A")
      elif result >= 75:
            print("B")
      else:
           print("C")
   else:
        print("Faild")
```

Loops

The statement executes only one time in the program so what if we need to repeat the statement more than once. For example, print(1) will print '1', What about printing the numbers from 1 to 10?.

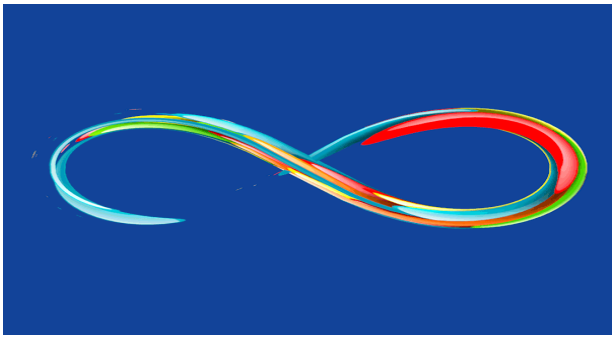
Loops : is used to repeat some statements a certain number of times until the condition is true.



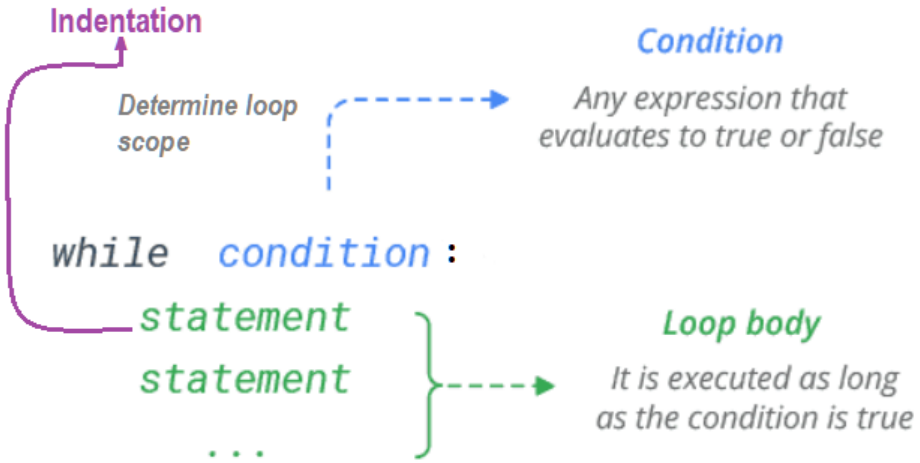
Loop Scope : indentation (white spaces at the beginning of the code) use to indicate loop scope

Infinite loop

What will happen if the condition doesn't change to be false at some time and still true? The loop will repeated forever and won't terminate (infinite loop)



While Loop



While loop executes a set of statements (loop body) as long as a condition is true. The condition should change in the body to be false at some time. For example :

```
num = 0
while num <= 15:
  num += 1
  print(num)
```

This code prints all numbers from 1 to 15 . When num equals 16 the condition will be false and the loop will terminate.

For loop

Used to iterate over sequences (String, List, dictionary ... etc) using in operator. For example:

```
countA = 0
countG = 0

for char in "AAGCTC":
   if char == 'A':
      countA += 1
   elif char == 'G':
      countG += 1

print("Number of A : ", countA, " & G : ", countG)
```

range(end number) function

Return sequence of numbers, Start by default from 0 , increment by 1, to ends at the specific number

It can take up to three parameters range(Start , end , increment).For example:

```
for num in range(0,20,2):
    print (num)
```

This code prints even numbers from 0 to 19 . to decrement you can pass -ve number.

Break

We can terminate the loop before it ends by using a break statement. For example if we look for a specific item in sequence and we want to stop the iterations when we find it.

```
for index in “Hello world”:
     If index == ‘o’:
        print (index)
        break
```

Continue

If we want to skip the rest of this iteration and start the next one we use a continue statement.

```
num = 15
while num >= 0:
  if num % 2 != 0:
    num -= 1  #
    continue
  print(num)
  num -= 1
```

This loop prints even numbers from 15 to 0 , if the num odd the rest of this iteration will be skipped.

Else

After finishing the loop normally (without using break), if we need to execute specific code so we use else statements .

```
for num in range(10):
    if num%3==0:
        count+=1
else :
    print("there is ", count, " numbers divisible by 3 from 0 to 9")
```

Nested loop

What if we want to repeat the same loop a certain number of times?. What about puting loop inside another loop. That's called a nested loop.

```
for row in range(3):
    for col in range(3):
        print(col, end=" ")
    print()
```

Output:

```
0 1 2
0 1 2
0 1 2
```

Lists

The list is one of the most popular compound data types (sequences) in Python. It contains any number of values in square brackets [] separated by commas. Values can have different dataType. For example :

```
List = [1, "AUG", True, [1, 3] ]
```

It can contain lists as an item.

Creating List

- Empty list:

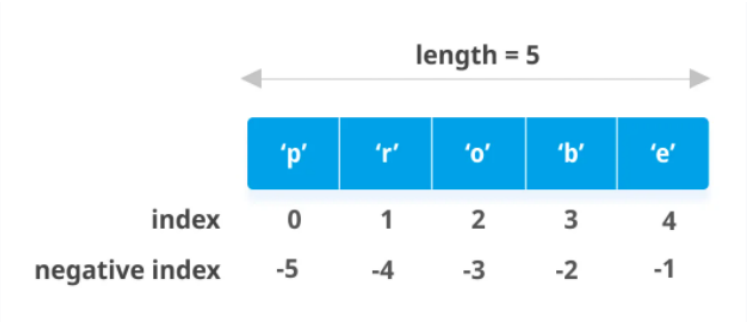
```
List = []
```

- List with initial values:

```
List = ["Hallo", 1, 2]
```

Access List Elements

We can use the [] operator to access an item in the list. Index starts from 0 or -1 to access items from the end of the list.



- List [2] get the third item in the list.
- List [0:3] get items in indices 0, 1, 2 (doesn't include 3)
- List [3:] get items from the index 3 to the end
- List [:: -1] get the list reversed

Update The List

You can change the value of the item in the list with another value that has any date Type.

```
List[2] = “change”
List [ : 2] = [1, true, 2]
```

To add a new item to the list we can use.append(item) The item will be added at the end of the list.

```
List.append(“add”)
```

Or using .insert(index, item) to add an item in a specific index.

```
List.insert(1,”add”)
```

You can also add a list to another list by using .extend(list) or + operator.

```
List1.extend(List2)
List1 += List2
```

To copy a list to another list we can use .copy(). If you change the copied list the second list won’t be affected (shadow copying).

```
List2=List1.copy()
```

Or use = operator. If you change any of the two lists the other list will change.

```
List1 = List2
```

Deleting From the List

- .remove(item)

  It uses for deleting the first appearance of a certain item in the list

- .pop()

  It uses to delete the last item in the list and return it.

- .clear()

  To delete all items in the list.

Additional Functions

- .sort()

  Sort the list that contains items that have the same data type

- len(object)

  Returns list's length (num of items).

- .count(item)

  Return the number of occurrences of a certain item

- .index(item)

  Return the first appearance of a certain item in the list

- .reverse()

  reverse the list.

List comprehension

shorter syntax to create a new list based on the values of an existing list (or string , tuple , dictionary ... etc) using for loop (and if condition if there).

```
newlist = [expression  for  x  in  sequence ]
newlist = [expression   for  x  in  sequence  if condition]
```

Example 1 (Copying list) :

```
requirements = ["apple", "milk", "kiwi", "meat"]
newRequirements = [x for x in requirements]
```

In this loop:

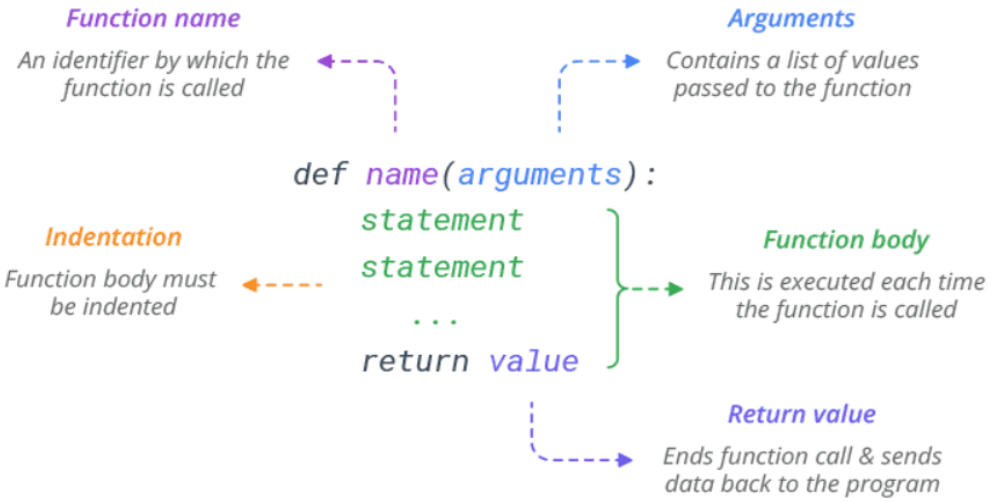
1. The for loop iterates on “requirements” list.
2. In each iterate append the current element in list

Example 2 (list consist of even num)

```
newList = [ num for num in range(100) if num % 2 == 0]
```

Functions

Definition :set of related statements that performs a certain task.
Advantages : it helps in organizing the code by breaking it out into small peaces. Also allow you to reuse the same code many times without writing it many times.



- def : keyword to mark the beginning of function's signature.after it we write function's name the parameters.
- Indentation is used to mark function scope.
- Function can have a return value in the function body.

Function's variables Scope :

Defined variables in the function are not visible outside it. After finishing its task all variables will be deleted from the memory.

```
def  changeValue():
        num = 10
        print("num value inside  : ", num)

num= 20
changeValue()
print("num value outside  : ", num)
```

Output :
num value inside : 10
num value outside : 20

Argument

Information that will be passed to the function. You can add many argument seperated by comma in the parentheses.

```
def functionName(arg1, arg2, arg3)
```

- Function can have no arguments.

```
def function()
```

- Arguments can have default value.

```
def functionName(arg1, arg2 = 2)
```

- If we don't know the number of argument we can use arbitrary arguments:
  - *arg
    the function will receive arguments as *tuple*

```
def addStudent(*info):
    students.append([info[0], info[1], totalGrades(info[2])])
addStudent(name, year, grades)
```

  - **arg
    the function will receive arguments as *dictionary*

```
def function(**kid):
    print("His last name is " + kid["lname"])

function(fname = "Tobias", lname = "Refsnes")
```

Function Call

To run the function we should call it and pass all needed parameters in the correct order.

- If the function has default value you can pass not all parameters
- You can pass parameters with the key = value syntax(order not matter)

```
def displayInfo(name, age):
    print("Your name is ", name)
    print( "Your age is : ",age)
displayInfo(age=10, name="Ahmed")
```