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
```

Função

Argumentos Default: Permitem que valores default sejam utilizados quando nenhum valor é especificado em um certo parâmetro.

Formato

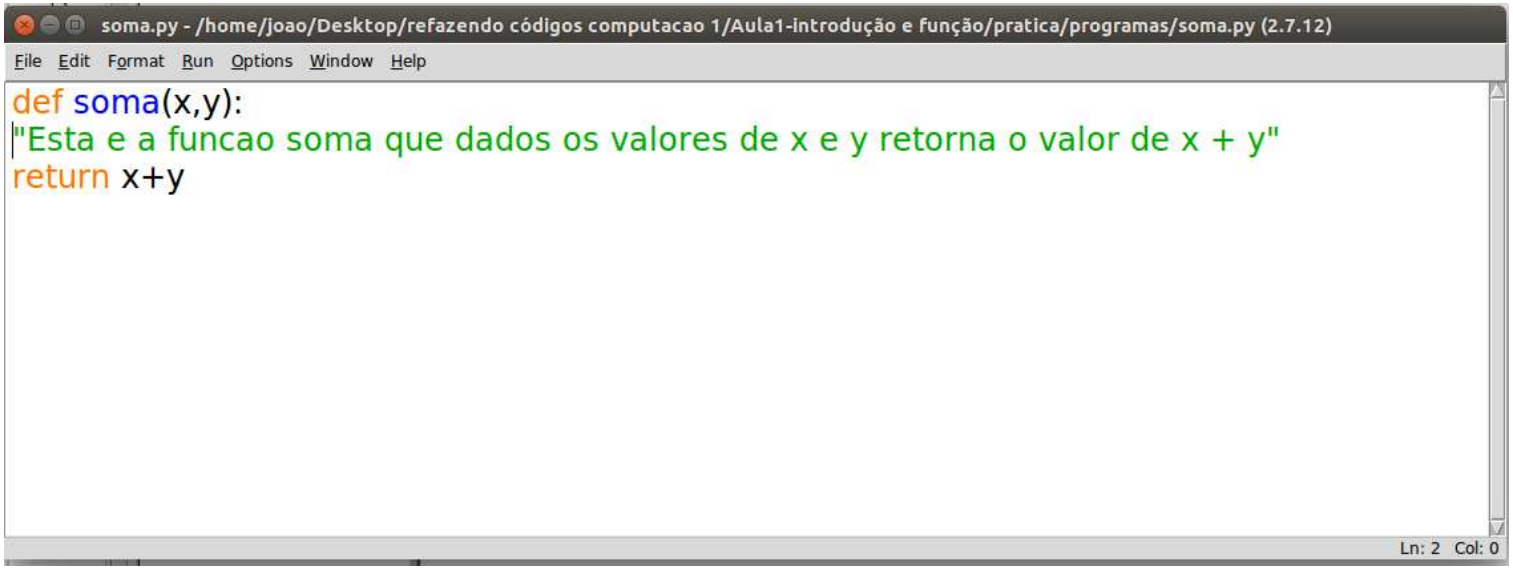
$$\text{def nome-funcao}(arg_0, \cdots, arg_N, arg_{N+1} = default_1, \cdots, arg_M = default_M)$$

$$\cdots$$

- $arg_0, \cdots, arg_N$: *Argumentos sem valores default.*
- $arg_{N+1} = default_1, \cdots, arg_M = default_M$: *Argumentos com valores default. Devem ser sempre os últimos argumentos.*

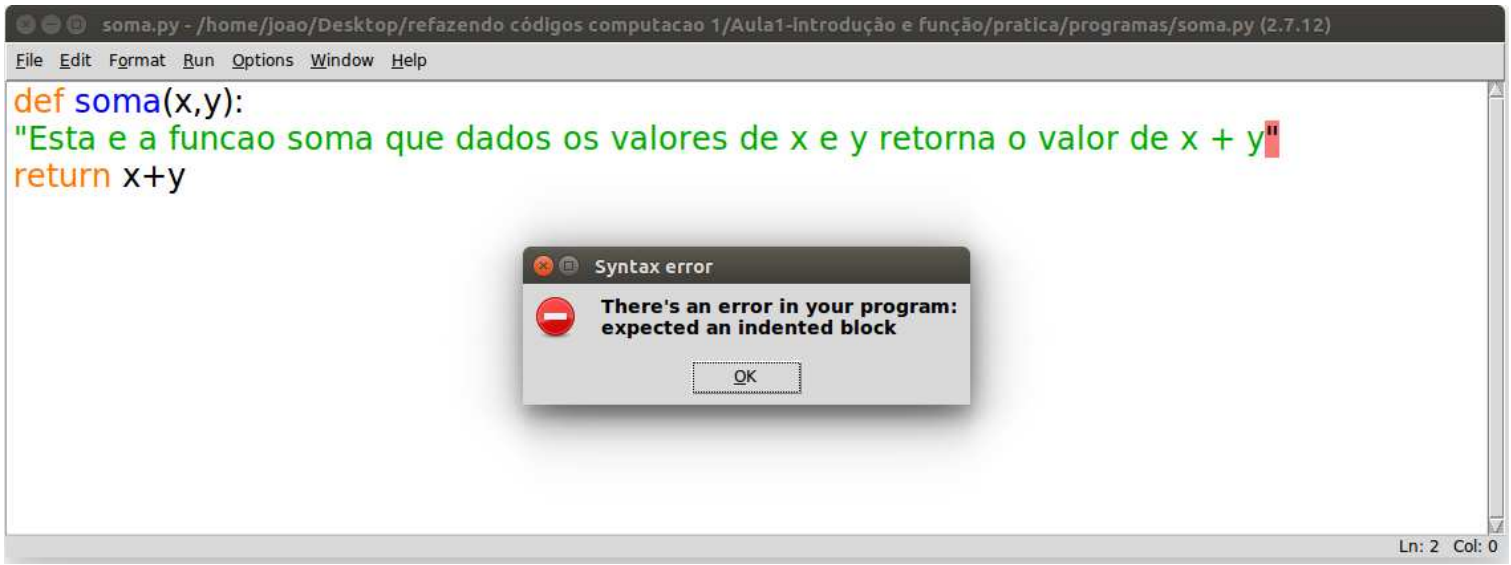
Editor IDLE

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



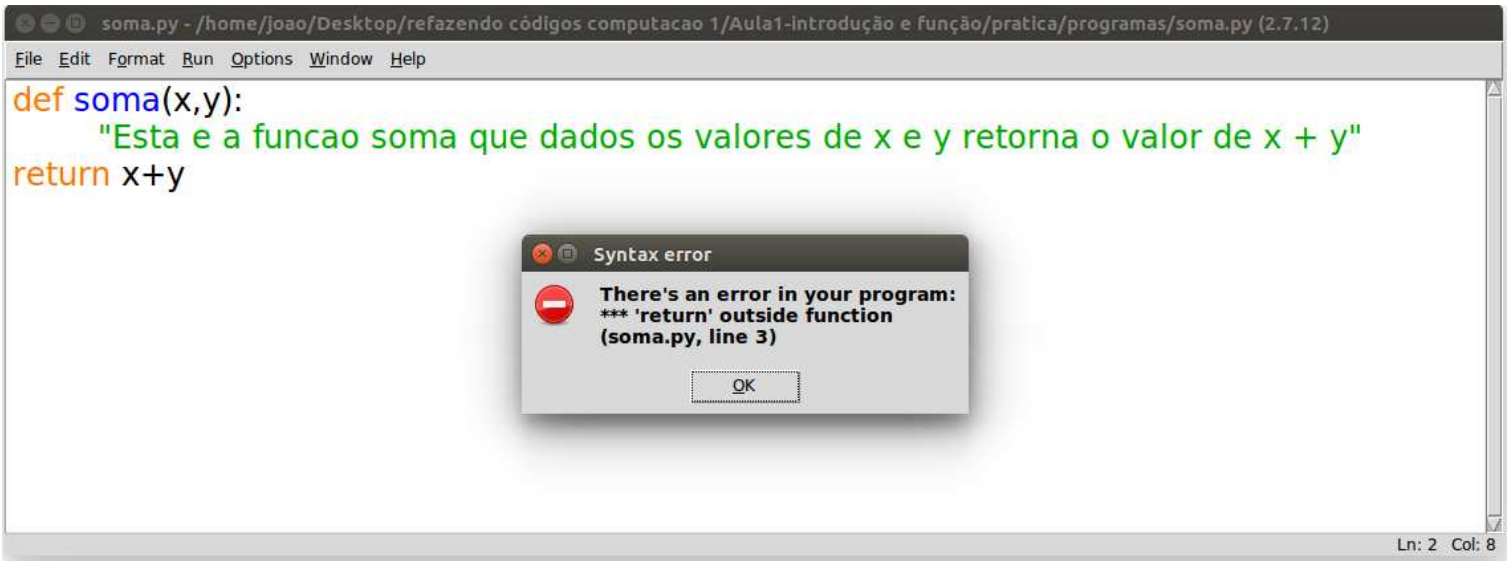
Editor IDLE

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



Editor IDLE

A identação 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 identação 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 náo 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
>>>
```

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 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

  $(2+3j)$ $(7j)$ $(5+0j)$

Tipos de Dados - Booleano (bool)

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

```
>>> 3>2
    True

>>> 10 <= 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**

Tipos de Dados - Sequência de caracteres: str

- Constantes string são escritas usando aspas simples ou duplas

  "a" ou 'a'

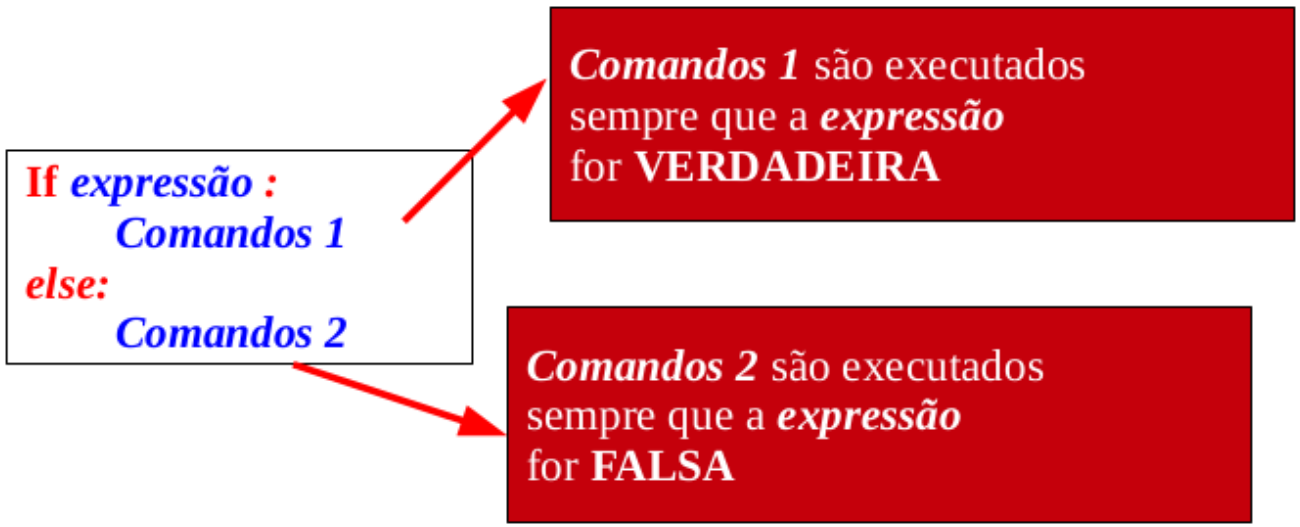
- O operador + pode ser usado para concatenar strings

  "a"+"b" é o mesmo que "ab"

- O operador * pode ser usado para repetir strings

  "a"*10 é o mesmo que "aaaaaaaaaa"

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 ⇔ 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"
```

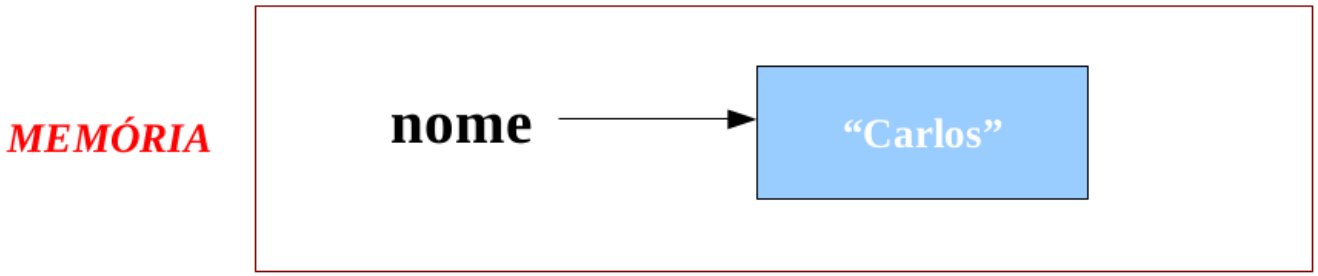
Variáveis e Atribuição

- **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
```



Atribuindo Valores a Variáveis

Uma variável é criada com um comando de atribuição: *variavel* = *valor*

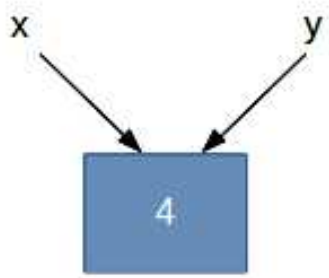
```
1 >>> x = 4
```

Um alias é um identificador que se refere a uma variável existente. É criado com uma atribuição *variavel* = *outra_variavel* já existente

```
1 >>> y = x
```

A variável *y* é um alias para a variável *x*. Portanto, *y* possui o mesmo valor e aponta para a mesma posição na memória que *x*.

```
1 >>> y
2     4
```



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
```

Variáveis – Tipo

- Python é uma linguagem dinamicamente tipada ou fracamente tipada.
- O tipo é atribuído de acordo com o valor atribuído à variável. Não é necessário *declarar previamente* o tipo.

```
>>> x = 4
>>> type(x)
    <class 'int'>
```

- O tipo de uma variável pode mudar depois de alguma operação ou nova atribuição.

```
>>> x = complex(x)
>>> type(x)
    <class 'complex'>
```

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.

```
>>> a = 'abcd'
>>> b = "1234"
>>> c = "#$5a"
>>> d = ''
>>> e = '  '
```

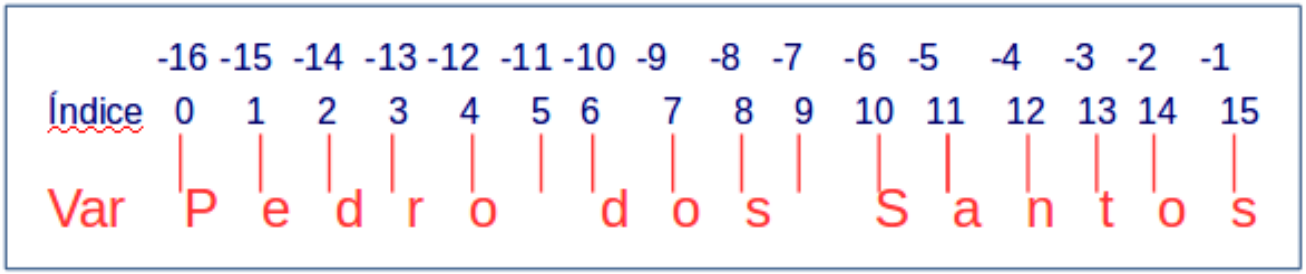
- Comprimento de uma string: número de caracteres que ela contém.

```
>>> s = '123456'
>>> len(s)
    6
```

Strings - Índices

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

Exemplo: var = "Pedro dos Santos"




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

Strings - Fatiamento

- Separa trechos de uma string.
- **Notação**: string[índice1:índice2]
  - Retorna os caracteres desde o de índice1 até o imediatamente anterior ao índice2
  - Se índice1 é omitido, é assumido 0.
  - Se índice2 é omitido, é assumido o fim da string.

Strings - Fatiamento

Incremento: podemos usar incremento / decremento para selecionar os elementos de uma string.

[start:end:step]: vai do índice *start* até *end* (sem ultrapassá-lo, com passo *step*)

Exemplo

```
>>> x= "abcde"
>>> x[0:-1:2]
    'ac'
>>> x[3:0:-1]
    'dcb'
```

Strings

- Elementos de uma string não aceitam o operador de atribuição.

```
>>> s = '123456'
>>> s[0] = '0'
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[0]='0'
TypeError: 'str' object does not support item assignment
```

- Strings são, portanto, imutáveis. Ou seja, os dados contidos em uma string não podem ser alterados.

Tuplas

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

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

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

```
>>> x = 1, 2, 3
>>> x
    (1, 2, 3)
>>> a, b, c = x
>>> a
    1
>>> b
    2
>>> c
    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:

```
>>> A = ()
    () # tupla vazia
>>> A = (10)
    10 # inteiro
>>> A = 10,
    (10,) # tupla unitaria
>>> A = (10,)
    (10,) # tupla unitaria
>>> A = 3*(10+3)
    39 # inteiro
>>> A = 3*(10+3,)
    (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.

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

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

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

- **Fatiamento**: idêntico às strings.

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

Tuplas

- **Concatenação e Replicação**

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

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

```
>>> x[0] = 9
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0]=9
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

```
>>> lista1 = ['calculo', 'fisica', 'computacao']
>>> lista2 = ['notas', 5.4, 'aprovado']
>>> lista2[1] = 6
>>> lista2
    ['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*

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*

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  """Paramentro 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
```

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  """Paramentro 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 varíavel *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  """Paramentro 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*

- 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(inf, 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 Repetição *for*

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

Estrutura de Repetição *for*

Faça uma função que dada uma palavra retorna uma lista formada por todas as vogais que aparecem na palavra.

Estrutura de Repetição *for*

Faça uma função que dada uma palavra retorna uma lista formada por todas as vogais que aparecem na palavra.

```
1  def vogaisPalavra(palavra):
2
3  """Funcao que retorna todas as vogais que aparecem em uma palavra
4  Parametro de entrada: str
5  Valor de retorno: list"""
6
7    vogais = ['a','e','i','o','u','A','E','I','O','U']
8    resposta = [ ]
9    for letra in palavra:
10     if letra in vogais:
11       list.append(resposta,letra)
12   return resposta
```