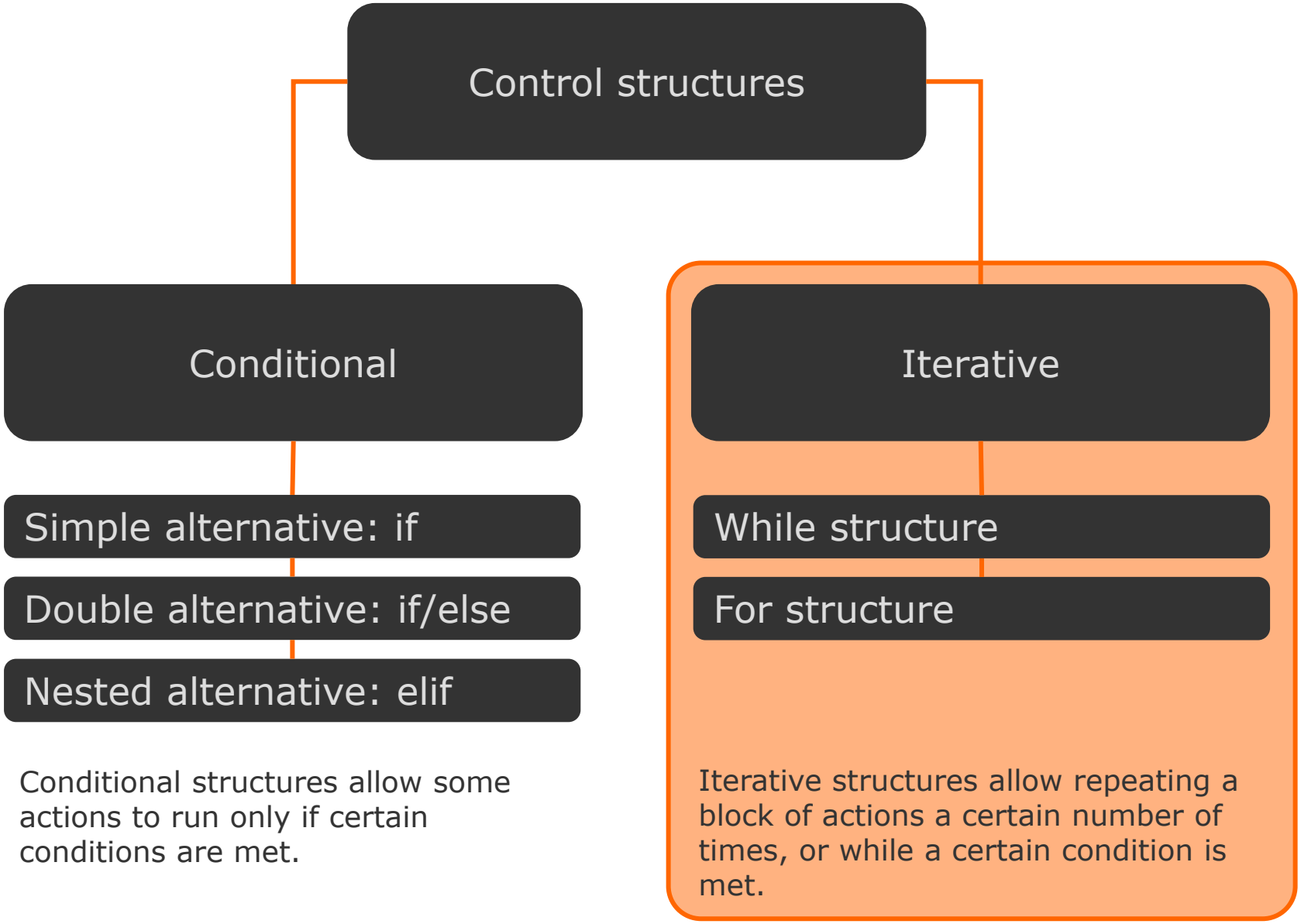
Iterative control structures

Control structures



Example 1

What actions should we do if we want to write natural numbers from 1 to 10 (one on each line) on the screen?

We could do:

```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

This program would work but it is not a good solution....

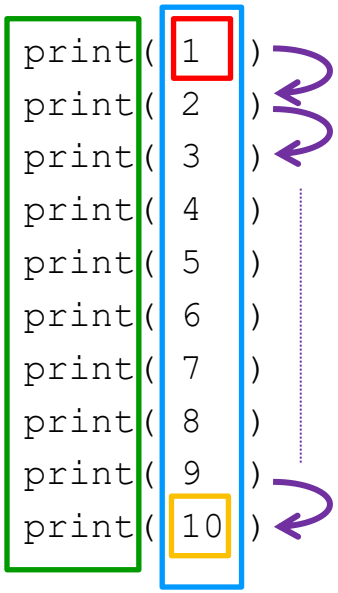
.... what if we want to write the first 1000 natural numbers?

We don't want to have to write 1000 lines of code!!!

Luckily, we have the iterative instructions.

Example 1

What actions should we do if we want to write natural numbers from 1 to 10 (one on each line) on the screen?



To use the iterative instructions we must answer 4 basic questions:

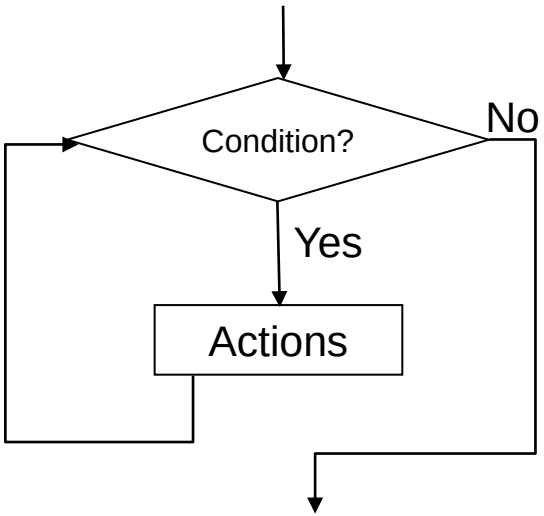
- What variables do we need?
- How should we initialize them?
- What actions should we repeat?
    - Update
- How do we know when to finish?

Iterative structures: while

Syntax

```
while (<condition>):
    <actions>
```

Flowchart



Example

```
number = 1

while (number <= 10):
    print(number)
    number+=1
```

Variable
Initialization
Condition for repetition
Repeated actions
Update

Example 2

What actions should we do if we want to add all the natural numbers from 1 to 10?

NOT VALID

1+2+3+4+5+6+7+8+9+10

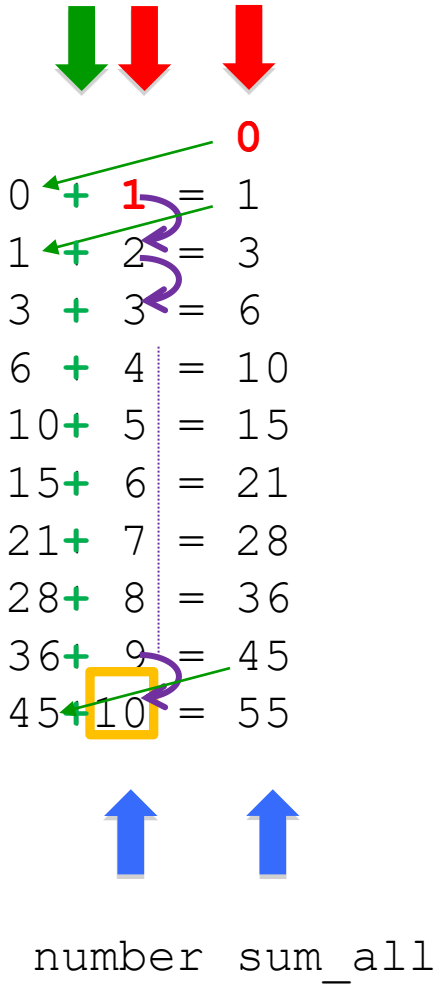
We are looking for a solution where the sum operation is repeated:

```
                0
 0 +  1 =  1
 1 +  2 =  3
 3 +  3 =  6
 6 +  4 = 10
10 +  5 = 15
15 +  6 = 21
21 +  7 = 28
28 +  8 = 36
36 +  9 = 45
45 + 10 = 55
```

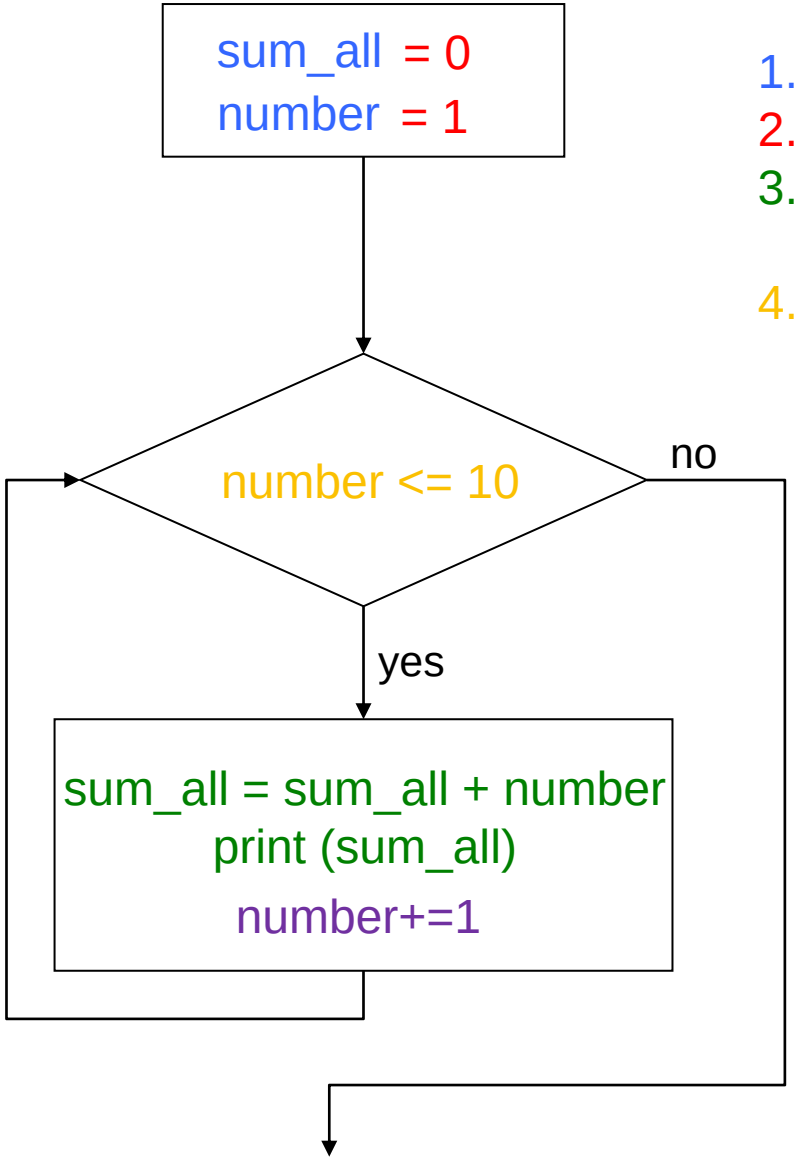
Example 2

What actions should we do if we want to add all the natural numbers from 1 to 10?

- What variables do we need?
- How should we initialize them?
- What actions should we repeat?
  - Update
- How do we know when to finish?



Example 2 (while)



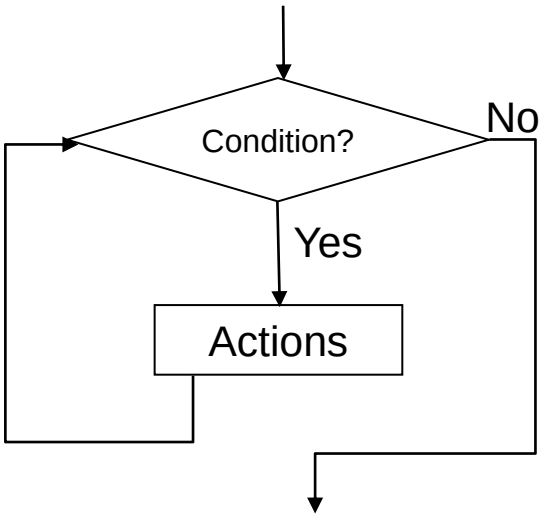
1.- What variables do we need?
2.- How should we initialize them?
3.- What actions should we repeat?
Update
4.- How do we know when to finish?

Iterative structures: while

Syntax

```
while (<condition>):
    <actions>
```

Flowchart



Example

```
number = 1
sum_all= 0

while (number <= 10):
    sum_all = sum_all + number
    print(sum_all)
    number+=1
```

Variable
Initialization

Condition for repetition

Repeated actions

Update

Iterative structures: while

What happens if we change one of the parts of the loop?

1,2,3,4, … 11

10 iterations

```
number = 1
sum_all = 0

while (number <= 10):
    sum_all = sum_all + number
    print(number, sum_all)
    number+=1
```

```
1  1
2  3
3  6
4  10
5  15
6  21
7  28
8  36
9  45
10 55
```

num sum

Iterative structures: while

What happens if we change one of the parts of the loop?

1,2,3,4, … 11

10 iterations

```
number = 1
sum_all = 0

while (number <= 10):
    number+=1
    sum_all = sum_all + number
    print(number, sum_all)
```

1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

num sum

2	2
3	5
4	9
5	14
6	20
7	27
8	35
9	44
10	54
11	65

Iterative structures: while

What happens if we change one of the parts of the loop?

```
number = 0
sum_all = 0

while (number <  10):
    number+=1
    sum_all = sum_all + number
    print(number, sum_all)
```

0,1,2,3,4, … 10

10 iterations

1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

num sum

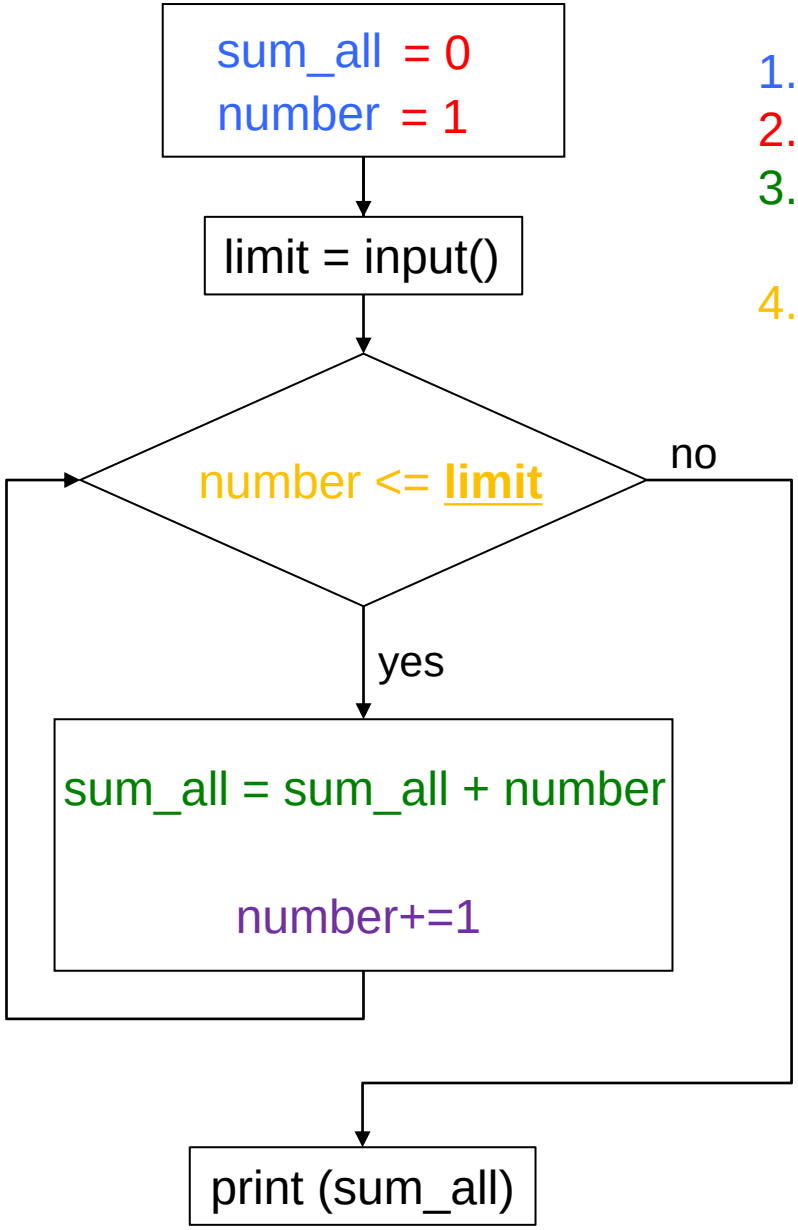
2	2
3	5
4	9
5	14
6	20
7	27
8	35
9	44
10	54
11	65

1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

Example 3

Modify the program in the example above to add all the natural numbers smaller than a number `n` entered by the user.

Example 3 (while)



1.- What variables do we need?
2.- How should we initialize them?
3.- What actions should we repeat?
Update
4.- How do we know when to finish?

```
number = 1
sum_all = 0

limit = int(input("Enter Number:"))

while (number <= limit):
    sum_all = sum_all + number
    number += 1

print(sum_all)
```

Example 3

Modify the program in the example above to add all the natural numbers smaller than a number `n` entered by the user.

- What happens if `n<0` ?

  Modify the program again to ensure that `n>=0`.

  If `n<0` the program must ask for another number again, until a positive one is entered.

  How can we do it using an iterative structure?

Example 3

```
number = 1
sum_all = 0

limit = int(input("Enter a number to compute the sum: "))
while (limit < 0):
    print("Error: The number must be higher than zero")
    limit = int(input("Enter a number to compute the sum: "))

while (number <= limit):
    sum_all = sum_all + number
    number += 1

print("The summation of ”,limit,"is:”,sum_all)
```

Exercise: Multifunction calculator

Write a program that simulates a calculator for real numbers.

- We will start from the program implemented in previous lecture: It asked the user to enter two numbers through the keyboard, showed a menu with the available operations, and asked the user what operation wanted to do.

- **1st version**: In the menu, we will **add an "Exit" option**.
  As long as the user does not select this option, the program will allow the user to select operations and will show the result on the screen. After that, the program will show the menu again.

- **2nd version**: Once the user has selected the option to exit, the program will ask if he/she wants to change the operands.
  - If the user answers **"Y"**, the program will **go back to the beginning** (ask the user to enter two numbers)
  - If the user answers **"N"**, the program will **end**.
  - Otherwise, the program will show an **error message** and will **repeat the question**.

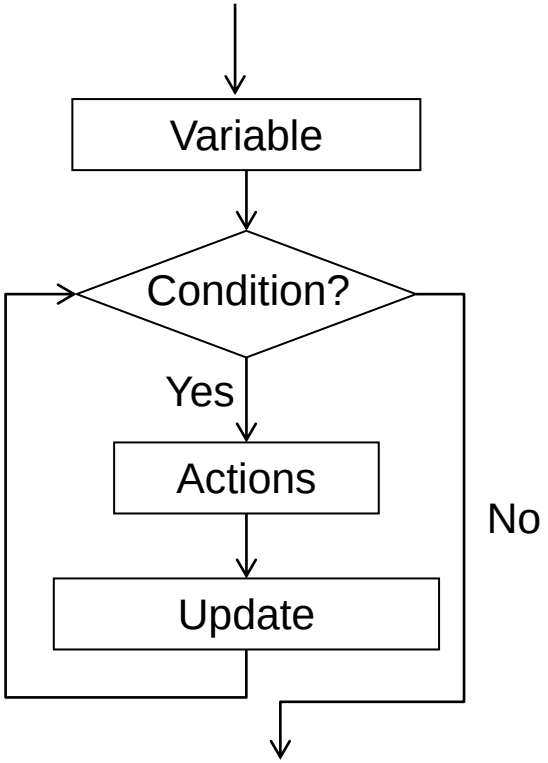
Iterative structures: for

The **for** structure is basically used when we want to execute a set of actions a **given number of times**.

Syntax

```
for <variable> in range(<start>,<stop>,<step>):
    <actions>
```

Flowchart




```
range(<start>,<stop>,<step>))
```

Function that generates the sequence of numbers starting at <start> and ending at <stop> -1 with a step <step>

By default, <start>= 0 and <step>= 1

Example

```
range(11)
    Generates 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
range(2,11,2)
    Generates 2, 4, 6, 8, 10
```

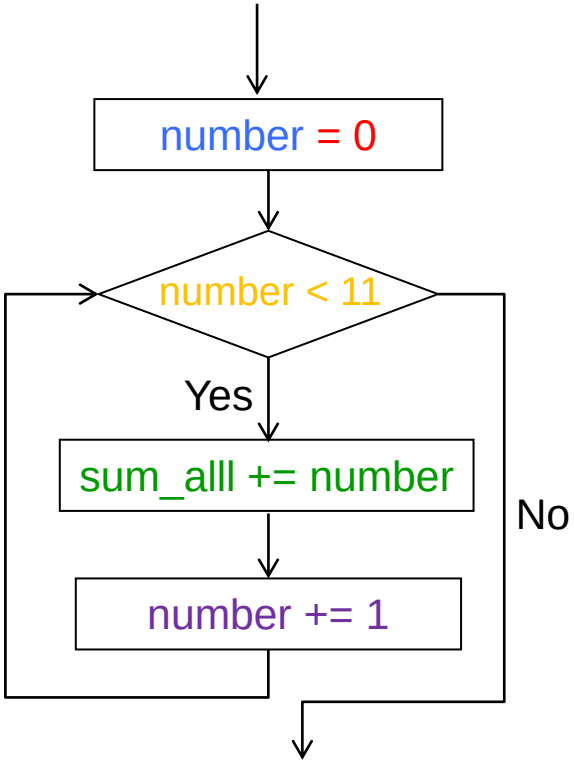
Iterative structures: for

The **for** structure is basically used when we want to execute a set of actions a **given number of times**.

Syntax

```
for <variable> in range(<start>,<stop>,<step>):
    <actions>
```

Flowchart



Example (summation)

```
suma_all = 0

for number in range(11):
    suma_all += number

print("Summation: ", sum_all)
```

Example 3: summation (for)

Modify the program in the example above to add all the natural numbers smaller than a number `n` entered via keyboard.

– What happens if `n<0` ?

Modify the program again to ensure that `n>=0`.

If `n<0` the program must ask for another number again, until a positive one is entered.

How can we do it using an iterative structure?

In this case:

- Use a **for structure** to compute the **sum**.
- Use a **while structure** to ensure the **number n is positive** and show an error message otherwise.

Example 3: summation (for)

```
sum_all = 0

limit = int(input("Enter a number to compute the sum: "))
while (limit < 0):
    print(" Error: The number must be higher than zero ")
    limit = int(input("Enter a number to compute the sum: "))

for number in range(limit+1):
    sum_all = sum_all + number

print(sum_all)
```

Exercise: Count positives and negatives

Write a program that reads 10 numbers from the keyboard and tells us how many positive numbers and how many negative numbers we have entered.

Zero does not count neither as positive nor as negative.

Exercise: Statistics of marks

Write a program that calculates some statistics from a series of marks entered by the user.

The program must ask the user how many marks he/she is going to enter.

Then, the program must ask the user to enter the marks one by one.

After that, the program must show:

- The **mean** of all the marks.
- **How many** students got **each grade** (A with Honors, A, B, C, Fail).

Remember the grades:

Fail	mark in the interval [0,5)
C	mark in the interval [5,7)
B	mark in the interval [7,9)
A	mark in the interval [9,10)
A with Honors	mark is 10

While vs for Loops

	`while`	**`for`**
Condition Type	Any	i € [start .. stop)
Nº of repetitions	It depends on condition: 0 or +	Known 0 or +
Counter	Must be initialized and updated	Managed by the instruction itself
Equivalence	NOT always can be rewritten as a `for`	Always can be rewritten as a `while`