

# Como definir uma função em Python

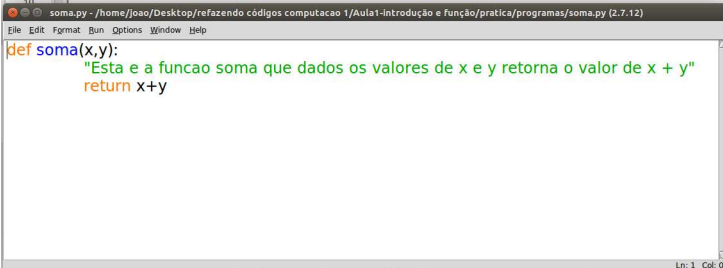
```
1 def nome_funcao(lista_parametros):  
2     """ Como se define uma funcao em Python """  
3     return valor de retorno
```

## Soma

- **Documentação / Comentário** : """ Esta e a funcao soma que dados os valores de x e y retorna o valor de x + y """
- **Nome da Função**: soma
- **Parâmetros**: x,y
- **Valor de Retorno**: : x+y

```
1 def soma(x, y):  
2     """ Esta e a funcao soma que dados os valores de x e y  
3     retorna o valor de x + y """  
4     return x+y
```

# Editor IDLE



The image shows a screenshot of the Python IDLE editor window. The title bar reads "soma.py - /home/joao/Desktop/refazendo códigos computacao 1/Aula1-Introdução e função/pratica/programas/soma.py (2.7.12)". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The main text area contains the following Python code:

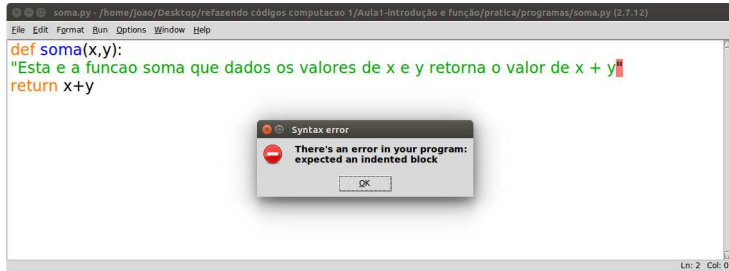
```
def soma(x,y):  
    "Esta e a funcao soma que dados os valores de x e y retorna o valor de x + y"  
    return x+y
```

The status bar at the bottom right indicates "Ln: 1 Col: 0".

**Execute sua função a partir do Editor IDLE:  
selecione **Run Module (F5)****

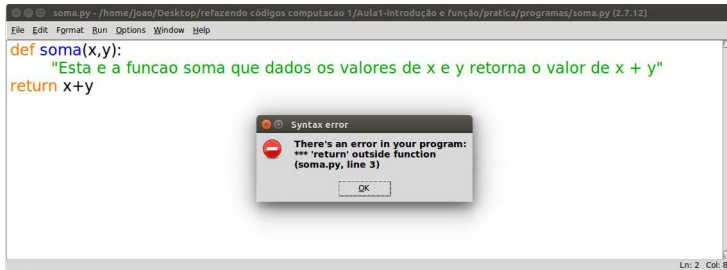
# Editor IDLE

A indentação é parte da sintaxe do Python. É através dela que podemos construir estruturas de código, como as funções.



# Editor IDLE

A indentação é parte da sintaxe do Python. É através dela que podemos construir estruturas de código, como as funções.



# Mensagens de Erro

Ao tentar interpretar o código que escrevemos, o Python avisa quando alguma coisa não foi compreendida através das mensagens de erro.

É importante ler as mensagens para saber onde estamos errando.

## Erros Frequentes

- **SyntaxError**: erros de sintaxe. Alguma palavra foi escrita incorretamente, ou algum símbolo foi esquecido (por exemplo, o `:` ao final da definição de uma função)
- **IndentationError**: Alguma linha teve sua indentação alterada manualmente (e erroneamente).

```
>>> def mult(x,y):
return x*y
File "<pyshell#9>", line 2
    return x*y
    ^
IndentationError: expected an indented block
```

# Mensagens de Erro

Ao tentar interpretar o código que escrevemos, o Python avisa quando alguma coisa não foi compreendida através das mensagens de erro.

É importante ler as mensagens para saber onde estamos errando.

## Erros Frequentes

- **NameError**: erro de nome. Algum nome foi usado sem ser anteriormente definido. Ocorre por exemplo ao chamar uma função que ainda não foi definida.

```
>>> def soma(x,y):  
    return x+y  
  
>>> Soma(3,4)  
  
Traceback (most recent call last):  
  File "<pyshell#3>", line 1, in <module>  
    Soma(3,4)  
NameError: name 'Soma' is not defined  
>>>
```

## Tipos Numéricos

Em computação, um tipo de dado é uma classificação dos dados. Essa classificação determina como os dados serão armazenados no computador e também permite a disponibilização de operações pré definidas na linguagem de programação.

- **Tipo inteiro (int)** : 10
- **Tipo ponto flutuante (float)**: 10.5 ,  $-190.00005$  ,  $15e - 5$
- **Tipo complexo (complex)** :  $(3 + 2j)$  ,  $(20j)$

# Tipos Numéricos

- **Números Inteiros:** **Int**

Os inteiros (int) têm precisão fixa ocupando tipicamente uma palavra de memória

Em PC's são tipicamente representados com 32 bits (de  $-2^{31}$  a  $2^{31} - 1$ )

- **Ponto Flutuante:** **Float**

Constantes têm que possuir um ponto decimal ou serem escritas em notação científica com a letra “e” (ou “E”) precedendo a potência de 10

10 int

10.0 float

- **Números Complexos:** **Complex**

Representados com dois números de ponto flutuante: um para a parte real e outro para a parte imaginária.

Constantes são escritas como uma soma sendo que a parte imaginária tem o sufixo *j* ou *J*

$(2 + 3j)$

$(7j)$

$(5 + 0j)$



# Primeiros Passos - Operadores

adição	+
subtração	-
multiplicação	*
divisão	/ ou //
exponenciação	**
módulo	%

## Regras de precedência

- 1 Expressões entre parênteses
- 2 Exponenciação
- 3 Multiplicação, Divisão e Módulo (\*)
- 4 Adição e Subtração (\*)

(\*) Esquerda para direita

## Tipos de Dados - Sequência de caracteres: str

- Constantes string são escritas usando aspas simples ou duplas

"a" ou 'a'

- O operador + pode ser usado para concatenar strings

"a"+"b" é o mesmo que "ab"

- O operador \* pode ser usado para repetir strings

"a"\*10 é o mesmo que "aaaaaaaaaa"

## Tipos de Dados - Booleano (bool)

- Assume apenas dois valores: verdadeiro (True) ou falso (False)
- É o tipo de dado resultante das operações de comparação.

```
1 >>> 3>2
2     True
3
4 >>> 10 <= 5
5     False
```

# Relações e Expressões Booleanas

- Operadores:  $>$  ,  $<$  ,  $==$  (igual),  $!=$  (diferente),  $>=$  ,  $<=$

## ATENÇÃO

- $X == Y$  : operador relacional  $\Rightarrow$  X É IGUAL A Y
- $X = Y$  : operador de atribuição  $\Rightarrow$  ATRIBUIR A X O VALOR DE Y

# Relações e Expressões Booleanas

- **Relações:**  $>$  ,  $<$  ,  $==$  (igual),  $!=$  (diferente),  $>=$  ,  $<=$
- **Operadores Lógicos:** not (negação), and (e), or (ou)
- **Expressões Booleanas:** Retornam como resultado de sua avaliação os valores verdadeiro (True) ou falso (False)

Exp 1	Exp 2	Exp 1 and Exp 2	Exp 1 or Exp 2	not Exp 1
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

**Exp1** e **Exp2** podem ser dados booleanas ou podem ser expressões booleanas compostas de operadores relacionais e operadores lógicos.

# Relações e Expressões Booleanas

**Ordem de Precedência:** do maior para o de menor precedência

- 1 **\*\***
- 2 **\*, /, //, %**
- 3 **+, -**
- 4 **<, <=, >, >=, !=, ==**
- 5 **not**
- 6 **and**
- 7 **or**

# Estrutura Condicional Composta



# Estrutura Condicional

Faça uma função que determina se um número inteiro  $X$  passado como parâmetro é positivo, negativo ou zero. O valor de retorno da função deve ser uma dentre as strings “ $X$  é positivo”, “ $X$  é negativo” ou “ $X$  é zero”.

```
1 def PosNegZero(X):  
2     """ Funcao ... """  
3  
4     if X > 0 :  
5         return str(X) + " e positivo"  
6     else:  
7         if X < 0 :  
8             return str(X) + " e negativo"  
9         else:  
10            return str(X) + " e zero"
```



# Estrutura Condicional

else: ... if  $\Leftrightarrow$  elif ...:

```
1 def PosNegZero(X):
2     """ Funcao ... """
3
4     if X > 0 :
5         return str(X) + " e positivo"
6     else:
7         if X < 0 :
8             return str(X) + " e negativo"
9         else:
10            return str(X) + " e zero"
```

```
1 def PosNegZero(X):
2     """ Funcao ... """
3
4     if X > 0 :
5         return str(X) + " e positivo"
6     elif X < 0 : # ESTA LINHA MUDOU !
7         return str(X) + " e negativo"
8     else:
9         return str(X) + " e zero"
```

# Variáveis e Atribuição

- **Atribuição:** O símbolo = é usado para atribuir um valor a uma variável.

*var = valor*

*var1, var2, ..., varN = valor1, valor2, ..., valorN*

```
...  
nome = "Carlos"  
return "Olá " + nome
```

**MEMÓRIA**

**nome**



"Carlos"

# Atribuindo Valores a Variáveis

## ATENÇÃO

Qual a diferença entre as funções abaixo ?

```
1 def teste1():  
2     a = 10  
3     a, b = 3 * a, a  
4     return a, b
```

```
1 def teste2():  
2     a = 10  
3     a = 3 * a  
4     b = a  
5     return a, b
```

# Atribuindo Valores a Variáveis

## ATENÇÃO

Qual a diferença entre as funções abaixo ?

```
1 def teste1():  
2     a = 10  
3     a, b = 3 * a, a  
4     return a, b
```

```
1 def teste2():  
2     a = 10  
3     a = 3 * a  
4     b = a  
5     return a, b
```

O lado direito da atribuição é sempre avaliado antes que a atribuição seja feita.

# Strings

- **Caracteres** são símbolos. Podem ser letras, números, caracteres especiais, e até o espaço em branco é um caractere.

Exemplo: 'a', '9', '#', ' '.

- Uma *string* é uma sequência de caracteres.

```
1 >>> a = 'abcd'
2 >>> b = "1234"
3 >>> c = "#$5a"
4 >>> d = ''
5 >>> e = ' '
```

- Comprimento de uma string: número de caracteres que ela contém.

```
1 >>> s = '123456'
2 >>> len(s)
3      6
```

# Strings - Índices

- Todo caractere de uma string é **indexado**, começando do primeiro caractere (**índice 0**) à esquerda.
- **Notação:** string[índice]

**Exemplo:** var = "Pedro dos Santos"

	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
<u>Índice</u>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Var	P	e	d	r	o		d	o	s		S	a	n	t	o	s

```
1 >>> var[2]
2     'd'
3 >>> var[9]
4     ''
5 >>> var[15]
6     's'
```

# Strings - Índices

- A string também pode ser indexada da direita para a esquerda, começando no **índice -1**.
- **Notação:** string[indice]

**Exemplo:** var = "Pedro dos Santos"

	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	
<u>Índice</u>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Var	P	e	d	r	o		d	o	s		S	a	n	t	o	s	

```
1 >>> var[-14]
2     'd'
3 >>> var[-7]
4     ' '
5 >>> var[-1]
6     's'
```

# Tuplas

- Uma **tupla** é uma sequência heterogênea (permite que seus elementos sejam de tipos diferentes):

```
1 >>> a = (1, 2, 3, 4)
2 >>> b = (1.0, 2, '3', 4+0j)
3 >>> c = 1, 2, 3, 4
4 >>> d = (1, )
```

- Valores em uma tupla podem ser distribuídos em variáveis como uma atribuição múltipla:

```
1 >>> x = 1, 2, 3
2 >>> x
3     (1, 2, 3)
4 >>> a, b, c = x
5 >>> a
6     1
7 >>> b
8     2
9 >>> c
10    3
```



# Tuplas

- **Tupla Vazia:** tupla sem elementos.
- **Tupla unitária:** contém um único elemento, que deve ser sucedido por uma vírgula.
- Os parênteses são opcionais se não provocarem ambiguidade.
- Um valor entre parênteses sem vírgula no final é meramente uma expressão.

**Qual o tipo de dado da variável A em cada um dos casos abaixo:**

```
1 >>> A = ()
2
3 >>> A = (10)
4
5 >>> A = 10,
6
7 >>> A = (10,)
8
9 >>> A = 3*(10+3)
10
11 >>> A = 3*(10+3,)
```

# Tuplas

- **Tupla Vazia:** tupla sem elementos.
- **Tupla unitária:** contém um único elemento, que deve ser sucedido por uma vírgula.
- Os parênteses são opcionais se não provocarem ambiguidade.
- Um valor entre parênteses sem vírgula no final é meramente uma expressão.

**Qual o tipo de dado da variável A em cada um dos casos abaixo:**

```
1 >>> A = ()
2     () # tupla vazia
3 >>> A = (10)
4     10 # inteiro
5 >>> A = 10,
6     (10,) # tupla unitária
7 >>> A = (10,)
8     (10,) # tupla unitária
9 >>> A = 3*(10+3)
10    39 # inteiro
11 >>> A = 3*(10+3,)
12    (13, 13, 13) # tupla
```

# Tuplas

- Tuplas são muito similares às strings em relação às operações.
- O tamanho de uma tupla é dado pela função `len`.

```
1 >>> x = (1, 2, 3)
2 >>> len(x)
3     3
```

- **Indexação:** começando do 0 à esquerda, ou de -1 à direita.

```
1 >>> x[0]
2     1
```

- **Fatiamento:** idêntico às strings.

```
1 >>> x[0:2]
2     (1, 2) # NOVA TUPLA
```

# Tuplas

- **Concatenação e Replicação**

```
1 >>> x*2
2     (1,2,3,1,2,3)
3 >>> x + (5,4)
4     (1,2,3,5,4)
```

- **Imutabilidade** : uma vez criada, uma tupla não pode ser alterada !

```
1 >>> x[0] = 9
2 Traceback (most recent call last):
3   File "<pyshell#2>", line 1, in <module>
4     x[0]=9
5 TypeError: 'tuple' object does not support item assignment
```

# Listas

- Tipo de dados mais versátil do Python.
- Uma lista é representada como uma sequência de valores entre colchetes e separados por vírgula.
- Os elementos de uma lista podem ser de tipos de dados diferentes.
- Listas são **mutáveis** !!!

## Exemplo

```
1 >>> lista1 = ['calculo', 'fisica', 'computacao']
2 >>> lista2 = ['notas', 5.4, 'aprovado']
3 >>> lista2[1] = 6
4 >>> lista2
5 ['notas', 6, 'aprovado']
```

# Estrutura de Repetição *while*

```
while <condição>:  
    <sequência de comandos>
```

- A <condição> é uma expressão ou dado do tipo booleano (**True** ou **False**), tal como os testes usados com o comando **IF**.
- Estrutura também conhecida como **laço de repetição** ou **“loop”**: o bloco de comandos é sequencialmente repetido tantas vezes quanto o teste da condição for verdadeiro.
- Somente quando a condição se torna falsa a próxima instrução após o bloco de comandos associado ao **while** é executada (fim do laço).

# Estrutura de Repetição *while*

```
while <condição>:  
    <sequência de comandos>
```

- Se a <condição> da estrutura while já for falsa desde o início, o bloco de <sequência de comandos> associado a ela nunca é executado.
- Deve haver algum processo dentro do bloco de <sequência de comandos> que torne a **condição** falsa e a repetição seja encerrada, ou um erro GRAVE ocorrerá: sua função ficará rodando para sempre!!

## Estrutura de Repetição *while*

Estrutura que permite a repetição de um conjunto de comandos. Até o momento vimos o *while*:

```
while <condição>:  
    <sequência de comandos>
```

- Com *while* podemos implementar qualquer algoritmo que envolva repetição.
- **DICA:** o *while* é mais recomendado quando não se sabe ao certo quantas vezes a repetição será feita, pois a **condição** é um teste booleano qualquer e não necessariamente uma contagem.



## Estrutura de Repetição *while*

Faça uma função que determina a soma de todos os números pares desde 100 até 200.

## Estrutura de Repetição *while*

Faça uma função que determina a soma de todos os números pares desde 100 até 200.

```
1 def somaPares():
2
3     """Funcao que calcula a soma dos numeros pares de 100 a 200
4     Parametro de entrada: nao tem
5     Valor de retorno: int"""
6
7     soma = 0 # variavel acumuladora
8     contador = 100 # o contador nao precisa começar de zero
9     while contador <= 200:
10         soma = soma + contador
11         contador = contador + 2 # o contador nao precisa ir de 1 em 1
12     return soma
```

## Estrutura de Repetição *while*

A função abaixo apresenta algum problema?

```
1 def exemplo3():
2
3 """ Parametro de entrada: nao tem
4 Valor de retorno: int """
5
6     x = 10
7     while x > 8:
8         x = x+ 2
9     return x
```

## Estrutura de Repetição *while*

A função abaixo apresenta algum problema?

```
1 def exemplo3():
2
3     """ Parametro de entrada: nao tem
4     Valor de retorno: int """
5
6     x = 10
7     while x > 8:
8         x = x+ 2
9     return x
```

- Sendo  $X$  igual a 10, o teste  $X > 8$  é inicialmente verdadeiro.
- Enquanto a condição for verdadeira, apenas o comando  $X = X + 2$  será executado. Porém incrementar a variável  $X$  não altera a validade da condição  $X > 8$ .
- Logo, a repetição segue **indefinidamente!** (Loop infinito)

# Estrutura de Repetição *while*

O que faz a seguinte função ?

```
1 def soma(numero):  
2  
3 """ Parametro de entrada: int  
4 Valor de retorno: int """  
5  
6 soma = 0  
7 contador = 0  
8 while contador < numero:  
9     soma = soma + contador  
10    contador = contador + 1  
11    return soma
```

## Estrutura de Repetição *while*

Faça uma função *somaPares* que recebe uma tupla de números inteiros e calcula a soma de todos os números pares que ocorrem nesta tupla.

Por exemplo, a chamada *somaPares*((3, 1, 2, 4, 6, 7, 2)) deve retornar 14 e

*somaPares*((1, 3, 5, 7)) deve retornar 0.

# Estrutura de Repetição *while*

Faça uma função *somaPares* que recebe uma tupla de números inteiros e calcula a soma de todos os números pares que ocorrem nesta tupla.

Por exemplo, a chamada *somaPares((3, 1, 2, 4, 6, 7, 2))* deve retornar 14 e *somaPares((1, 3, 5, 7))* deve retornar 0.

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     indice = 0
8     while indice < len(tupla):
9         if tupla[indice] % 2 == 0:
10            soma = soma + tupla[indice]
11            indice = indice + 1
12     return soma
```

Estamos usando a variável *indice* para percorrer a *tupla*.

É possível acessar os elementos na tupla sem precisarmos da variável *indice*!

# Estrutura de Repetição *for*

Faça uma função *somaPares* que recebe uma tupla de números inteiros e calcula a soma de todos os números pares que ocorrem nesta tupla.

Por exemplo, a chamada *somaPares*((3, 1, 2, 4, 6, 7, 2)) deve retornar 14 e *somaPares*((1, 3, 5, 7)) deve retornar 0.

Com o comando *for*, podemos pegar um a um os elementos que formam a tupla dada como entrada:

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```



# Estrutura de Repetição *for*

```
1 def somaPares(tupla):  
2  
3 """ Parametro de entrada: tupla  
4 Valor de retorno: int """  
5  
6 soma = 0  
7 for elemento in tupla:  
8     if elemento % 2 == 0:  
9         soma = soma + elemento  
10    return soma
```

● somaPares((10,21,32,43))

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):  
2  
3 """ Parametro de entrada: tupla  
4 Valor de retorno: int """  
5  
6 soma = 0  
7 for elemento in tupla:  
8     if elemento % 2 == 0:  
9         soma = soma + elemento  
10 return soma
```

- somaPares((10,21,32,43))
- soma = 0

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0:

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):  
2  
3 """ Parametro de entrada: tupla  
4 Valor de retorno: int """  
5  
6 soma = 0  
7 for elemento in tupla:  
8     if elemento % 2 == 0:  
9         soma = soma + elemento  
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0:

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0: (False)



# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- `somaPares((10,21,32,43))`
- `soma = 0`
- `for elemento in (10,21,32,43):`
  - `if 10 % 2 == 0: (True)`
    - `soma = 0 + 10 = 10`
  - `if 21 % 2 == 0: (False)`
  - `if 32 % 2 == 0:`

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0: (False)
  - if 32 % 2 == 0: (True)

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0: (False)
  - if 32 % 2 == 0: (True)
    - soma = 10 + 32 = 42

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0: (False)
  - if 32 % 2 == 0: (True)
    - soma = 10 + 32 = 42
  - if 43 % 2 == 0:

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- somaPares((10,21,32,43))
- soma = 0
- for elemento in (10,21,32,43):
  - if 10 % 2 == 0: (True)
    - soma = 0 + 10 = 10
  - if 21 % 2 == 0: (False)
  - if 32 % 2 == 0: (True)
    - soma = 10 + 32 = 42
  - if 43 % 2 == 0: (False)

# Estrutura de Repetição *for*

```
1 def somaPares(tupla):
2
3     """ Parametro de entrada: tupla
4     Valor de retorno: int """
5
6     soma = 0
7     for elemento in tupla:
8         if elemento % 2 == 0:
9             soma = soma + elemento
10    return soma
```

- `somaPares((10,21,32,43))`
- `soma = 0`
- `for elemento in (10,21,32,43):`
  - `if 10 % 2 == 0: (True)`
    - `soma = 0 + 10 = 10`
  - `if 21 % 2 == 0: (False)`
  - `if 32 % 2 == 0: (True)`
    - `soma = 10 + 32 = 42`
  - `if 43 % 2 == 0: (False)`
- `return 42`

# Estrutura de Repetição *for*

- A função `range(...)` pode ter 1, 2 ou 3 argumentos:
  - `range(numero)`: faz com que a variável do `for` assuma valores de 0 a `numero-1`

`for x in range(10):` → x recebe 0,1,2,...,9

- `range(inf,sup)`: faz com que a variável do `for` assuma valores de `inf` a `sup-1`

`for x in range(3,8):` → x recebe 3,4,5,6,7

- `range(inc, sup, inc)`: faz com que a variável do `for` assuma valores de `inf` a `sup-1` com incremento de `inc`

`for x in range(3,8,2):` → x recebe 3,5,7

## Estrutura de Repetição *for*

Faça uma função que determina a soma de todos os números pares desde 100 até 200. (Usando **for** ao invés de **while**)



# Estrutura de Repetiç o *for*

Faça uma funç o que determina a soma de todos os n meros pares desde 100 at  200. (Usando **for** ao inv s de **while**)

```
1 def somaPares():
2
3     """Funcao que soma todos os numeros pares de 100 ate 200
4     Parametro de entrada: nao tem
5     Valor de retorno: int"""
6
7     soma = 0
8     for par in range(100,202,2):
9         soma = soma + par
10    return soma
```

OU

```
1 def somaPares():
2
3     """Funcao que soma todos os numeros pares de 100 ate 200
4     Parametro de entrada: nao tem
5     Valor de retorno: int"""
6
7     soma = 0
8     lista = range(100,202,2)
9     for par in lista:
10        soma = soma + par
11    return soma
```

# Estrutura de Repetição

**IMPORTANTE:** diferença de uso entre *while* e *for*:

- **while:** decisão sobre repetir ou não baseia-se em teste booleano.  
Risco de loop infinito. :-)
- **for:** Contagem automática do número de repetições.