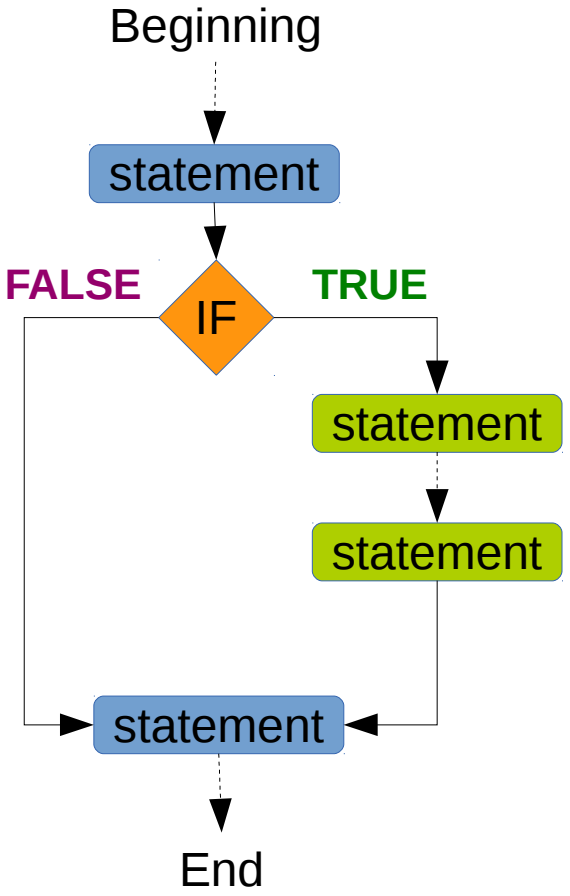
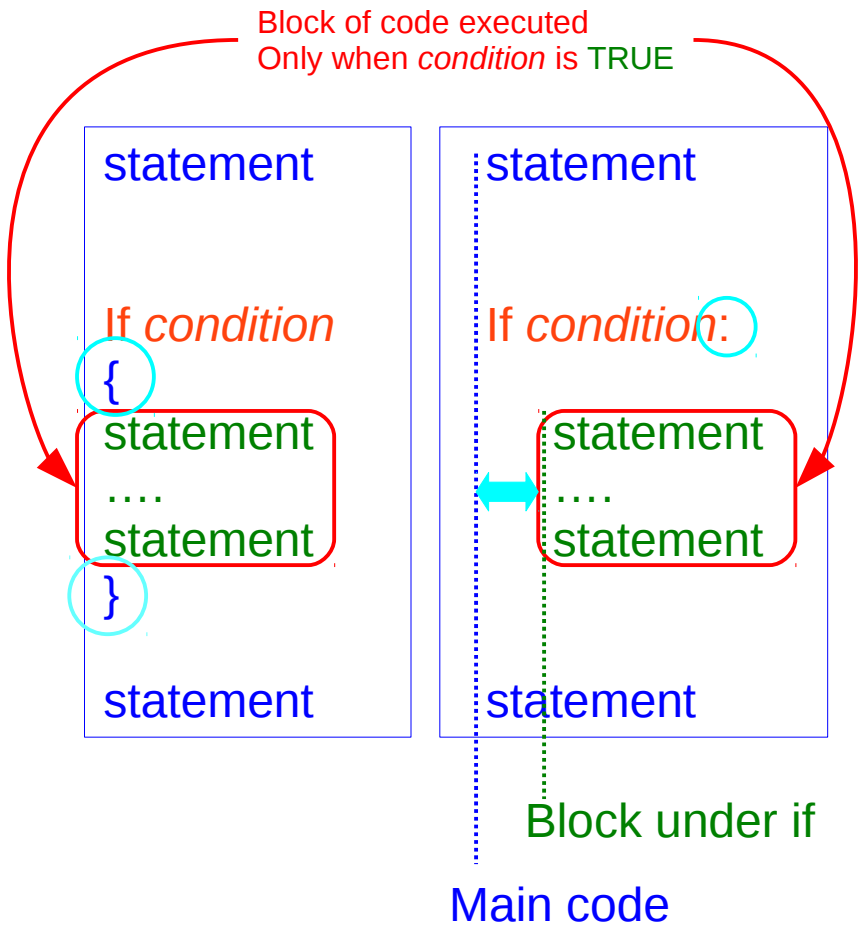
Flow control: decisions

Conditional decisions enable a program to follow different courses of action depending on the states of variables.





Actual example

```
x=input("Enter a number: ")
y=int(x)
if y < 0 :
      print ("condition true")
      print ("It is negative")
print ("End")
```

Homework: why do we need to use the function int(x)?

Block under if

Main code

We need a way to differentiate the code to be executed when the condition is met from the rest of the code. Each programming language uses a specific label. In many cases (e.g. Perl and R) the block of code is included within curly brackets: {}. **In python the code is *indented (*that is moved to the right a given number of spaces) and the line leading to the block ends with a colon (:)**.

Important: be consistent with the indentation

Unit2_example1.py

Flow control: decisions

1. INTERACTIVE PYTHON INTERPRETER

```
>>> x=input("Enter a number: ")
Enter a number: -1
>>> y=int(x)
>>> if y<0 :
...     print("condition true")
...     print("It is negative")
...
condition true
It is negative
>>> print ("End")
End
>>>
```

Python shell (interactive python interpreter)

Type some spaces Between "..." and commands (INDENTATION)

Type SAME # of spaces as before

Press enter ONLY (no spaces) to indicate end of code block.

Important: it is critical to be consistent with the indentation within a block

2. PYTHON SCRIPT

```
x=input("Enter a number: ")
y=int(x)
if y<0 :
    print("condition true")
    print("It is negative")
print ("End")
```

Type some spaces before commands (INDENTATION)

Type SAME # of spaces as before

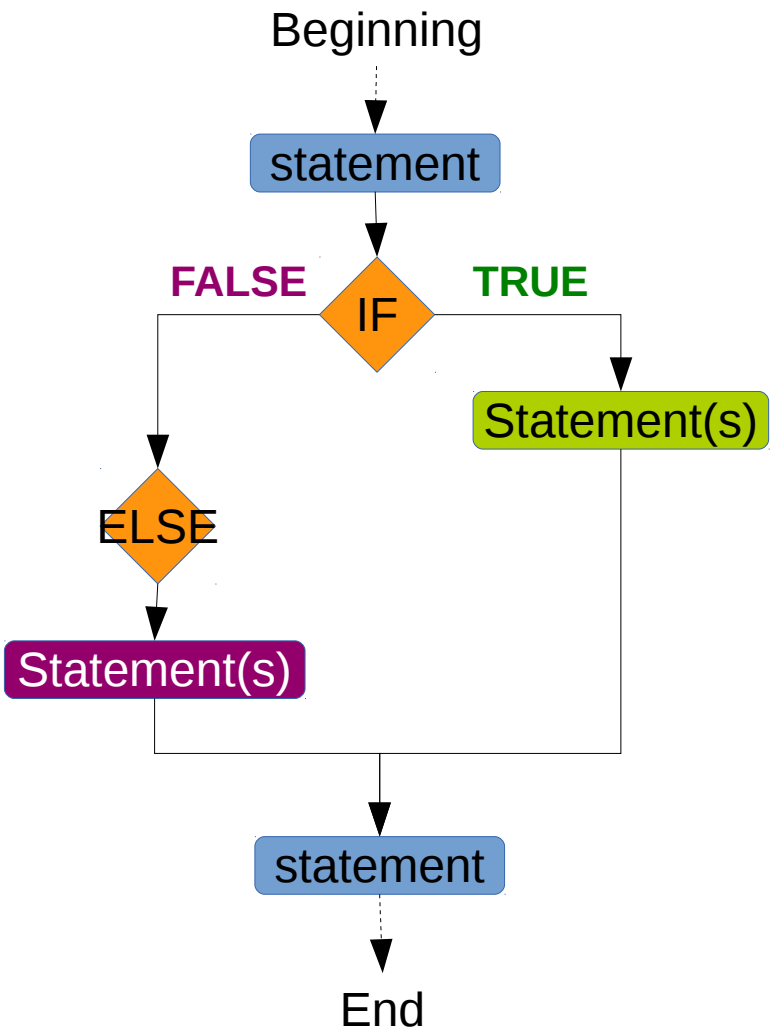
Type in your favourite text editor.
Save it as:
MyFirstScript.py
(or any other name)

```
usuario@Master-24:~$ python3 UNit4_example1.py
Enter a number: -1
condition true
It is negative
End
usuario@Master-24:~$
```

Note that this is a regular terminal (bash shell)
NOT a python shell

Important: it is critical to be consistent with the indentation within a block

Flow control: Two-way decisions (if-else)



Block of code executed
Only when *condition* is TRUE

Block of code executed
Only when *condition* is FALSE

statement

If *condition*:

statement
....

Else

statement
....

statement

Block under if/else

Main code

Actual example

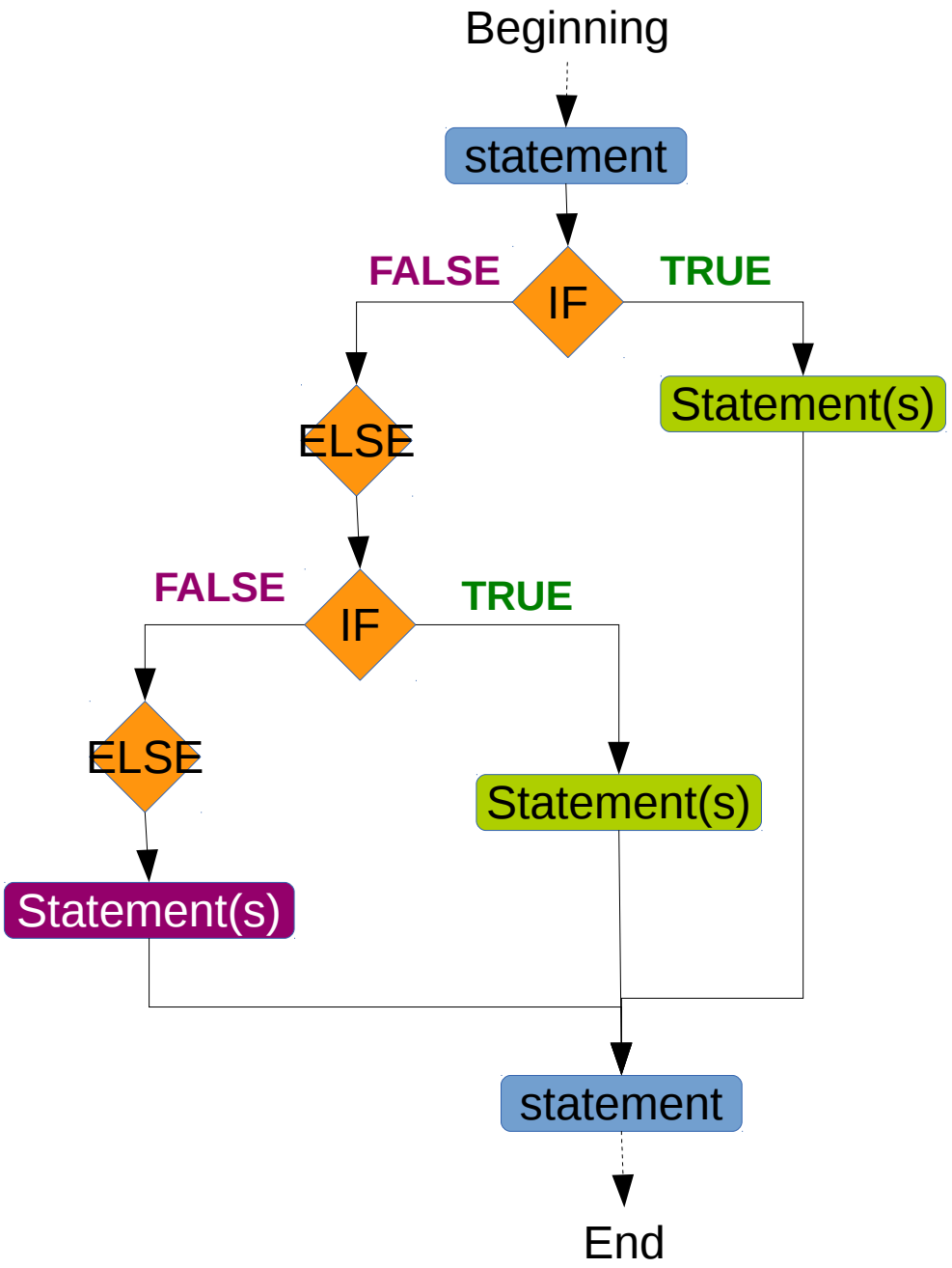
```
x=input("Enter a number: ")
y=int(x)
If y < 0 :
        print ("condition true")
        print ("It is negative")
else:
        print ("condition false")
        print ("It is positive)
print ("fin del programa")
```

Block under if

Main code

Unit2_example2.py

Flow control: Multi-way decisions (nested if-else)



Actual example

```
x=input("Enter a number: ")
y=int(x)
If y < 0 :
        print (" first condition true")
        print ("It is negative")
else:
        if y > 0:
                print ("second condition true")
                print ("It is positive")
        else:
                print ("1st and 2nd conditions false")
                print ("It is zero")
print ("fin del programa")
```

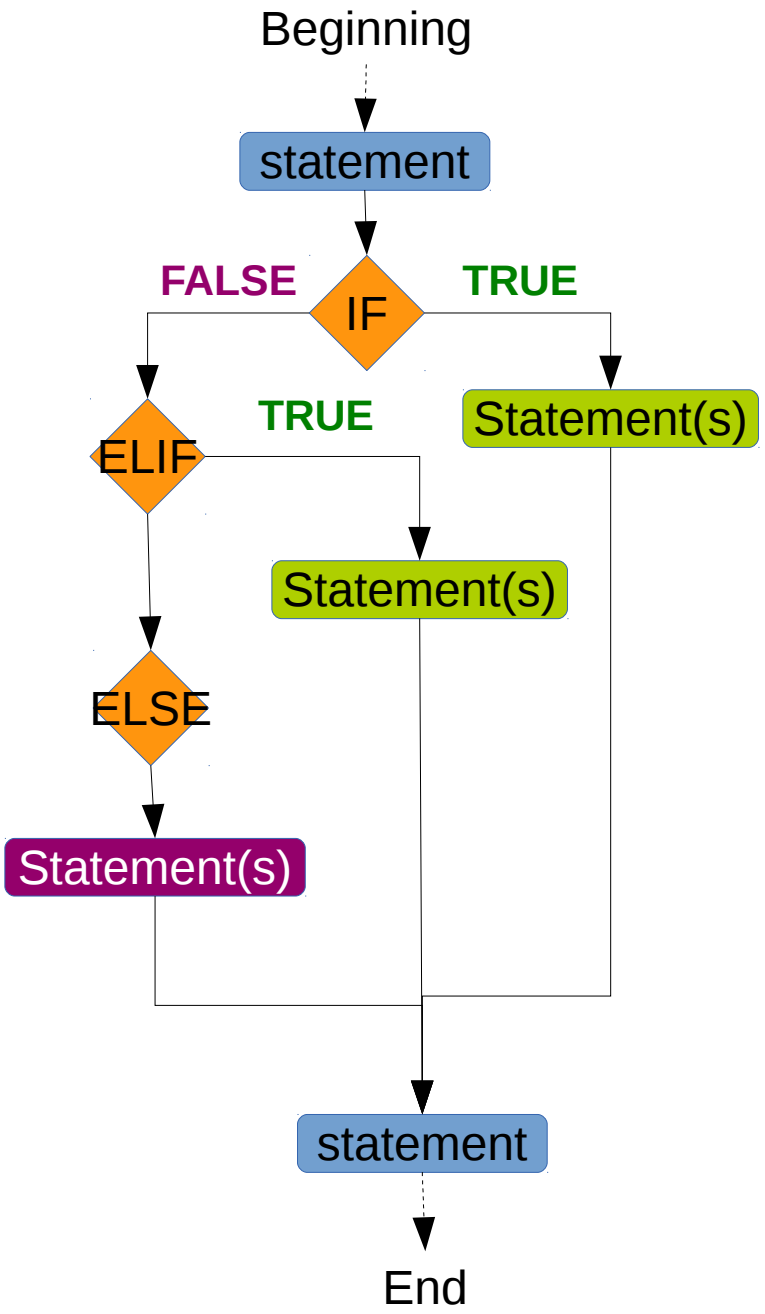
Block second first if/else

Block under first if/else

Main code

Unit2_example3.py

Flow control: Multi-way decisions (elif)



Actual example

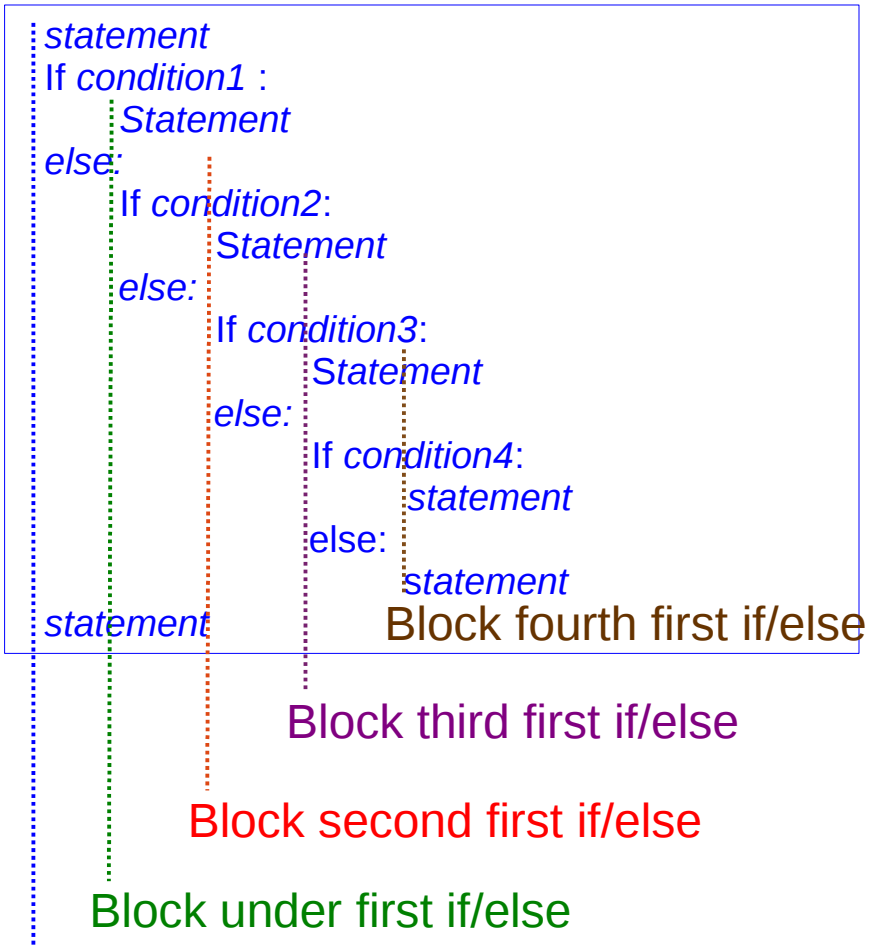
```
x=input("Enter a number: ")
y=int(x)
If y < 0 :
        print (" first condition true")
        print ("It is negative")
elif y > 0:
        print ("second condition true")
        print ("It is positive)
else:
        print (1st and 2nd conditions false")
        print ("It is zero")
print ("fin del programa")
```

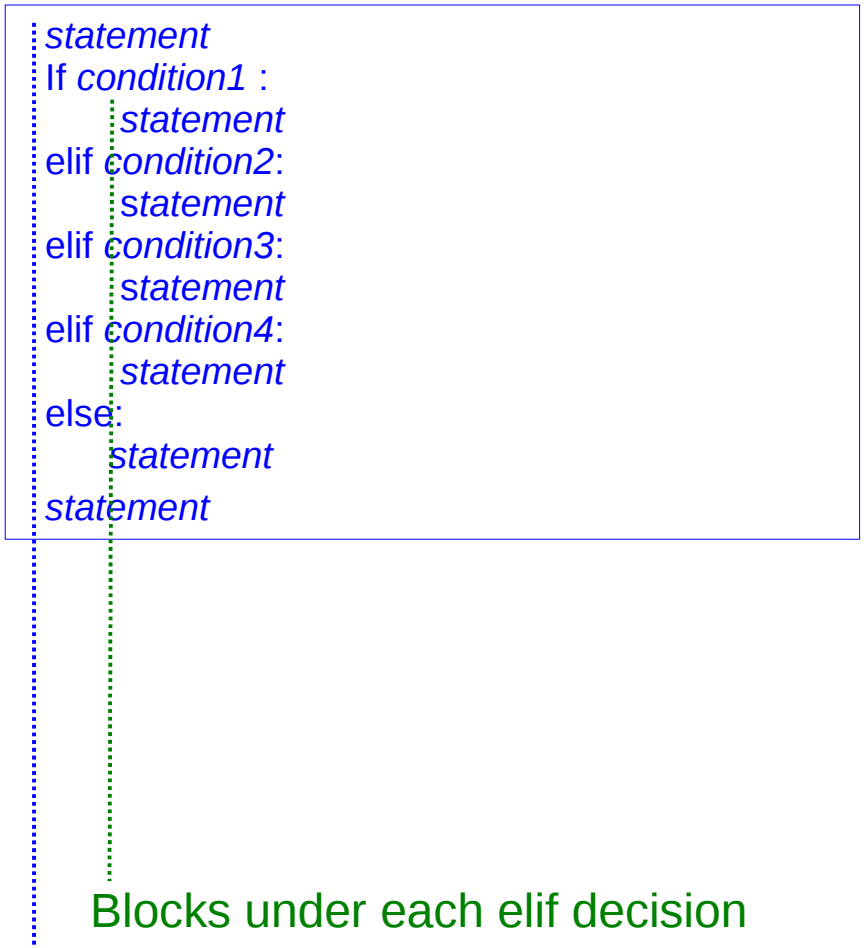
Block under first if/else

Main code

Unit2_example4.py

Flow control: Two-way decisions (nested vs elif)





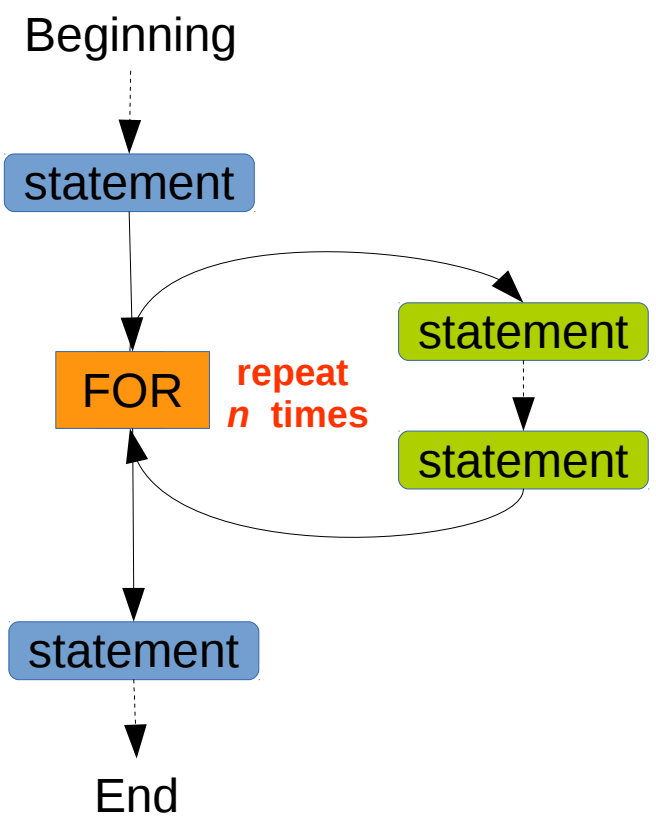
Note: it is not mandatory to include the else clause in neither of the cases.

Loops: for-type and while-type

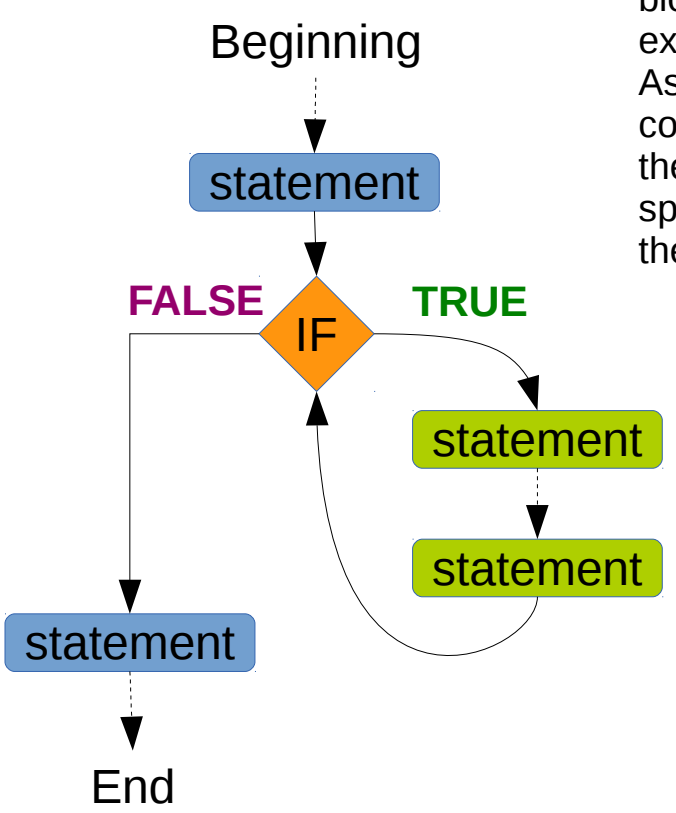
Loops repeat a block of code several times. There are two main types:

- "***for***" loops.- The block is repeated ***for*** a predetermined number of times
- "***while***" loops.- The block is repeated ***while*** a condition is true.

FOR loops

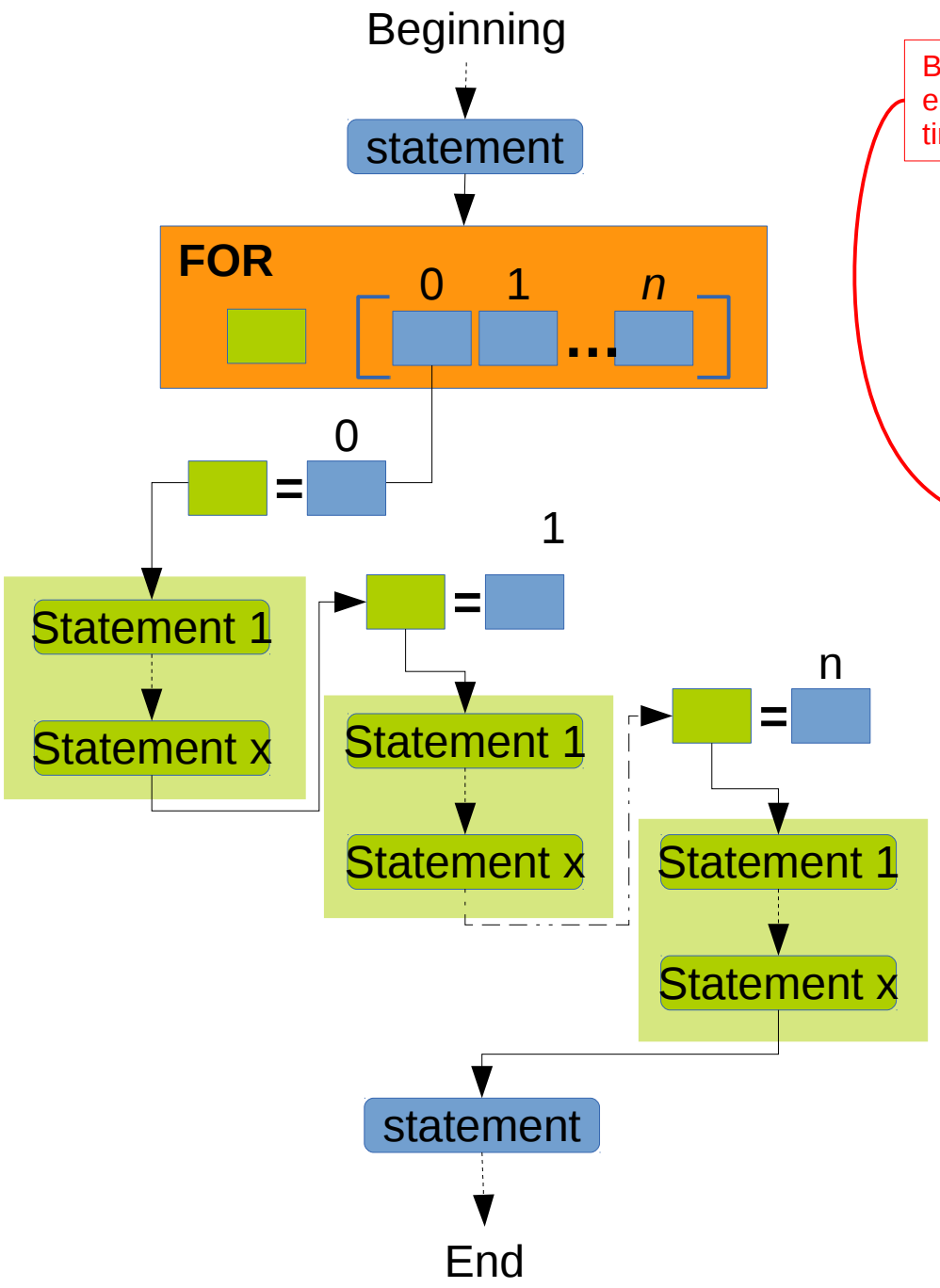


WHILE loops



As in the case of decisions, we need a way to differentiate the block of code to be repeatedly executed.
As in decisions, in python, the code is *indented (t*hat is moved to the right a given number of spaces) and the line leading to the block ends with a colon (:).

"for" loops (*definite* or *counted* loops)



Block of code executed for each element in *list* (as many times as elements in *list*.

```
statement

for var in list

    Statement 1
    ....
    Statement x

statement
```

Block under for

Main code

Actual example

```
# times table
x=7
y=[0,1,2,3,4,5,6,7,8,9,10]
for z in y:
        out=str(x)+" x "+str(z)+" = "
        print (out, x*z)
print ("fin del programa")
```

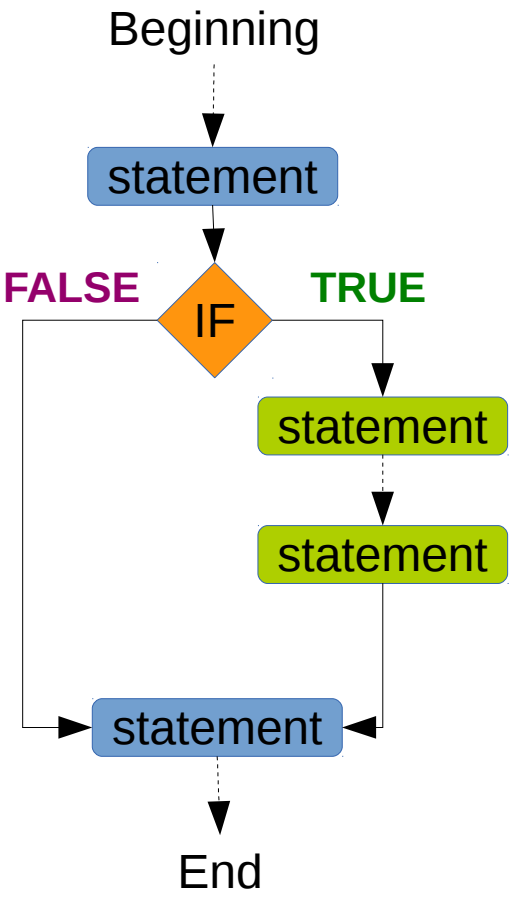
Block under for

Main code

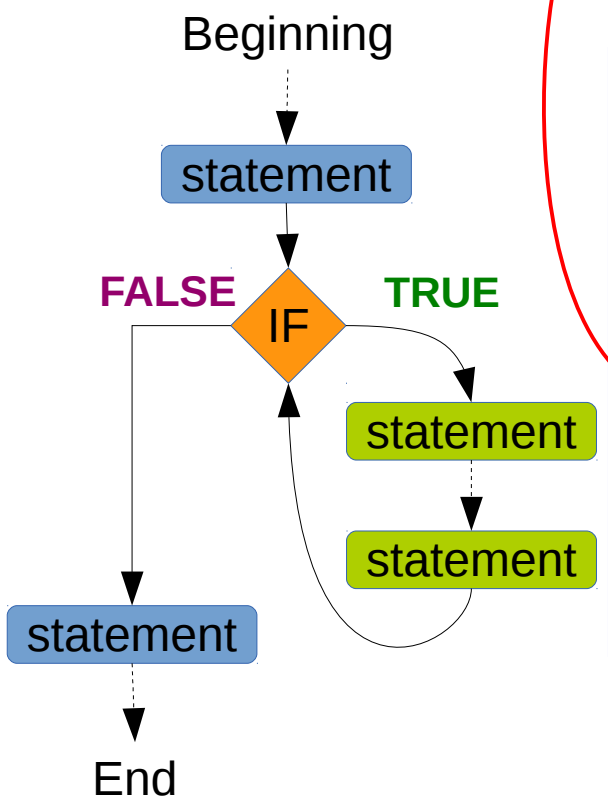
Unit2_TimesTable_v1.py

"while" loops (*indefinite* or *conditional* loops)

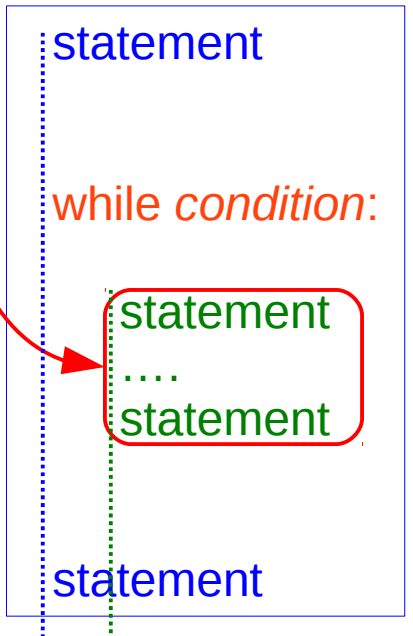
_if_ conditional



while loop



Block of code executed
Only WHILE *condition* is TRUE



Actual example

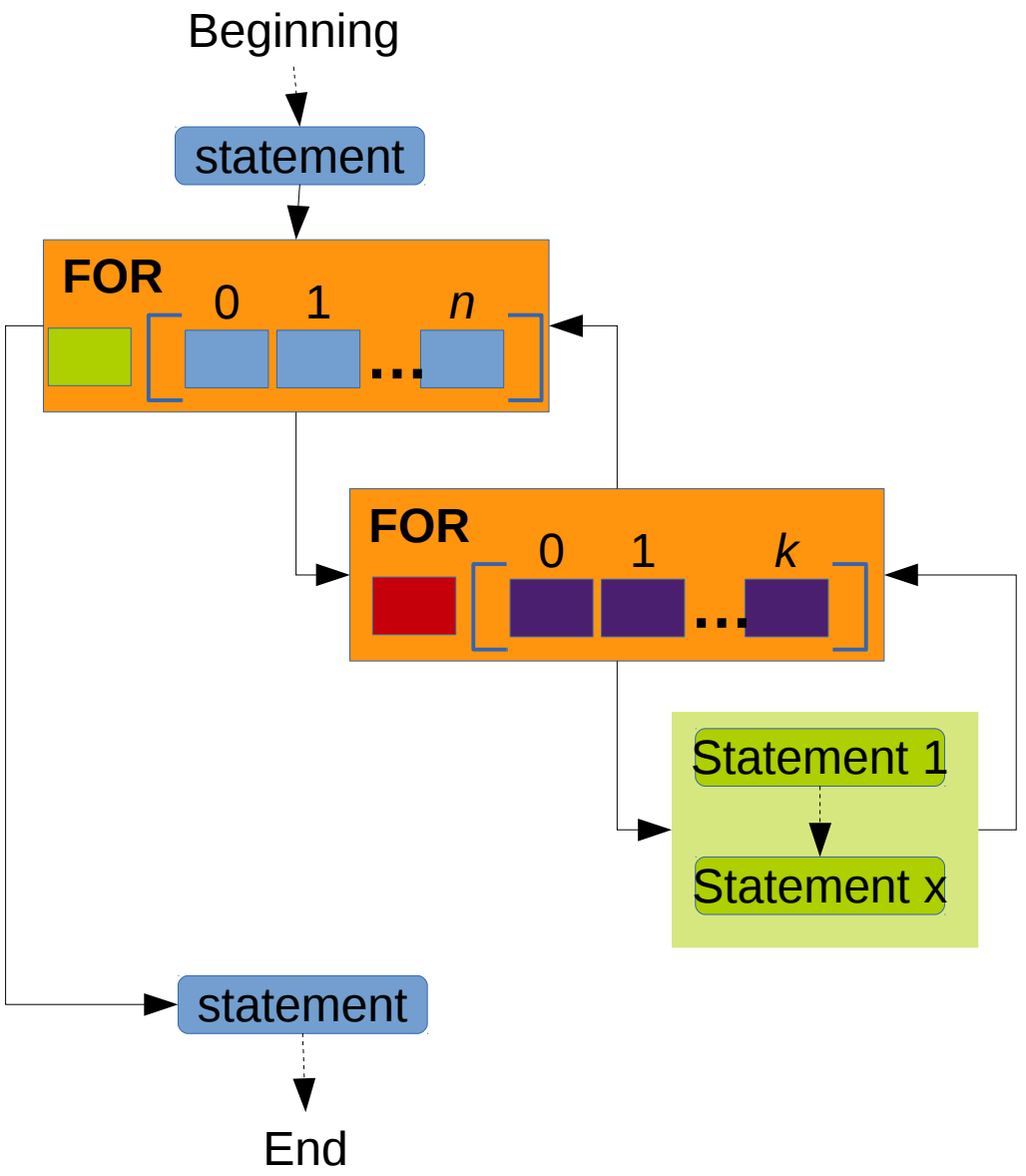
```
# times table version 2
x=7
k=0
while k<11:
        out=str(x)+" x "+str(k)+" = "
        print (out, x*k)
        k=k+1
print ("fin del programa")
```

Block under while
Main code

Note that, if the logical test is never false, the program will keep looping forever in an "infinite loop".

Unit2_TimesTable_v2.py

Nested loops



Actual example

```
# times tables version 3
x=[0,1,2,3,4,5,6,7,8,9,10]
y=[0,1,2,3,4,5,6,7,8,9,10]
for z in x:
        print (z," Times Table")
        for k in y:
                out=str(z)+" x "+str(k)+" = "
                print (out, z*k)
print ("fin del programa")
```

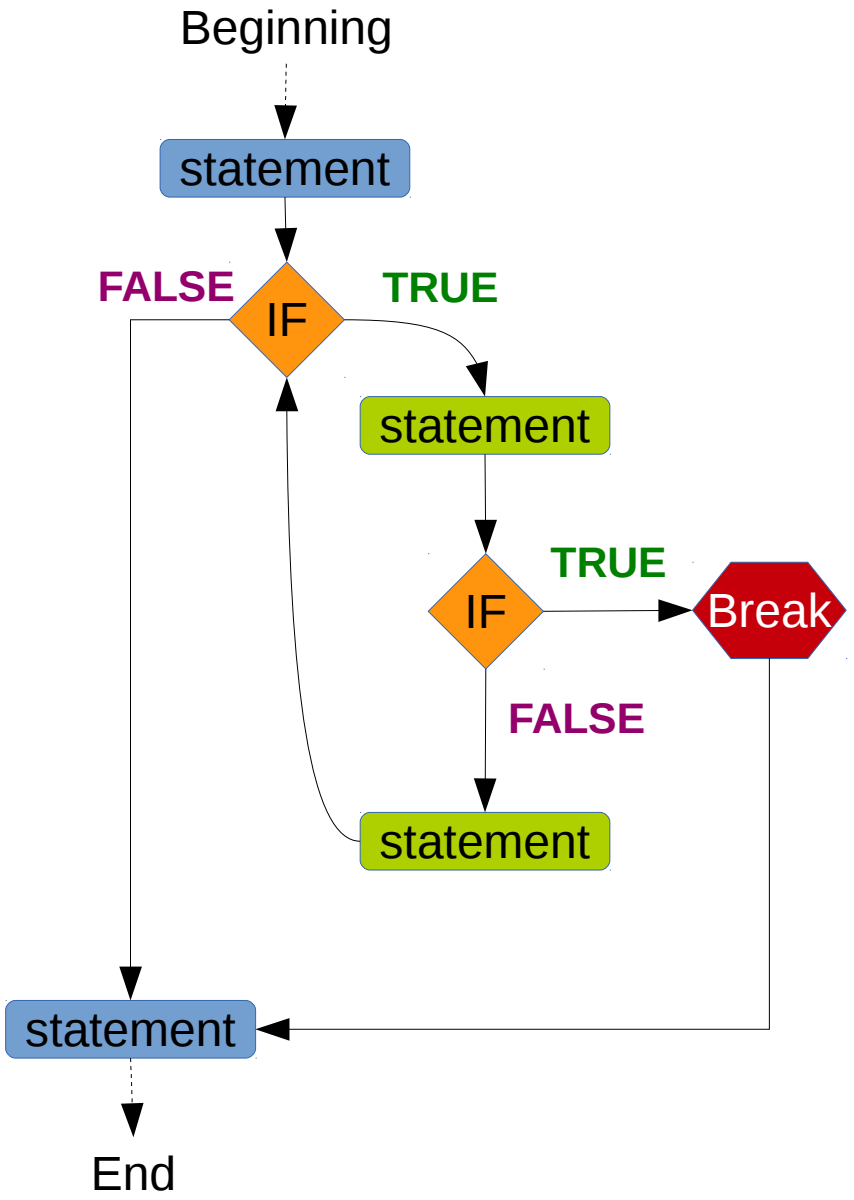
Block under first loop

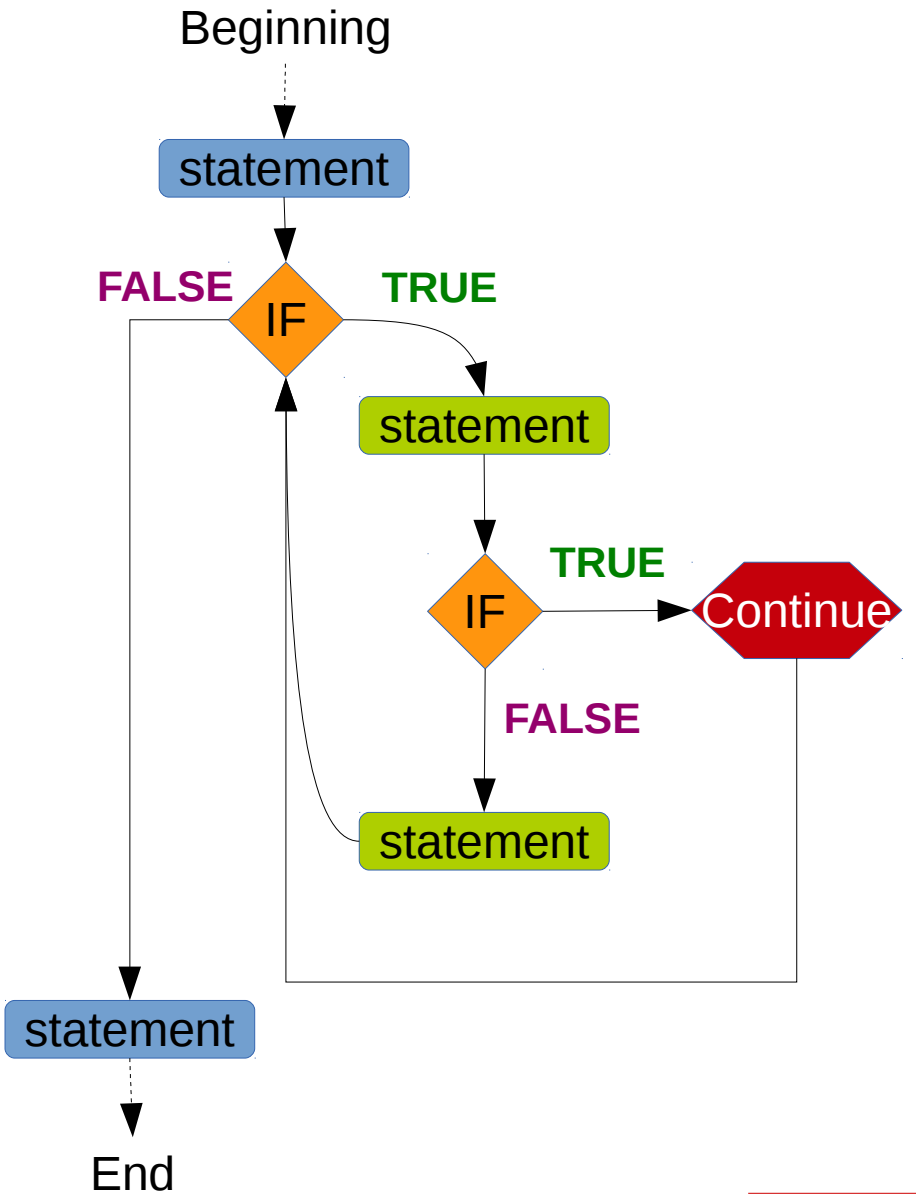
Block under first loop

Main code

Unit2_TimesTable_v3.py

Exiting loops: break and continue





Examples of exiting a loop and *range* function

Actual example

```
# times tables version 4
z=-1
y=range(10)
while z<10:
        z=z+1
        print (z," Times Table")
        q=input('Press "e" to exit, "s" to skip this    number and any other key to continue. ')
        if q=="e":
                break
        elif q=="s":
                continue
        for k in y:
                out=str(z)+" x "+str(k)+" = "
                print (out,z*k)
print ("End")
```

Block under first loop

Block under first loop

Main code

Explore range() function.

Optional homework:
Modify Unit2_example4.py so that it keeps asking the user to input a number until he/she types *q* (letter “q”) or any other character of your choice.

Some on-line programming aids

We'll see these resources in Unit2

- http://pythontutor.com/
- http://scratch.mit.edu/
- https://github.com/
- http://blog.coderscrowd.com/real-time-programming-for-bioinformatics-and-for-fun/