Fundamentos da Programação

Tuplos

Aula 7

José Monteiro

(slides adaptados do Prof. Alberto Abad)

Tuplos

- Um tuplo é uma sequência *imutável* de elementos.
- Cada elemento pode ser referenciado através do seu índice ou posição.
- Representação externa de um tuplo em Python (BNF):

`<tuplo> ::= () | (<elemento>, <elementos>)`

`<elementos> ::= <nada> | <elemento> | <elemento>, <elementos>`

`<elemento> ::= <expressão> | <tuplo> | <lista> | <dicionário>`

`<nada> ::=`

Exemplos de Tuplos

```
>>> a = ()
>>> type(a)
...
>>> a = (2)
>>> type(a)
...
>>> a = (2,)
>>> type(a)
...
>>> a = (2,4,5)
>>> type(a)
...
>>> a = (2,4,5,)
>>> type(a)
...
>>> a = (2,4,5,'ola')
>>> type(a)
...
>>> a = (2,4,5,'ola',(8,9,))
>>> type(a)
...
>>> a = (2,4,(False,5),True,(8,9,))
>>> type(a)
...
```

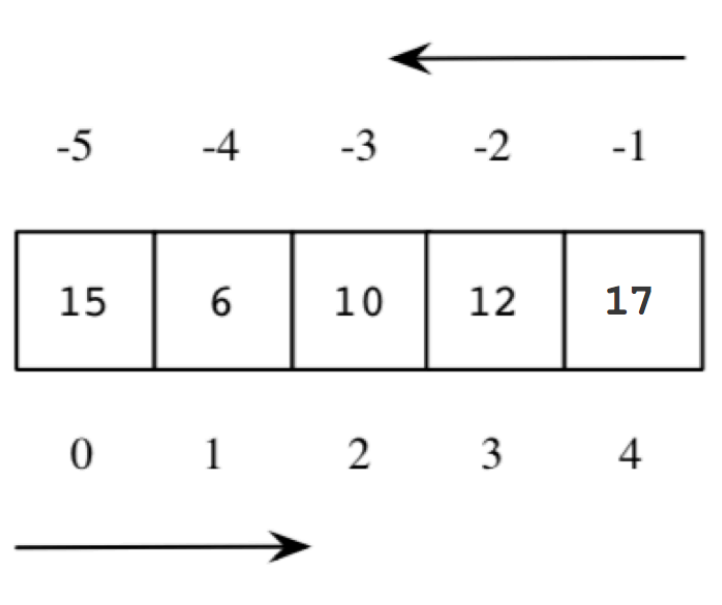
In [ ]:

Aceder a Elementos de um Tuplo

- Sintaxe BNF:

`<nome indexado> ::= <nome>[<expressão>]`

- Índices (inteiros):



Exemplos de Indexação de Tuplos

```
>>> notas = (15, 6, 10, 12, 17)
>>> notas[0]
15
>>> notas[2]
10
>>> notas[-1]
17
>>> notas[-2]
12
>>> notas[3+1]
17
>>> i = 5
>>> notas[i-4]
6
```

In [ ]:

Exemplos de Indexação de Tuplos

```
>>> notas[9]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
>>> v = (12, 10, (15, 11, 14), 18, 17)
>>> v[2]
(15, 11, 14)
>>> v[2][1]
11
>>> v[2] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

In [ ]:

Operações sobre tuplos

Operação	*Tipo dos argumentos*	*Valor*
t_1 `+` t_2	Tuplos	A concatenação dos tuplos t_1 e t_2.
t `*` i	Tuplo e inteiro	A repetição i vezes do tuplo t.
t`[`i_1`:`i_2`]`	Tuplo e inteiros	O sub-tuplo de t entre os índices i_1 e $i_2 - 1$.
e `in` t	Universal e tuplo	`True` se o elemento e pertence ao tuplo t; `False` em caso contrário.
e `not in` t	Universal e tuplo	A negação do resultado da operação e `in` t.
`tuple(`a`)`	Lista ou dicionário ou cadeia de carateres	Transforma o seu argumento num tuplo. Se não forem fornecidos argumentos, devolve o tuplo vazio.
`len(`t`)`	Tuplo	O número de elementos do tuplo t.

Operações sobre Tuplos: Concatenação

```
>>> a = (2, 1, 3, 7, 5)
>>> b = (8, 2, 4, 7)
>>> a + b
(2, 1, 3, 7, 5, 8, 2, 4, 7)
>>> c = a + b
>>> a * b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'tuple'
>>> a * 2
(2, 1, 3, 7, 5, 2, 1, 3, 7, 5)
```

- Note-se a sobrecarga dos operadores $+$ e $*$
- Que acontece com `a + (2)` ?

In [ ]:

Operações sobre Tuplos: _Slicing_

- Seleção dos elementos de um tuplo (sub-tuplo) desde a posição inicial (*inclusive*) até posição final (*exclusive*) com passos ou incrementos fixos:

`vetor[inicio:fim:incr] ==> (vetor[inicio], vetor[inicio+1*incr], ..., vetor[inicio+i*incr])`

```
>>> a = (2, 1, 3, 7, 5)
>>> a[2:4]
(3, 7)
>>> a[:3]
(2, 1, 3)
>>> a[4:]
(5,)
>>> a[:]
(2, 1, 3, 7, 5)
>>> a[::2]
(2, 3, 5)
>>> a[-1::-1]
(5, 7, 3, 1, 2)
```

In [ ]:

Operações sobre Tuplos: _in, not in, len, tuple_

```
>>> a = (2, 1, 3, 7, 5)
>>> b = (8, 2, 4, 7)
>>> 1 in a
True
>>> 1 in b
False
>>> 'b' not in b
True
>>> len(a)
5
>>> tuple('hello world')
('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
>>> tuple(8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

In [ ]:

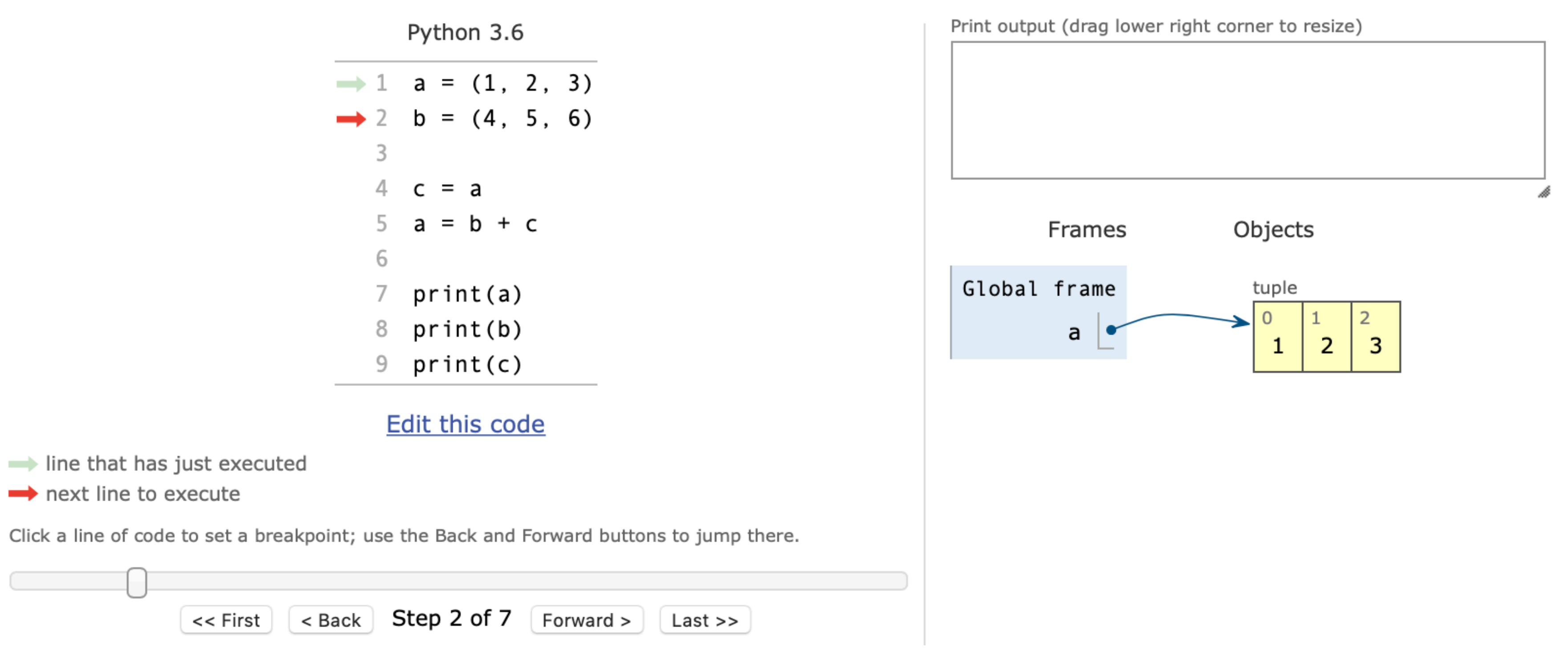
Sobre a Imutabilidade dos Tuplos

```
>>> a = (3, 4, 5, 6)
>>> b = (7, 8)
>>> a = a + b
>>> a
(3, 4, 5, 6, 7, 8)
```

- O que esta a acontecer? Os tuplos não são imutáveis!?
- Um tuplo ser *imutável* significa que:
  - Não podemos alterar um valor de um elemento de um tuplo.
  - Podemos criar tuplos (com mesmo nome) a partir de outros tuplos.
  - Para efectuarmos transformações sobre tuplos temos de aplicar as operações acima e construir novos tuplos.

In [ ]:

Sobre a Imutabilidade dos Tuplos



```
a = (1, 2, 3)
b = (4, 5, 6)

c = a
a = b + c

print(a)
print(b)
print(c)
```

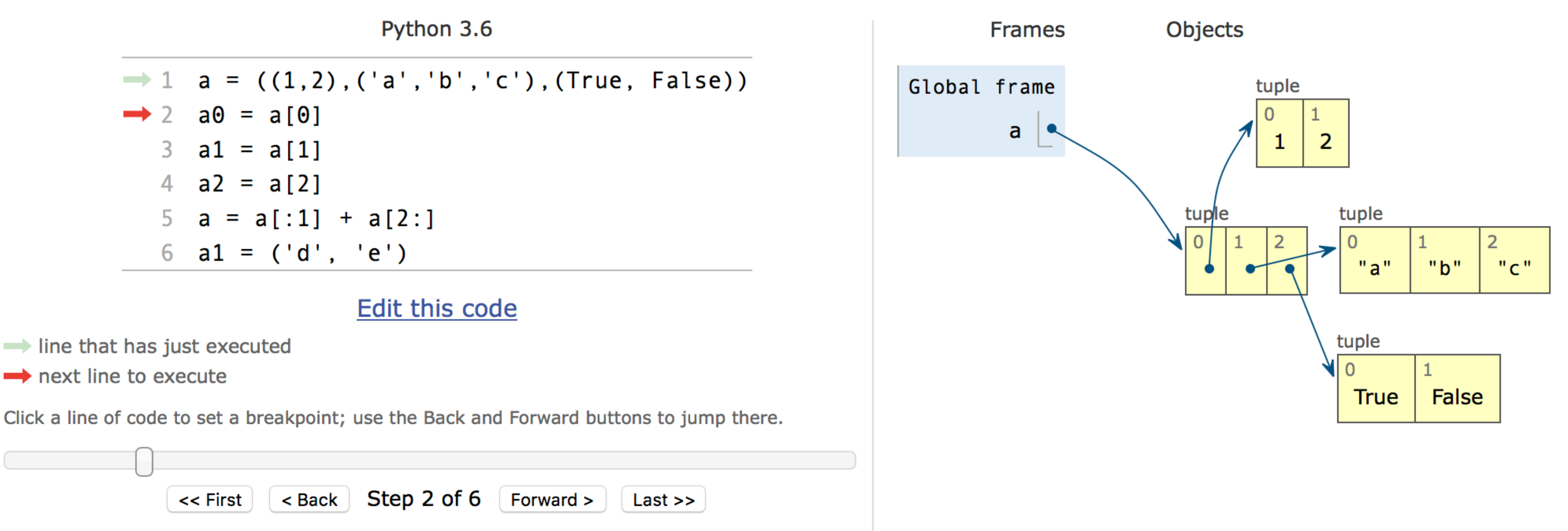
line that has just executed

next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

Step 2 of 7

Sobre a Imutabilidade dos Tuplos



```
a = ((1,2),('a','b','c'),(True, False))
a0 = a[0]
a1 = a[1]
a2 = a[2]
a = a[:1] + a[2:]
a1 = ('d', 'e')
```

line that has just executed

next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

Step 2 of 6

Tuplos: Exercício 1

Substitui Elemento

```
def substitui(tuplo, pos, elemento):
    pass
```

- Levanta `IndexError` se `pos` esta fora dos limites do `tuplo`

Exemplos:

```
>>> a = (2, 1, 3, 3, 5)
>>> substitui(a, 2, 'a')
(2, 1, 'a', 3, 5)
>>> substitui(a, 4, 'a')
(2, 1, 3, 3, 'a')
>>> a = substitui(a, 0, 'a')
>>> a = substitui(a, 5, 'a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in substitui
IndexError: substitui: no tuplo dado como primeiro argumento
>>> a
('a', 1, 3, 3, 5)
```

Tuplos: Exercício 1

Substitui Elemento

```
def substitui(tuplo, pos, elemento):
    pass
```

- Levanta `IndexError` se `pos` esta fora dos limites do `tuplo`

Exemplos:

```
>>> a = (2, 1, 3, 3, 5)
>>> substitui(a, 2, 'a')
(2, 1, 'a', 3, 5)
>>> substitui(a, 4, 'a')
(2, 1, 3, 3, 'a')
>>> a = substitui(a, 0, 'a')

>>> a = substitui(a, 5, 'a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in substitui
IndexError: substitui: no tuplo dado como primeiro argumento
>>> a
('a', 1, 3, 3, 5)
```

In [ ]:

```
def substitui(t, p, e):
    if p >= len(t) or p < 0:
        raise IndexError('indice fora dos limites')
    return t[:p] + (e,) + t[p+1:]

a = (1, 2, 3, 4, 5)
a = substitui(a, -1, True)
print(a)
```

Tuplos: Exercício 2

Calcula Soma dos Elementos do Tuplo

```
def soma_elementos(t):
    pass # completar!
    while i < len(t):
        pass # completar!
    pass # completar!
```

- Q0: Completar código: soma acumulada dos elementos dum vector de inteiros
- Q1: Como optimizar a condição?
- Q2: Alterar para obter tuplo de quadrados
- Q3: Como verificar tipos (vetor de inteiros)?

Tuplos: Exercício 2

Calcula Soma dos Elementos do Tuplo

```
def soma_elementos(t):
    pass # completar!
    while i < len(t):
        pass # completar!
    pass # completar!
```

- Q0: Completar código: soma acumulada dos elementos dum vector de inteiros
- Q1: Como optimizar a condição?
- Q2: Alterar para obter tuplo de quadrados
- Q3: Como verificar tipos (vetor de inteiros)?

In [ ]:

```
def soma_elementos(t):
    soma = 0
    i = 0
    while i < len(t):
        soma = soma + t[i]
        i = i + 1
    return soma

print(soma_elementos(((1,2,3))))

#question1: como optimizar a condição? --> tamanho = len(t) antes da condição
#question2: alterações para obter tuplo de quadrados
# --> res = (), dentro do loop res = res + (t[i]*t[i],))
#question3: como verificar tipos? --> dentro do loop, if type(t[i]) != int lançar error
```

Tuplos: Exercício 3

Calcula Soma Vetorial de 2 Tuplos

- Função que devolve vetor soma
- Garantir compatibilidade (mesmo tamanho)

Tuplos: Exercício 3

Calcula Soma Vetorial de 2 Tuplos

- Função que devolve vetor soma
- Garantir compatibilidade (mesmo tamanho)

In [ ]:

```
def soma_vectores(v1, v2):
    if len(v1) != len(v2):
        raise ValueError('tamanho dos vetores é incompatível')
    res = ()
    i = 0
    while i < len(v1):
        res = res + (v1[i] + v2[i],)
        i = i + 1
    return res

print(soma_vectores((1,2,3),(0,7,2)))
```

Tuplos: Exercício 4

Função *Alisa*

```
>>> alisa((2, 4, (8, (9, (7, ), 3, 4), 7), 6, (5, (7, (8, )))))
(2, 4, 8, 9, 7, 3, 4, 7, 6, 5, 7, 8)
```

t	i	t[:i]	t[i]	t[i+1:]
((1, 2), 3, (4, (5)))	0	()	(1, 2)	(3, (4, 5))
(1, 2, 3, (4, 5))	1			
(1, 2, 3, (4, 5))	2			
(1, 2, 3, (4, 5))	3	(1, 2, 3)	(4, 5)	()
(1, 2, 3, 4, 5)	4			

Tuplos: Exercício 4

Função *Alisa*

- Para escrever a função `alisa`, iremos utilizar a função embedida `isinstance`, em BNF:

` ::= isinstance(,)

::= | `

- Alternativa a `type` que retorrna `True` ou `False` is isinstance:

```
>>> isinstance(3, int)
True
>>> isinstance(3, (int, bool))
True
>>> isinstance(True, (int, bool))
True
>>> isinstance(5.6, (int, bool))
False
>>> isinstance('a', (int, bool))
False
>>> isinstance('a', (int, bool, str))
True
>>> isinstance((8,), tuple)
True
```

In [ ]:

Tuplos: Exercício 4

Função *Alisa*: proposta de solução

In [ ]:

```
def alisa(t):
    i = 0
    while i < len(t):
        if isinstance(t[i], tuple):
            t = t[:i] + t[i] + t[i+1:]
        else:
            i = i + 1

    return t

alisa((2, 4, (8, (9, (7, ), 3, 4), 7), 6, (5, (7, (8, )))))
```

Tarefas para a próxima aula

- Trabalhar matéria apresentada hoje
- Tentar fazer os Exercícios propostos ou não acabados