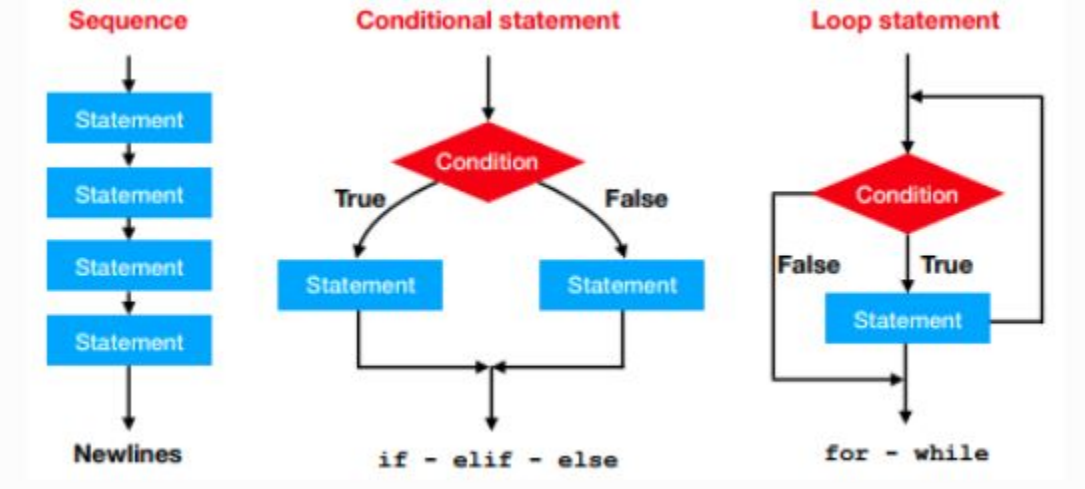
Execution flow

Three types:

Sequential

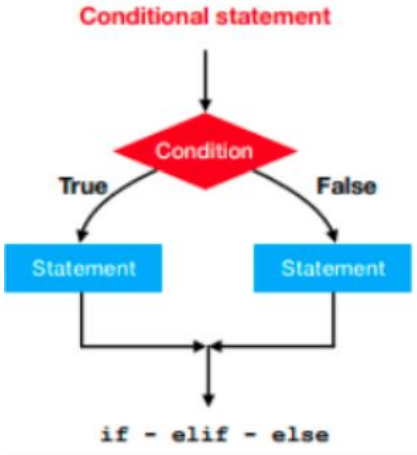
Conditional branch

Loop



If statement - basic

```
if condition :

    #This is the True branch
    #do something

else:

    #This is the False branch (or else branch)
    #do something else
```



Indentation is important in python!

If statement - basic

```
if condition :

    #This is the True branch
    #do something

else:

    #This is the False branch (or else branch)
    #do something else
```

Indentation is important in python!

THINGS TO REMEMBER

1. If condition is true the first branch is executed otherwise execution goes to the second;
2. A colon **":"** is placed after the condition and after else;
3. Indentation is used to specify block of codes;
4. **if** and **else** keywords are at the same indentation level;
5. **else** is optional

Scientific Programming Practical 4

Introduction

Luca Bianco - Academic Year 2020-21
luca.bianco@fmach.it

If statement - basic

Example: Let's get an integer from the user and test if it is even or odd, printing the result to the screen.

```
print("Dear user give me an integer:")
num = int(input())
res = ""
if num % 2 == 0:
    #The number is even
    res = "even"
else:
    #The number is odd
    res = "odd"

print("Number ", num, " is ", res)
```

```
Dear user give me an integer:
3
Number  3  is  odd
```

If - elif - else

```
if condition :

    #This is branch 1
    #do something

elif condition1 :

    #This is branch 2
    #do something

elif condition2 :

    #This is branch 3
    #do something

else:

    #else branch. Executed if all other conditions are false
    #do something else
```

Note that **branch 1** is executed if condition is **True**, **branch 2** if and only if **condition is False and condition1 is True**, **branch 3** if condition is **False, condition 1 is False and condition2 is True**. **If all conditions are False the else branch is executed**.

If - elif - else

Example: The tax rate of a salary depends on the income. If the income is < 10000 euros, no tax is due, if the income is between 10000 euros and 20000 the tax rate is 25%, if between 20000 and 45000 it is 35% otherwise it is 40%. What is the tax due by a person earning 35000 euros per year?

```
income = 35000
rate = 0.0

if income < 10000:
    rate = 0
elif income < 20000:
    rate = 0.2
elif income < 45000:
    rate = 0.35
else:
    rate = 0.4

tax = income*rate

print("The tax due is ", tax, " euros (i.e ", rate*100, "%)")
```

```
The tax due is  12250.0  euros (i.e  35.0 %)
```

Spot the difference!

```python
#Example 1

val = 10

if val > 5:
    print("Value >5")
elif val > 5:
    print("I said value is >5!")
else:
    print("Value is <= 5")
```

```python
#Example 2

val = 10

if(val > 5):
    print("\n\nValue is >5")

if(val > 5):
    print("I said Value is >5!!!")
```

Spot the difference!

```
#Example 1

val = 10

if val > 5:
    print("Value >5")
elif val > 5:
    print("I said value is >5!")
else:
    print("Value is <= 5")
```

Output:

```
Value >5
```

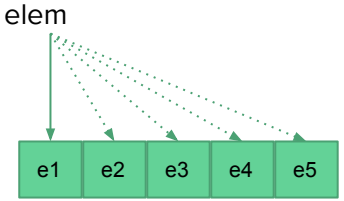
```
#Example 2

val = 10

if(val > 5):
    print("\n\nValue is >5")

if(val > 5):
    print("I said Value is >5!!!")
```

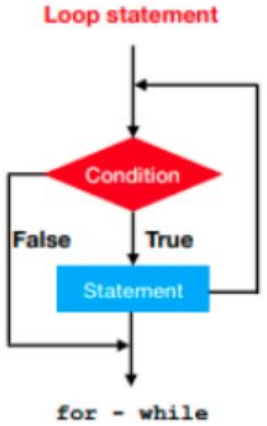
Output:

```
Value is >5
I said Value is >5!!!
```

For loops

```
for elem in collection :
    #OK, do something with elem
    # instruction 1
    # instruction 2
```





Depending on the type of the collection elem will get different values. Recall from the lecture that:

`str`	`for` iterates over the characters
`list`	`for` iterates over the elements
`tuple`	`for` iterates over the elements
`dict`	`for` iterates over the keys

For loops

```
S = "Hi there from python"
Slist = S.split(" ")
Stuple = ("Hi","there","from","python")
print("String:", S)
print("List:", Slist)
print("Tuple:", Stuple)

#for loop on string
print("On strings:")
for c in S:
    print(c)

print("\nOn lists:")
#for loop on list
for item in Slist:
    print(item)

print("\nOn tuples:")
#for loop on list
for item in Stuple:
    print(item)
```

```
String: Hi there from python
List: ['Hi', 'there', 'from', 'python']
Tuple: ('Hi', 'there', 'from', 'python')
On strings:
H
i

t
h
e
r
e

f
r
o
m

p
y
t
h
o
n

On lists:
Hi
there
from
python

On tuples:
Hi
there
from
python
```

Looping over a range

Given E, S and step integers

```
range(E)        # ranges from 0 to E-1
range(S,E)      # ranges from S to E-1
range(S,E,step) # ranges from S to E-1 with +step jumps
```

Remember:

S included,
E excluded!

Example: Given a list of integers, return a list with all the even numbers.

```
myList = [1, 7, 9, 121, 77, 82]
onlyEven = []

for i in range(0, len(myList)):  #this is equivalent to range(len(myList)):
    if myList[i] % 2 == 0 :
        onlyEven.append(myList[i])

print("original list:", myList)
print("only even numbers:", onlyEven)
```

```
original list: [1, 7, 9, 121, 77, 82]
only even numbers: [82]
```

Range

Note: range works differently in Python 2.x and 3.x

In Python 3 the *range* function returns an iterator rather storing the entire list.

In [7]:
```
#Check out the difference:
print(range(0,10))

print(list(range(0,10)))
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Looping over a range

Example: Let's consider the two DNA strings s1 = "ATACATATAGGGCCAATTATTATAAGTCAC" and s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC" that have the same length. Let's create a third string out such that $out[i]$ is " | " if $s1[i] == s2[i]$, " " otherwise.

```
s1 = "ATACATATAGGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC"

outSTR = ""
for i in range(len(s1)):
    if s1[i] == s2[i]:
        outSTR = outSTR + "|"
    else:
        outSTR = outSTR + " "

print(s1)
print(outSTR)
print(s2)
```

```
ATACATATAGGGCCAATTATTATAAGTCAC
   ||  || |  |  |    |  ||||| |
CGCCACTTAAGCGCCCTGTATTAAAGTCGC
```

Nested for loops

It is possible to place one for loop into another one (or more..)

```
for i in collection:
    for j in another_collection:
        #do some stuff with i and j
```

Nested for loops

Given the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ stored as a list of lists (i.e. matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]].

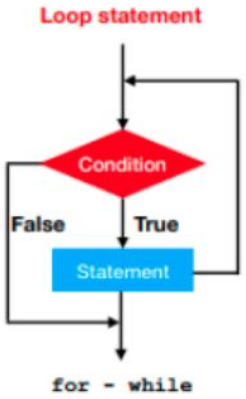
Print it out as:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for i in range(len(matrix)):
    line = ""
    for j in range(len(matrix[i])):
        line = line + str(matrix[i][j]) + " " #note int --> str conversion!
    print(line)
```

While loops

```
while condition:

    #do something
    #update the value of condition
```



Useful to loop until a specific condition is True

Print the numbers from 0 to 4

Anything wrong with this?

```
i = 0
while i < 5:
    print("i now is:", i)
    i = i + 1 #THIS IS VERY IMPORTANT!
```

```
i now is: 0
i now is: 1
i now is: 2
i now is: 3
i now is: 4
```

Note: The loop will continue until *condition* holds true and the only code executed is the block defined through the indentation. This block of code must update the value of condition otherwise the interpreter will get stuck in the loop and will never exit.

Nesting while and for loops

```
for i in range(1,10):
    j = 1
    output = ""
    while j<= i:
        output = str(j) + " " + output
        j = j + 1
    print(output)
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

Let’s print the following picture to std out

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

Nesting while and for loops

spot the difference...

```
for i in range(1,10):
    j = 1
    output = ""
    while j<= i:
        output = str(j) + " " + output
        j = j + 1
    print(output)
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

```
for i in range(1,10):
    j = 1
    while j<=i:
        print(j, end = " ")
        j = j + 1
    print("")
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

Let's print the following picture to std out

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

http://qcbsciprolab2020.readthedocs.io/en/latest/practical4.html

?

Exercises

1. Given the integer 134479170, print if it is divisible for the numbers from 2 to 16. Hint: use for and if.

Show/Hide Solution

2. Given the DNA string

```
DNA="GATTACATATATCAGTACAGATATATACGCGCGGGCTTACTATTAAAAACCCC"
```

write a Python script that reverse-complements it. To reverse-complement a string of DNA, one needs to replace and A with T, T with A, C with G and G with C, while any other character is complemented in N. Finally, the sequence has to be reversed (e.g. the first base becomes the last). For example, ATCG becomes CGAT.

Show/Hide Solution

3. Count how many of the first 100 integers are divisible by 2, 3, 5, 7 but not by 10 and print these counts. Be aware that a number can be divisible by more than one of these numbers (e.g. 6) and therefore it must be counted as divisible by all of them (e.g. 6 must be counted as divisible by 2 and 3).

Show/Hide Solution

4. Write a python script that creates the following pattern:

```
+
++
+++
++++
+++++
++++++
+++++++ <-- 7
++++++
+++++
++++
+++
++
+
```