**37 – For In e Ranges**

<https://docs.python.org/3/reference/compound_stmts.html#for>

**For in**

O loop for in, é mais usado em iterações;

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

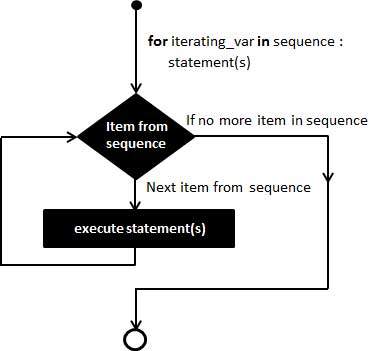
Syntax

for iterating\_var in sequence:

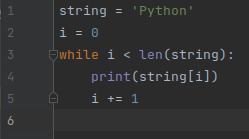
statements(s)

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.

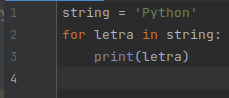
Flow Diagram



Exemplo: imprimir cada letra de uma string, vimos na aula passada:

Bom, sabemos que uma string é iterável, ou seja é uma sequência de elementos. Podemos então utilizar o for in para percorrer uma string:

O primeiro item (caracter) da string é atribuído a variável letra, então é impresso a letra, depois o próximo item (caracter) é atribuído a variável letra, então é impresso a letra, depois ... , depois o último item (caracter) da string é atribuído a variável letra, então é impresso a letra.

**for else**

Sintaxe:

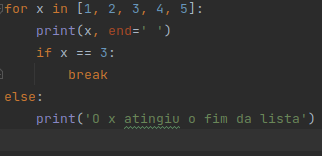
**for <var> in <iterable>:**

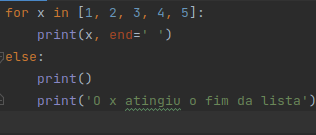
**<bloco do for>**

**else:**

**<bloco do else>**

O else só é exibido se o <var> atingir o ultimo item da sequência <iterable>

The Built-in function: **range()**

*class***range**(*stop*)

*class***range**(*start*, *stop*[, *step*])

Rather than being a function, [range](https://docs.python.org/3/library/stdtypes.html" \l "range" \o "range) is actually an immutable sequence type, as documented in [Ranges](https://docs.python.org/3/library/stdtypes.html" \l "typesseq-range) and [Sequence Types — list, tuple, range](https://docs.python.org/3/library/stdtypes.html" \l "typesseq).

Ou seja é função que gera o tipo de sequência imutável Range do python.

The immutable sequence type: **Range**

Ranges

The [range](https://docs.python.org/3/library/stdtypes.html" \l "range" \o "range) type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in [for](https://docs.python.org/3/reference/compound_stmts.html" \l "for) loops.

*class***range**(*stop*)

*class***range**(*start*, *stop*[, *step*])

The arguments to the range constructor must be integers (either built-in [int](https://docs.python.org/3/library/functions.html" \l "int" \o "int) or any object that implements the [\_\_index\_\_()](https://docs.python.org/3/reference/datamodel.html" \l "object.__index__" \o "object.__index__) special method). If the *step* argument is omitted, it defaults to 1. If the *start* argument is omitted, it defaults to 0. If *step* is zero, [ValueError](https://docs.python.org/3/library/exceptions.html" \l "ValueError" \o "ValueError) is raised.

For a positive *step*, the contents of a range r are determined by the formula r[i] = start + step\*i where i >= 0 and r[i] < stop.

For a negative *step*, the contents of the range are still determined by the formula r[i] = start + step\*i, but the constraints are i >= 0 and r[i] > stop.

A range object will be empty if r[0] does not meet the value constraint. Ranges do support negative indices, but these are interpreted as indexing from the end of the sequence determined by the positive indices.

Ranges containing absolute values larger than [sys.maxsize](https://docs.python.org/3/library/sys.html" \l "sys.maxsize" \o "sys.maxsize) are permitted but some features (such as [len()](https://docs.python.org/3/library/functions.html" \l "len" \o "len)) may raise [OverflowError](https://docs.python.org/3/library/exceptions.html" \l "OverflowError" \o "OverflowError).

Range examples:

>>>

**>>>** list(range(10))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**>>>** list(range(1, 11))

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**>>>** list(range(0, 30, 5))

[0, 5, 10, 15, 20, 25]

**>>>** list(range(0, 10, 3))

[0, 3, 6, 9]

**>>>** list(range(0, -10, -1))

[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

**>>>** list(range(0))

[]

**>>>** list(range(1, 0))

[]

Ranges implement all of the [common](https://docs.python.org/3/library/stdtypes.html" \l "typesseq-common) sequence operations except concatenation and repetition (due to the fact that range objects can only represent sequences that follow a strict pattern and repetition and concatenation will usually violate that pattern).

**start**

The value of the *start* parameter (or 0 if the parameter was not supplied)

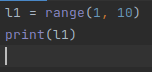
**stop**

The value of the *stop* parameter

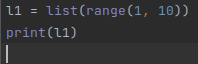
**step**

The value of the *step* parameter (or 1 if the parameter was not supplied)

The advantage of the [range](https://docs.python.org/3/library/stdtypes.html" \l "range" \o "range) type over a regular [list](https://docs.python.org/3/library/stdtypes.html" \l "list" \o "list) or [tuple](https://docs.python.org/3/library/stdtypes.html" \l "tuple" \o "tuple) is that a [range](https://docs.python.org/3/library/stdtypes.html" \l "range" \o "range) object will always take the same (small) amount of memory, no matter the size of the range it represents (as it only stores the start, stop and step values, calculating individual items and subranges as needed). Fontes: <https://docs.python.org/3/library/functions.html#func-range> <https://docs.python.org/3/library/stdtypes.html#range>

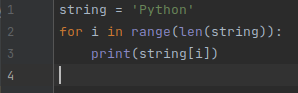
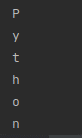
 

A função list() transforma o objeto range em uma lista (veremos na proxima aula):

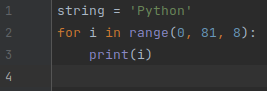
 

Como range() gera uma sequência, podemos usar ela com o for in, neste caso cada elemento da sequência é um número.

Logo aquele mesmo exemplo usando **for in** e **range()** ficaria:

Outro exemplo, múltiplos de 8:

 irá printar 0, 8, 16, ..., 64, 72, 80