

# Introdução à Física Computacional I (4300218)

Prof. André Vieira  
[apvieira@if.usp.br](mailto:apvieira@if.usp.br)  
Sala 3120 – Edifício Principal

## Aula 2

Programação em Python para físicos:  
controle de fluxo, listas e arrays

# Controle de fluxo: `if`

- Caso deseje que algum conjunto de comandos seja executado apenas se alguma condição for satisfeita, utilize a instrução `if`.

```
x = int(input("Digite um inteiro menor ou igual a 10:"))  
  
if x>10:  
    print("Valor maior do que 10. Corrigindo...")  
    x = 10  
  
print("Seu número é",x)
```

```
===== RESTART: /private/tmp/temp.py =====  
Digite um inteiro menor ou igual a 10: 5  
Seu número é 5  
>>>  
  
===== RESTART: /private/tmp/temp.py =====  
Digite um inteiro menor ou igual a 10: 15  
Valor maior do que 10. Corrigindo...  
Seu número é 10  
>>> |
```

# Controle de fluxo: `if`

- À instrução `if` deve se seguir `<condição>`:

```
if x>10:
```

- A linha imediatamente abaixo deve ser “indentada” (avançada). Qualquer número de espaços pode ser utilizado, desde que consistentemente.

```
    print("Valor maior do que 10. Corrigindo...")
```

```
    x = 10
```

- Desfaça o avanço para encerrar a lista de comandos vinculada à condição.

```
print("Seu número é",x)
```

# Controle de fluxo: `if`

- Condições mais comumente utilizadas

`if x == 1`: verifica se  $x = 1$  (note a duplicação “`==`”)

`if x > 1`: verifica se  $x > 1$

`if x >= 1`: verifica se  $x \geq 1$

`if x < 1`: verifica se  $x < 1$

`if x <= 1`: verifica se  $x \leq 1$

`if x != 1`: verifica se  $x \neq 1$

# Controle de fluxo: `if`

- Combinação de condições: “e”, “ou”

```
if x >= 1 and x <= 10:
```

```
    print("O valor do número é adequado")
```

```
if x < 1 or x > 10:
```

```
    print("Número fora do intervalo.")
```

- Combinação de condições: “senão”

```
if x >= 1 and x <= 10:
```

```
    print("O valor do número é adequado")
```

```
else:
```

```
    print("Número fora do intervalo.")
```

# Controle de fluxo: `while`

- Uma variante sofisticada da instrução `if` é um laço `while` (“enquanto”):

```
x = int(input("Digite um inteiro menor que 11: "))
while x > 10:
    print("O número deve ser menor que 11.")
    x = int(input("Digite um inteiro menor que 11: "))
print("Seu número é", x)
```

```
===== RESTART: /private/tmp/temp.py =====
Digite um inteiro menor que 11: 15
O número deve ser menor que 11.
Digite um inteiro menor que 11: 13
O número deve ser menor que 11.
Digite um inteiro menor que 11: 10
Seu número é 10
>>> |
```

# Controle de fluxo: `while`

- O funcionamento de um laço `while` pode ser alterado pelas instruções `break` e `continue`.
- Exemplo 1:

```
temp.py - /private/tmp/temp.py (3.7.4)
x = 0
while x < 3:
    x += 1
    print("O valor de x é",x)
    if x == 2:
        break
    print("Cheguei à última linha do laço")
print("Fim")

===== RESTART: /private/tmp/temp.py =====
0 valor de x é 1
Cheguei à última linha do laço
0 valor de x é 2
Fim
>>> |
```

# Controle de fluxo: `while`

- O funcionamento de um laço `while` pode ser alterado pelas instruções `break` e `continue`.
- Exemplo 2:

```
temp.py - /private/tmp/temp.py (3.7.4)
x = 0
while x < 3:
    x += 1
    print("O valor de x é",x)
    if x == 2:
        continue
    print("Cheguei à última linha do laço")
print("Fim")

----- RESTART: /private/tmp/temp.py -----
O valor de x é 1
Cheguei à última linha do laço
O valor de x é 2
O valor de x é 3
Cheguei à última linha do laço
Fim
>>>
```

# Controle de fluxo: `while`

- Exemplo 3: os números de Fibonacci

Os números de Fibonacci são uma sequência de inteiros em que cada número é igual à soma dos dois números anteriores, com os dois primeiros números ambos iguais a 1. O código abaixo gera os números de Fibonacci menores ou iguais a 1000.



```
*temp.py - /private/tmp/temp.py (3.7.4)*
f1, f2 = 1, 1
while f1 <= 1000:
    print(f1)
    f1, f2 = f2, f1+f2
```

-----  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55  
89  
144  
233  
377  
610  
987

# Exercício 1

Os números de Catalan são uma sequência de números inteiros 1, 1, 2, 5, 14, 42, 132, ... relevantes em mecânica quântica e na teoria de sistemas desordenados. Eles são dados por

$$C_0 = 1, \quad C_{n+1} = \frac{4n+2}{n+2} C_n.$$

Escreva um programa que calcule e imprima em ordem crescente todos os número de Catalan menores ou iguais a um bilhão.

# Exercício 1: solução

```
teste.py - /tmp/teste.py (3.6.8)
File Edit Format Run Options Window Help
n, c = 0, 1
while c <= 10**9:
    print(c)
    c = (4*n+2)*c//(n+2)
    n += 1
```

```
=====
1
1
2
5
14
42
132
429
1430
4862
16796
58786
208012
742900
2674440
9694845
35357670
129644790
477638700
>>> |
```

Por que utilizar a divisão inteira?

Por que fazer primeiro a multiplicação por  $c$  e não a divisão inteira?

# “Recipientes”: listas

- Python permite representar entidades mais complicadas do que variáveis isoladas. através de objetos chamados *listas* (e suas extensões).

```
[ "Newton", "Curie", "Einstein" ]
```

```
[ -3, 1, 4, 1, 5, 9 ]
```

```
[ -3, 1.0, 4+0j, "um" ]
```

```
r = [ 1, 1, 2, 3, 5, 8, 13, 21 ]
```

# “Recipientes”: listas

- Elementos de uma lista podem ser variáveis:

```
x, y, z = 1, 1, 2
```

```
r = [ x, y, z ]
```

- Nesse caso, note que alterar o valor de uma das variáveis após a definição da lista **não** afeta a própria lista. Continuando o exemplo:

```
print(r)
```

```
x = 0
```

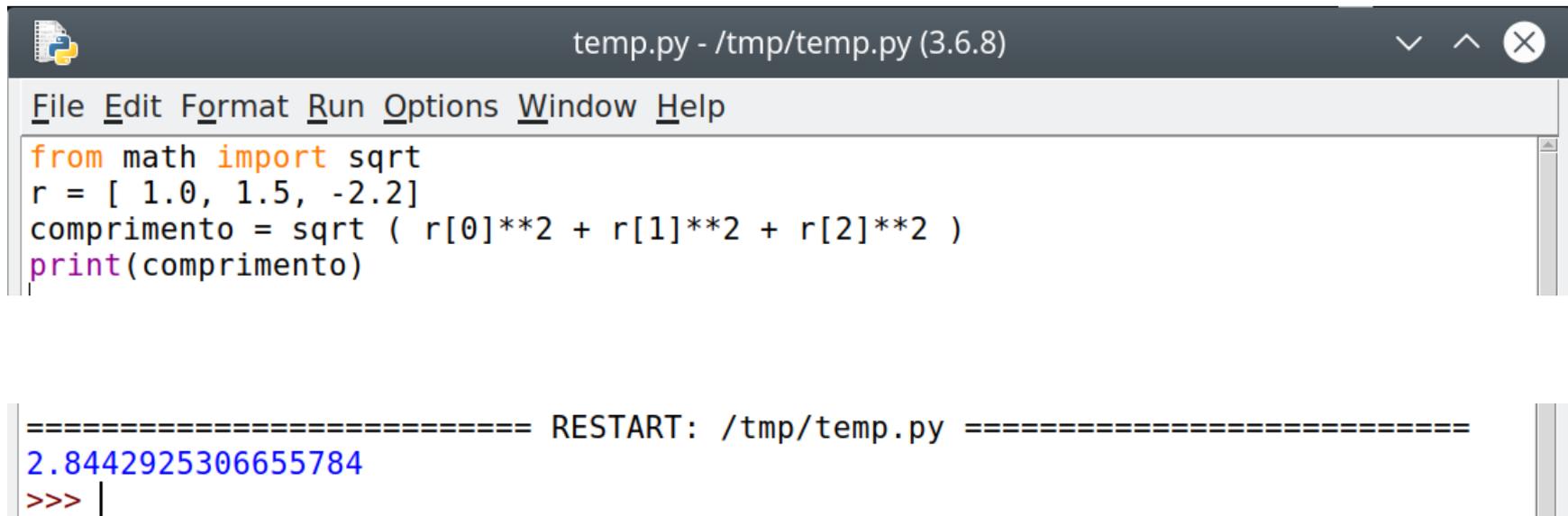
```
print(x, y, z)
```

```
print(r)
```

```
=====
[1, 1, 2]
0 1 2
[1, 1, 2]
>>>
```

# “Recipientes”: listas

- Para nos referirmos aos elementos individuais de uma lista `r`, usamos a notação `r[0]`, `r[1]`, `r[2]` etc. Note que o índice do primeiro elemento é `0`, não `1`.

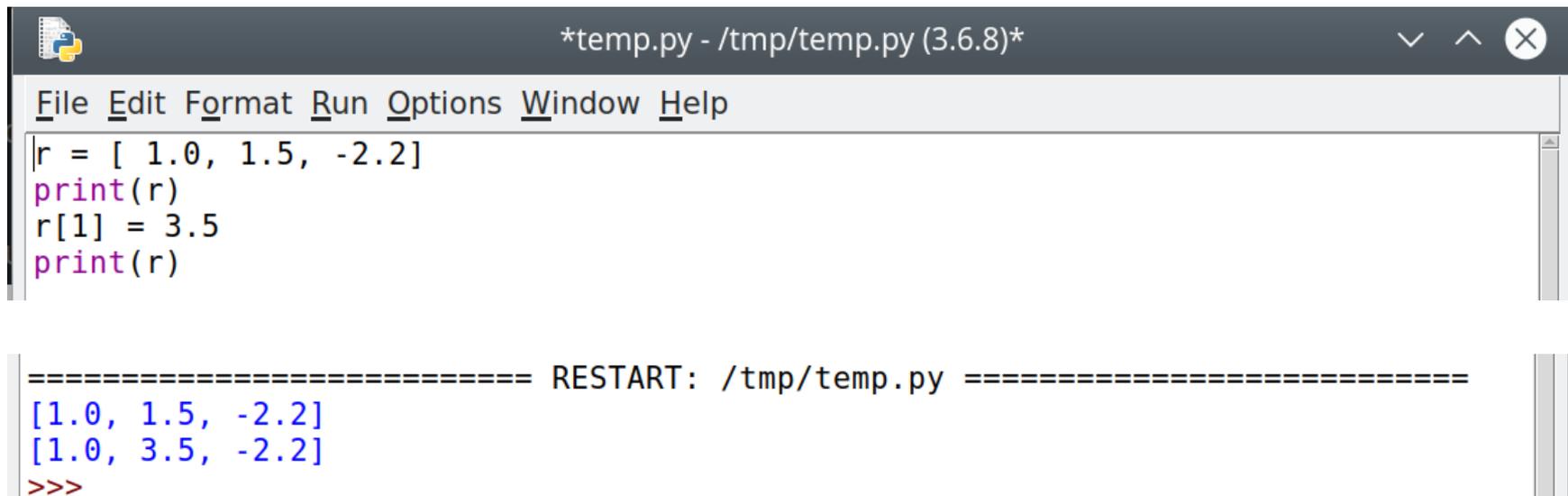


```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from math import sqrt
r = [ 1.0, 1.5, -2.2]
comprimento = sqrt ( r[0]**2 + r[1]**2 + r[2]**2 )
print(comprimento)

===== RESTART: /tmp/temp.py =====
2.8442925306655784
>>> |
```

# “Recipientes”: listas

- Podemos alterar um elemento específico de uma lista, como no exemplo abaixo.

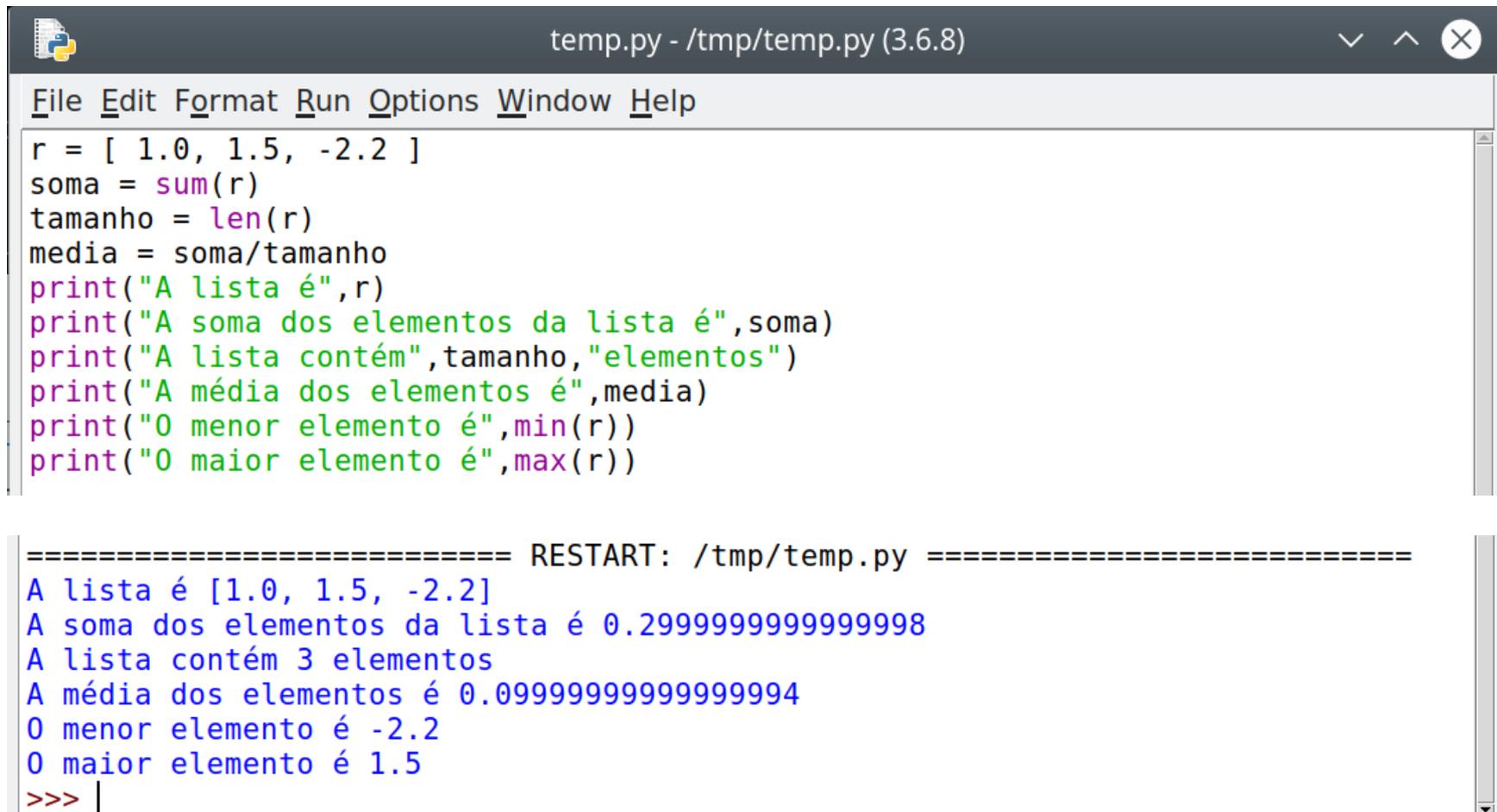


```
*temp.py - /tmp/temp.py (3.6.8)*
File Edit Format Run Options Window Help
r = [ 1.0, 1.5, -2.2]
print(r)
r[1] = 3.5
print(r)

===== RESTART: /tmp/temp.py =====
[1.0, 1.5, -2.2]
[1.0, 3.5, -2.2]
>>>
```

# “Recipientes”: listas

- Python dispõe de funções que permitem realizar operações sobre uma lista inteira. Observe o exemplo abaixo:

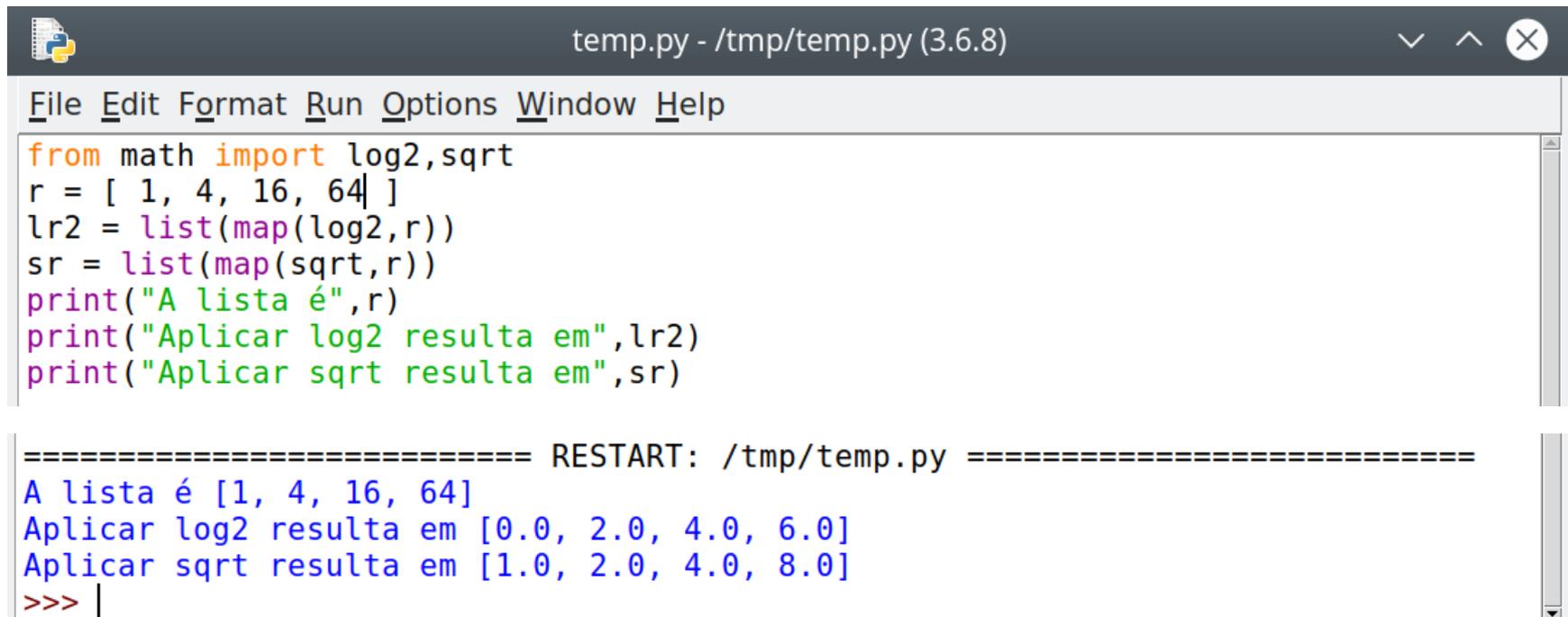


```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
r = [ 1.0, 1.5, -2.2 ]
soma = sum(r)
tamanho = len(r)
media = soma/tamanho
print("A lista é",r)
print("A soma dos elementos da lista é",soma)
print("A lista contém",tamanho,"elementos")
print("A média dos elementos é",media)
print("O menor elemento é",min(r))
print("O maior elemento é",max(r))

===== RESTART: /tmp/temp.py =====
A lista é [1.0, 1.5, -2.2]
A soma dos elementos da lista é 0.29999999999999998
A lista contém 3 elementos
A média dos elementos é 0.099999999999999994
O menor elemento é -2.2
O maior elemento é 1.5
>>> |
```

# “Recipientes”: listas

- Outro recurso útil é a função `map`, que permite aplicar uma outra função a todos os elementos de uma lista de uma só vez.



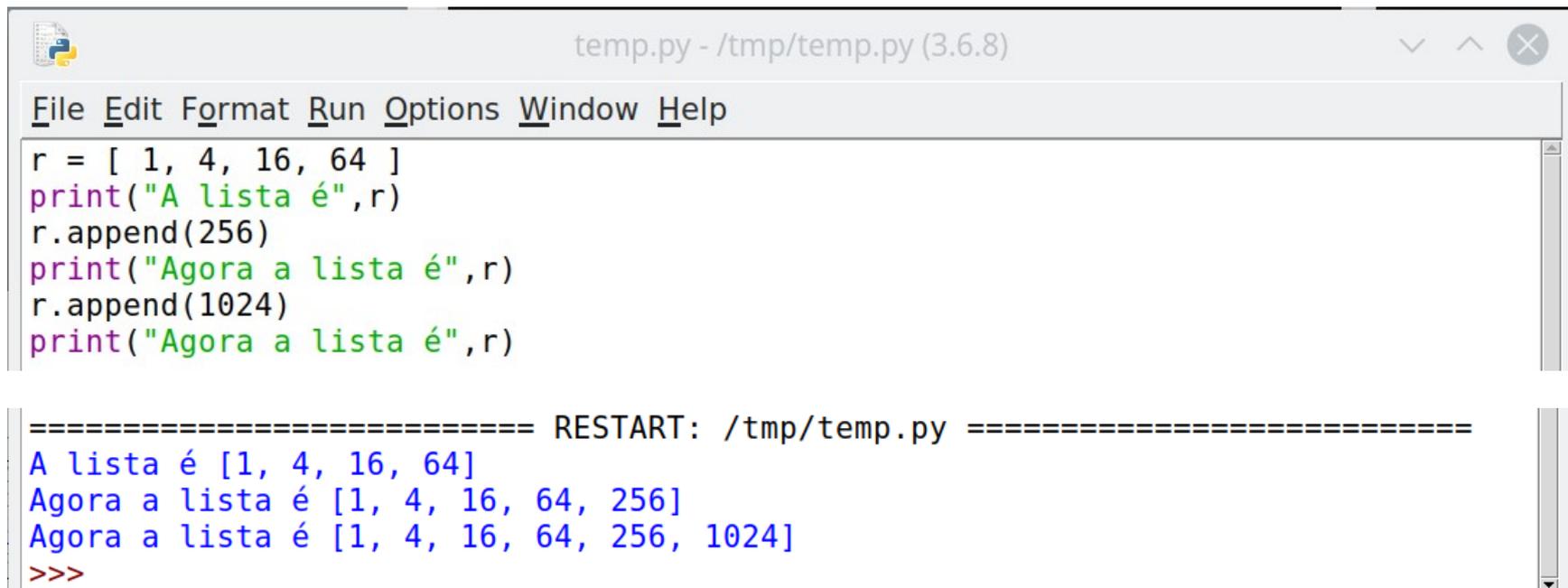
```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from math import log2,sqrt
r = [ 1, 4, 16, 64 ]
lr2 = list(map(log2,r))
sr = list(map(sqrt,r))
print("A lista é",r)
print("Aplicar log2 resulta em",lr2)
print("Aplicar sqrt resulta em",sr)

===== RESTART: /tmp/temp.py =====
A lista é [1, 4, 16, 64]
Aplicar log2 resulta em [0.0, 2.0, 4.0, 6.0]
Aplicar sqrt resulta em [1.0, 2.0, 4.0, 8.0]
>>> |
```

Note que foi preciso criar uma lista (`list`) após a aplicação da função `map`, que por si só produz apenas um *iterador*, algo semelhante a uma lista mas que não fica automaticamente armazenado na memória.

# “Recipientes”: listas

- Para adicionar um elemento ao final de uma lista utiliza-se o *método* `.append(<valor>)`



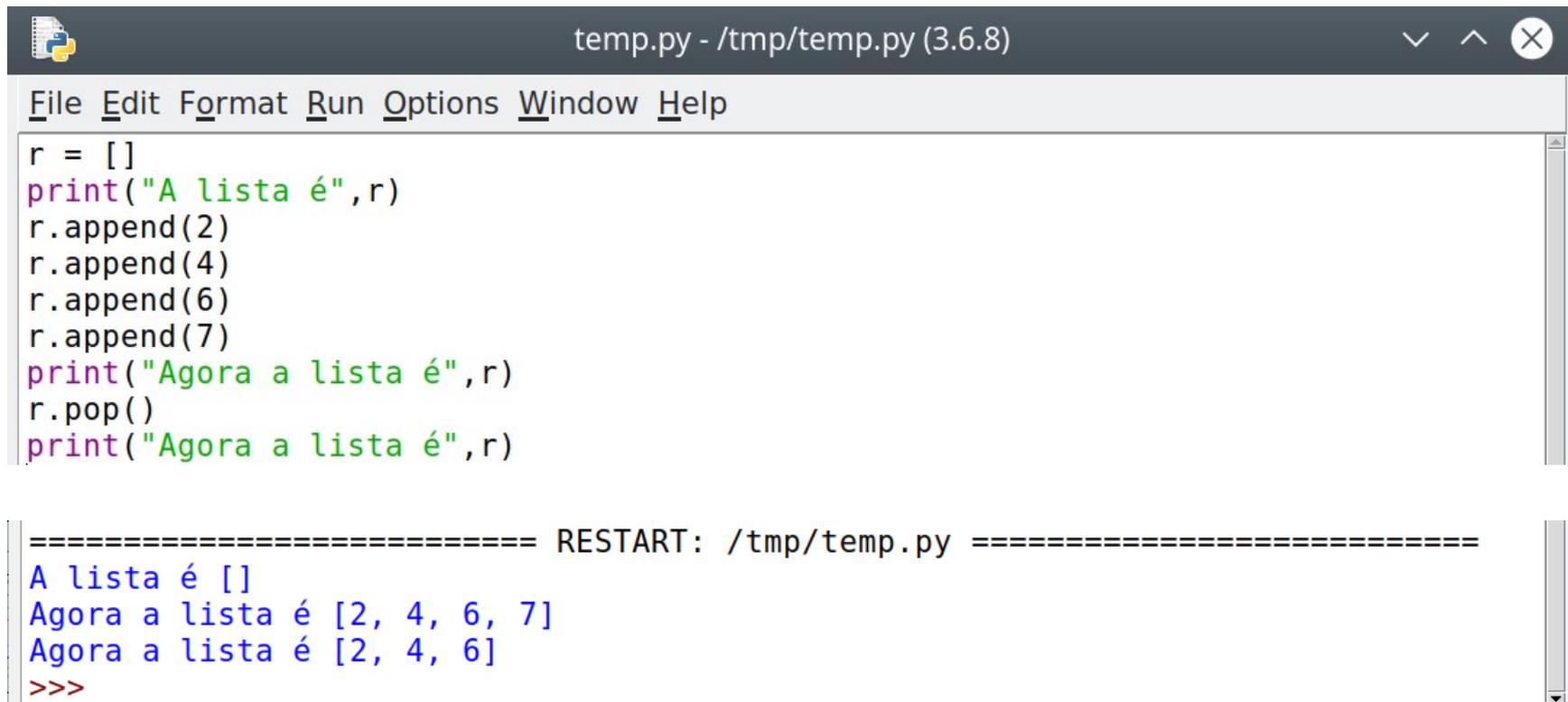
```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
r = [ 1, 4, 16, 64 ]
print("A lista é",r)
r.append(256)
print("Agora a lista é",r)
r.append(1024)
print("Agora a lista é",r)

===== RESTART: /tmp/temp.py =====
A lista é [1, 4, 16, 64]
Agora a lista é [1, 4, 16, 64, 256]
Agora a lista é [1, 4, 16, 64, 256, 1024]
>>>
```

Assim como os atributos, que discutimos na aula anterior, os métodos são exemplos da orientação a objetos do Python. Para números complexos, um método conveniente é o `.conjugate()`, que produz o conjugado de um número: `zc = z.conjugate()`

# “Recipientes”: listas

- Uma lista pode ser criada ainda vazia, e também é possível remover elementos do final de uma lista usando o *método* `.pop()`



```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
r = []
print("A lista é",r)
r.append(2)
r.append(4)
r.append(6)
r.append(7)
print("Agora a lista é",r)
r.pop()
print("Agora a lista é",r)

===== RESTART: /tmp/temp.py =====
A lista é []
Agora a lista é [2, 4, 6, 7]
Agora a lista é [2, 4, 6]
>>>
```

# “Recipientes”: listas

- É possível “fatiar” uma lista, selecionando apenas os elementos em uma faixa de interesse. Veja o exemplo a seguir:

```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
r = [ 1, 3, 5, 7, 9, 11, 13]
print("A lista é",r)
s = r[2:5]
print("O terceiro, o quarto e o quinto elementos formam a lista",s)
t = r[:5]
print("Até o quinto elemento a lista é",t)
u = r[2:]
print("A partir do terceiro elemento a lista é",u)

===== RESTART: /tmp/temp.py =====
A lista é [1, 3, 5, 7, 9, 11, 13]
O terceiro, o quarto e o quinto elementos formam a lista [5, 7, 9]
Até o quinto elemento a lista é [1, 3, 5, 7, 9]
A partir do terceiro elemento a lista é [5, 7, 9, 11, 13]
>>> |
```

# “Recipientes”: arrays

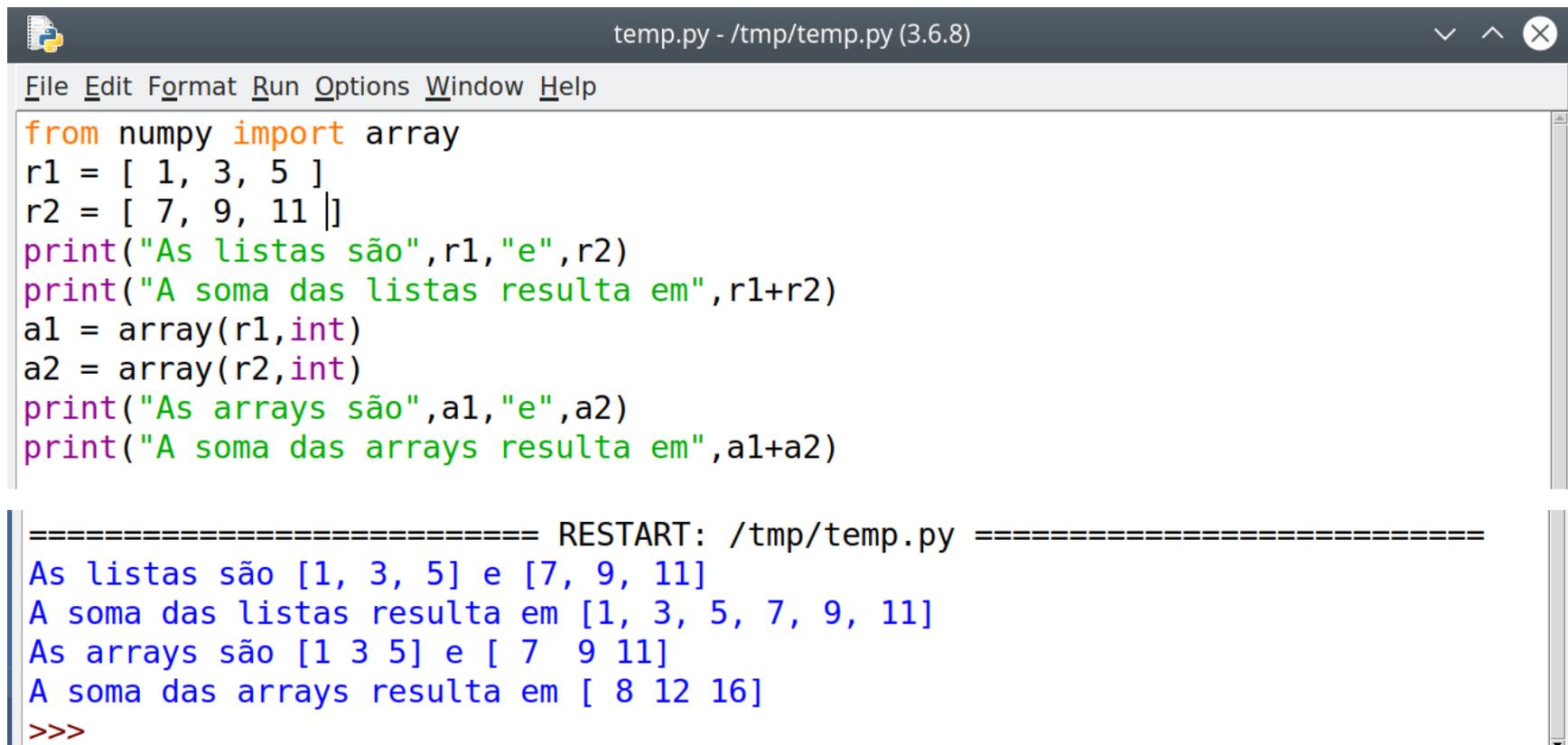
- À primeira vista, uma lista pode parecer um vetor. No entanto, observem o resultado de tentar somar duas listas:

```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
r = [ 1, 3, 5]
print("A primeira lista é ",r)
s = [ 7, 9, 11]
print("A segunda lista é ",s)
print("A soma das duas listas resulta em",r+s)

===== RESTART: /tmp/temp.py =====
A primeira lista é [1, 3, 5]
A segunda lista é [7, 9, 11]
A soma das duas listas resulta em [1, 3, 5, 7, 9, 11]
>>> |
```

# “Recipientes”: arrays

- Para trabalhar com vetores e matrizes, vamos utilizar o objeto `array` definido pelo pacote `numpy`. O exemplo abaixo cria vetores.



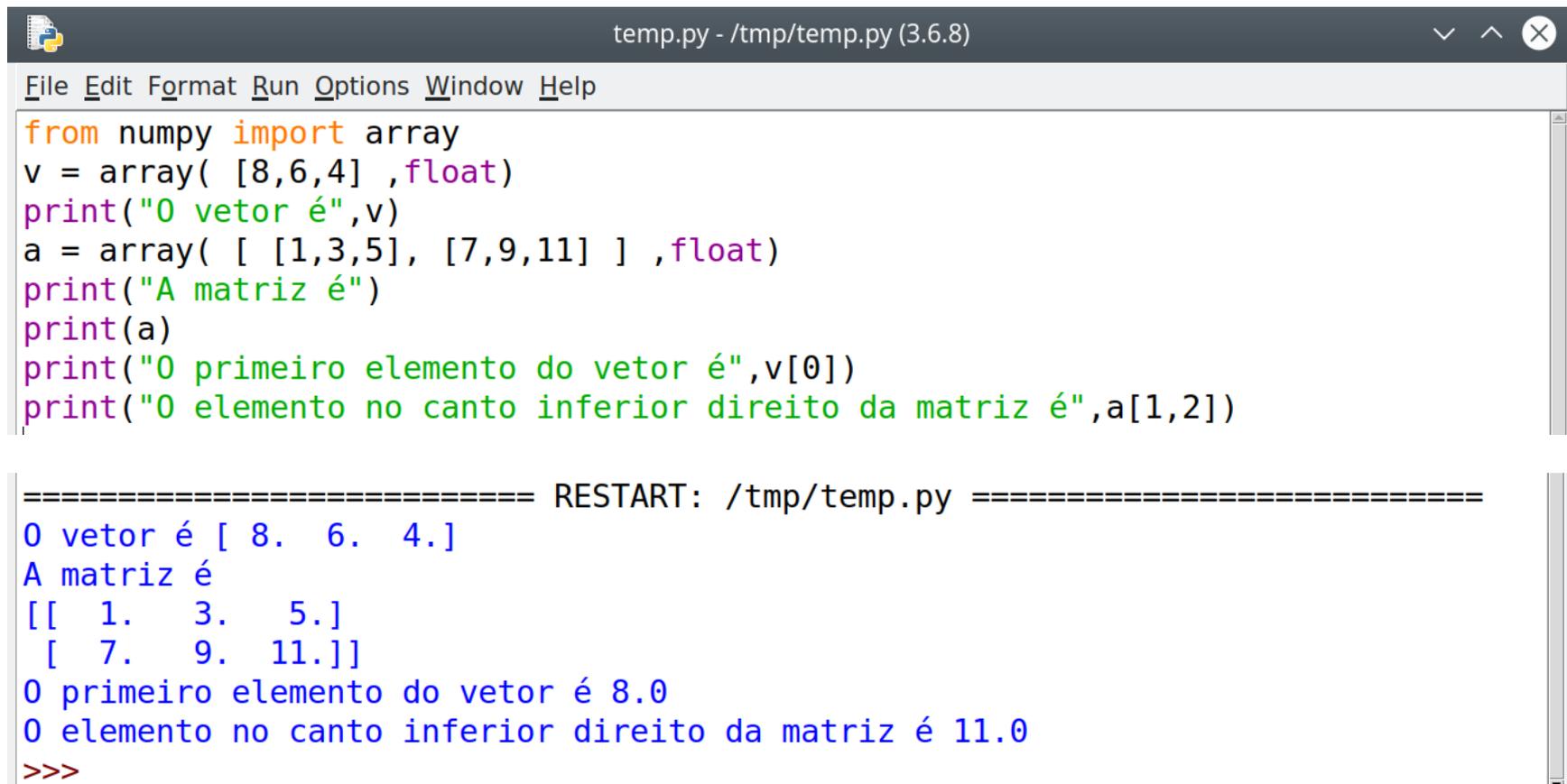
```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array
r1 = [ 1, 3, 5 ]
r2 = [ 7, 9, 11 ]
print("As listas são",r1,"e",r2)
print("A soma das listas resulta em",r1+r2)
a1 = array(r1,int)
a2 = array(r2,int)
print("As arrays são",a1,"e",a2)
print("A soma das arrays resulta em",a1+a2)

===== RESTART: /tmp/temp.py =====
As listas são [1, 3, 5] e [7, 9, 11]
A soma das listas resulta em [1, 3, 5, 7, 9, 11]
As arrays são [1 3 5] e [ 7  9 11]
A soma das arrays resulta em [ 8 12 16]
>>>
```

Para criar uma `array` de números reais ou complexos, basta substituir o argumento `int` por `float` ou `complex`.

# “Recipientes”: arrays

- Para criar uma matriz, basta utilizar uma lista de listas como argumento para `array`:



```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array
v = array( [8,6,4] ,float)
print("O vetor é",v)
a = array( [ [1,3,5], [7,9,11] ] ,float)
print("A matriz é")
print(a)
print("O primeiro elemento do vetor é",v[0])
print("O elemento no canto inferior direito da matriz é",a[1,2])

===== RESTART: /tmp/temp.py =====
O vetor é [ 8.  6.  4.]
A matriz é
[[ 1.  3.  5.]
 [ 7.  9. 11.]]
O primeiro elemento do vetor é 8.0
O elemento no canto inferior direito da matriz é 11.0
>>>
```

Note como é feita a referência aos elementos de um vetor e de uma matriz, e lembre que os índices começam de 0, não 1.

# “Recipientes”: arrays

- O pacote `numpy` permite criar arrays de 0s ou 1s. Também é possível criar uma array vazia e preencher seus elementos posteriormente.

```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import zeros,ones,empty
v = zeros(4,float)
print("0 vetor é",v)
a = ones([2,3],float)
print("A primeira matriz é")
print(a)
b = empty([2,2],float)
b[0,0],b[0,1],b[1,0],b[1,1] = 1,2,4,9
print("A segunda matriz é")
print(b)

===== RESTART: /tmp/temp.py =====
0 vetor é [ 0.  0.  0.  0.]
A primeira matriz é
[[ 1.  1.  1.]
 [ 1.  1.  1.]]
A segunda matriz é
[[ 1.  2.]
 [ 4.  9.]]
>>>
```

# Aritmética com arrays

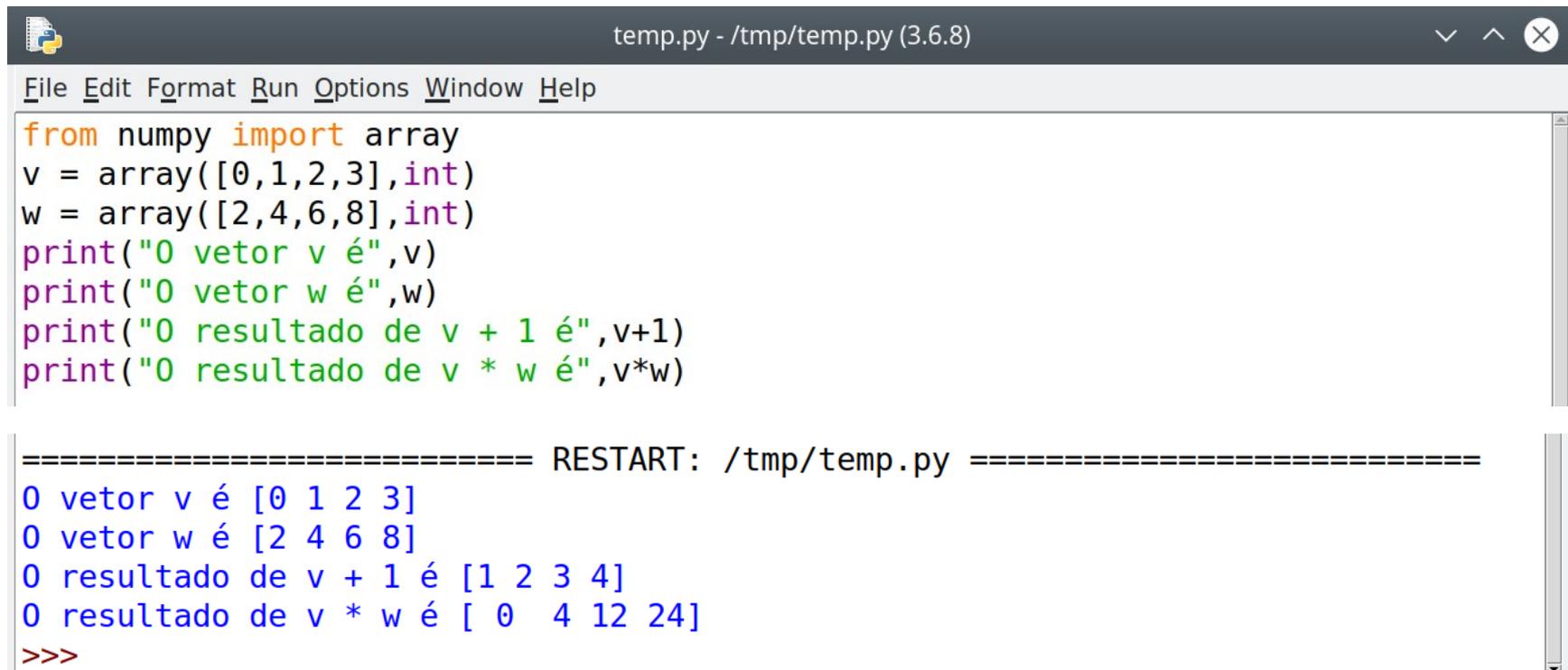
- Além das operações envolvendo os elementos individuais de uma array, é possível realizar operações que envolvem a array inteira:

```
*temp.py - /tmp/temp.py (3.6.8)*
File Edit Format Run Options Window Help
from numpy import array
v = array([1,2,3,4],int)
w = 2*v
print("0 primeiro vetor é",v)
print("0 segundo vetor, igual a 2 vezes o primeiro, é",w)
print("A soma dos dois vetores é",v+w)

===== RESTART: /tmp/temp.py =====
0 primeiro vetor é [1 2 3 4]
0 segundo vetor, igual a 2 vezes o primeiro, é [2 4 6 8]
A soma dos dois vetores é [ 3  6  9 12]
>>> |
```

# Aritmética com arrays

- Podemos tentar outras operações, como somar um número a uma array e a operação  $*$  entre duas arrays.



```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array
v = array([0,1,2,3],int)
w = array([2,4,6,8],int)
print("0 vetor v é",v)
print("0 vetor w é",w)
print("0 resultado de v + 1 é",v+1)
print("0 resultado de v * w é",v*w)

===== RESTART: /tmp/temp.py =====
0 vetor v é [0 1 2 3]
0 vetor w é [2 4 6 8]
0 resultado de v + 1 é [1 2 3 4]
0 resultado de v * w é [ 0  4 12 24]
>>>
```

Note que na adição o número é somado a cada elemento do vetor, e na multiplicação os elementos correspondentes são multiplicados entre si, o que **não** corresponde a um produto escalar ou vetorial.

# Aritmética com arrays

- Para realizar os produtos escalar e vetorial, o `numpy` dispõe das funções `dot` e `cross`.

```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array,dot,cross
v = array([1,2,3],int)
w = array([4,5,6],int)
print("0 vetor v é",v)
print("0 vetor w é",w)
print("Seu produto escalar é",dot(v,w))
print("Seu produto vetorial é",cross(v,w))

===== RESTART: /tmp/temp.py =====
0 vetor v é [1 2 3]
0 vetor w é [4 5 6]
Seu produto escalar é 32
Seu produto vetorial é [-3  6 -3]
>>>
```

# Aritmética com arrays

- A operação `dot`, a adição entre arrays e a multiplicação de uma array por um número funcionam também com matrizes

```
File Edit Format Run Options Window Help
from numpy import array,dot
a = array( [[1,3],[2,4]] ,int)
b = array( [[4,-2],[-3,1]], int)
print("A matriz a é")
print(a)
print("A matriz b é")
print(b)
print("O produto 2*a resulta em")
print(2*a)
print("O resultado de dot(a,b) é",dot(a,b))
v = array( [2,1], int)
print("Agora definimos um vetor v igual a",v)
print("O resultado de dot(a,v) é",dot(a,v))
print("O resultado de dot(v,b) é",dot(v,b))

===== RESTART: /tmp,
A matriz a é
[[1 3]
 [2 4]]
A matriz b é
[[ 4 -2]
 [-3  1]]
O produto 2*a resulta em
[[2 6]
 [4 8]]
O resultado de dot(a,b) é [[-5  1]
 [-4  0]]
Agora definimos um vetor v igual a [2 1]
O resultado de dot(a,v) é [5 8]
O resultado de dot(v,b) é [ 5 -3]
>>> |
```

Note que ao operar com uma matriz e um vetor a função `dot` trata o vetor como coluna se ele for o segundo argumento e como linha se ele for o primeiro argumento.

# Aritmética com arrays

- É possível realizar várias operações mais sofisticadas com vetores, mas via de regra isso funciona mal com matrizes.

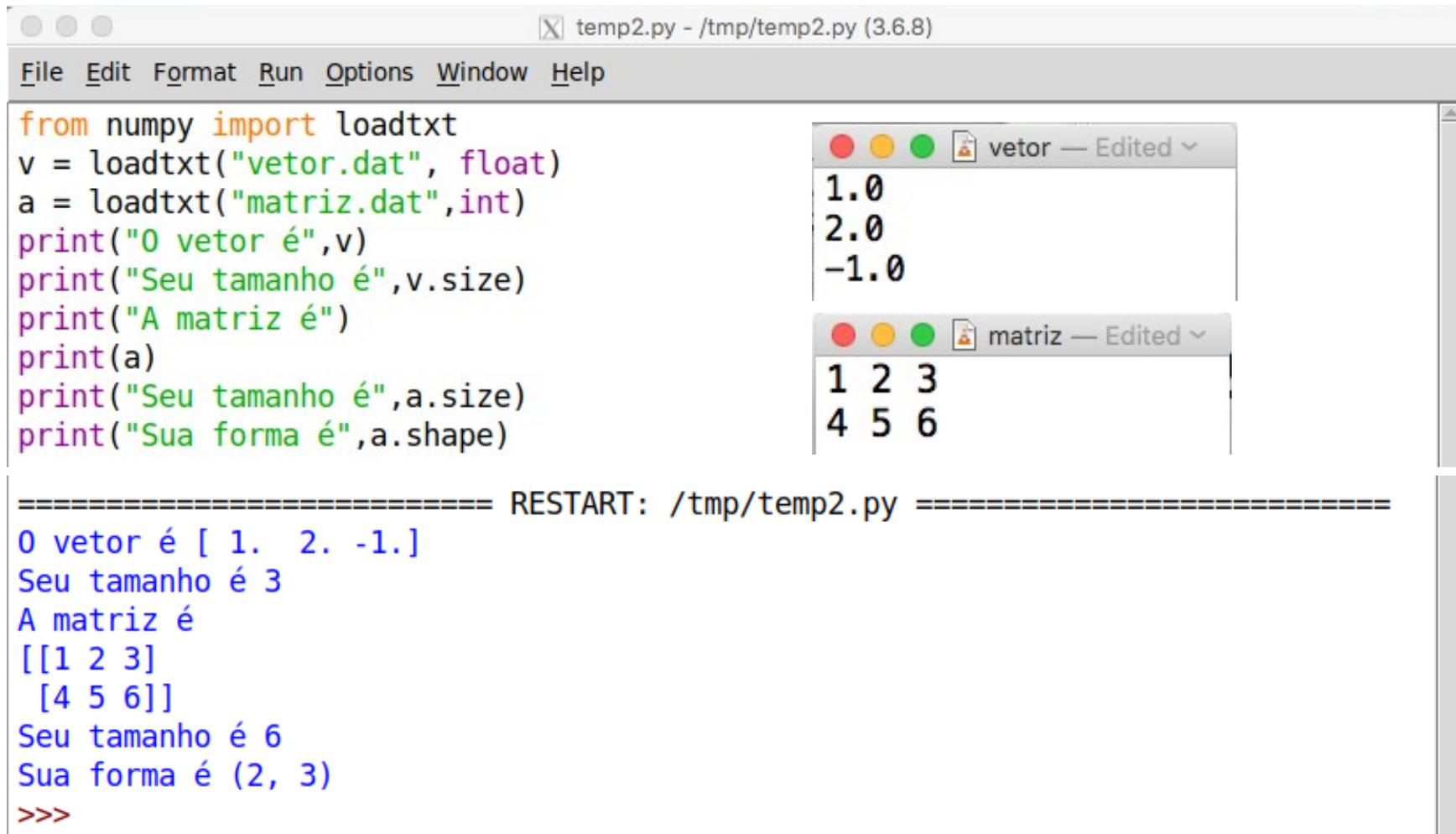
```
temp2.py - /tmp/temp2.py (3.6.8)
File Edit Format Run Options Window Help

from numpy import array
from math import sqrt
v = array( [9,16], int)
w = array(list(map(sqrt,v)),int)
print("0 vetor é",v)
print("Transformá-lo com sqrt resulta em",w)
a = array( [[1,4],[25,36]] ,int)
b = array(list(map(sqrt,a)),int)

===== RESTART: /tmp/temp2.py =====
0 vetor é [ 9 16]
Transformá-lo com sqrt resulta em [3 4]
Traceback (most recent call last):
  File "/tmp/temp2.py", line 8, in <module>
    b = array(list(map(sqrt,a)),int)
TypeError: only length-1 arrays can be converted to Python scalars
>>>
```

# Aritmética com arrays

- É possível definir um objeto `array` a partir de arquivos de dados. Além disso, objetos `array` possuem atributos que dão informações úteis.



```
temp2.py - /tmp/temp2.py (3.6.8)
File Edit Format Run Options Window Help

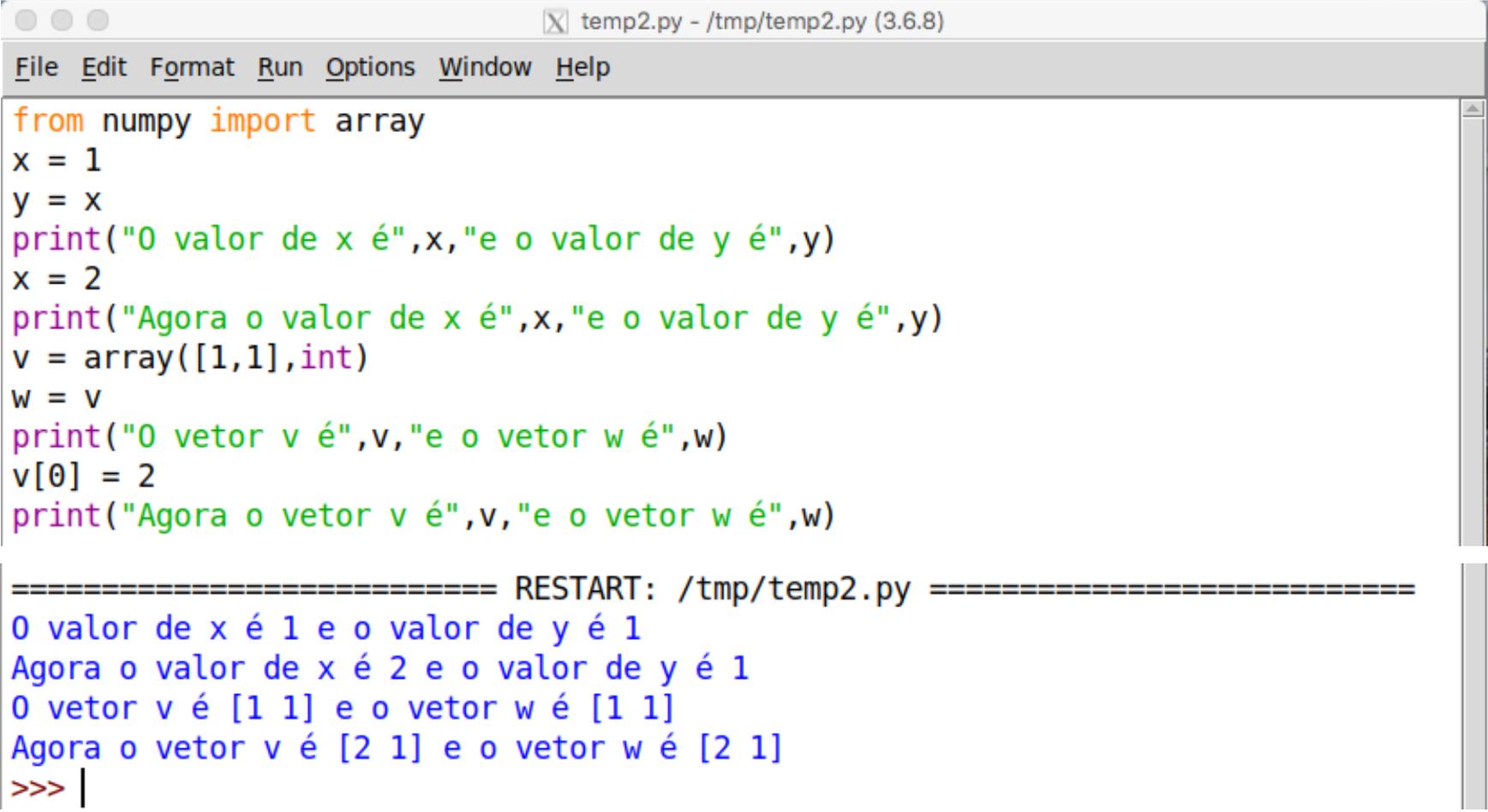
from numpy import loadtxt
v = loadtxt("vetor.dat", float)
a = loadtxt("matriz.dat", int)
print("O vetor é",v)
print("Seu tamanho é",v.size)
print("A matriz é")
print(a)
print("Seu tamanho é",a.size)
print("Sua forma é",a.shape)

===== RESTART: /tmp/temp2.py =====
O vetor é [ 1.  2. -1.]
Seu tamanho é 3
A matriz é
[[1 2 3]
 [4 5 6]]
Seu tamanho é 6
Sua forma é (2, 3)
>>>
```

The screenshot shows a Python IDE window titled "temp2.py - /tmp/temp2.py (3.6.8)". The code in the editor defines two NumPy arrays: a 1D array 'v' from 'vetor.dat' and a 2D array 'a' from 'matriz.dat'. The output in the console shows the arrays and their attributes. Two preview windows are open: 'vetor' showing the values [1.0, 2.0, -1.0] and 'matriz' showing the matrix [[1, 2, 3], [4, 5, 6]].

# Aritmética com arrays

- Finalmente, veja no exemplo abaixo um comportamento peculiar dos objetos `array`.



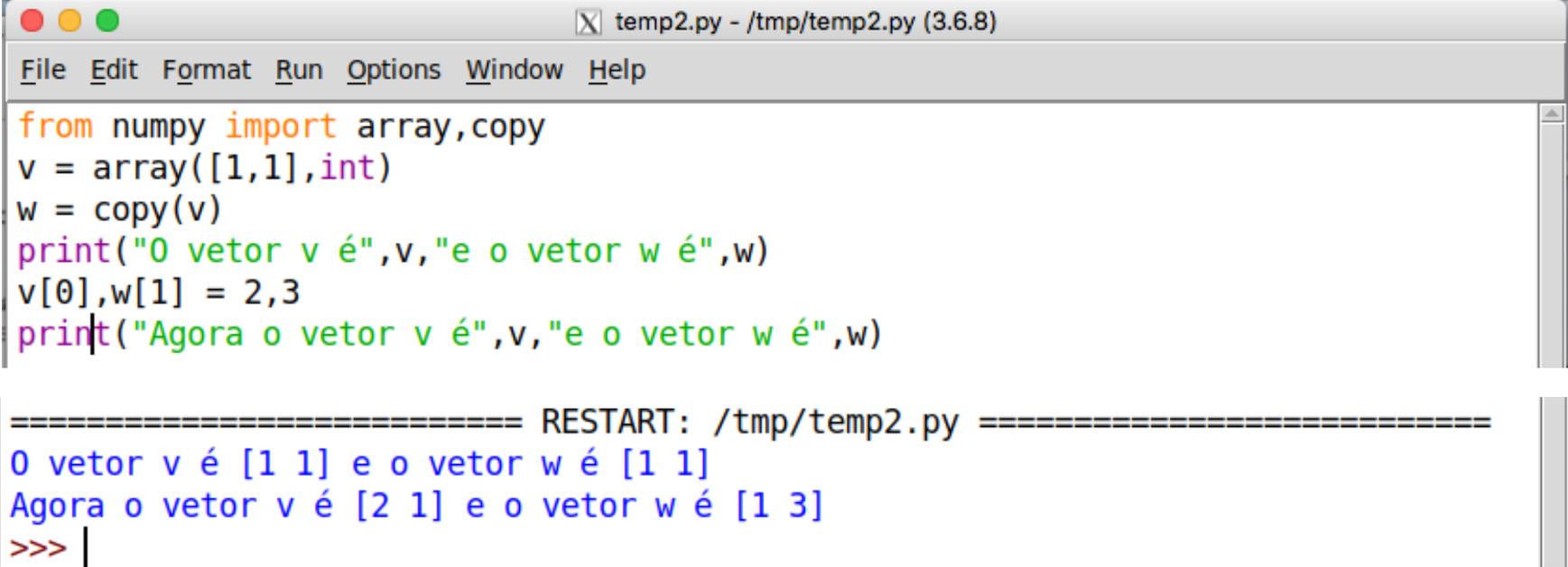
```
temp2.py - /tmp/temp2.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array
x = 1
y = x
print("O valor de x é",x,"e o valor de y é",y)
x = 2
print("Agora o valor de x é",x,"e o valor de y é",y)
v = array([1,1],int)
w = v
print("O vetor v é",v,"e o vetor w é",w)
v[0] = 2
print("Agora o vetor v é",v,"e o vetor w é",w)

===== RESTART: /tmp/temp2.py =====
O valor de x é 1 e o valor de y é 1
Agora o valor de x é 2 e o valor de y é 1
O vetor v é [1 1] e o vetor w é [1 1]
Agora o vetor v é [2 1] e o vetor w é [2 1]
>>> |
```

A instrução `y = x` cria uma nova variável com o valor que `x` guarda naquele momento, mas a instrução `w = v` cria um “sinônimo” para `v`.

# Aritmética com arrays

- Se quiser criar uma nova `array` com o valor de uma antiga, use a função `copy` do `numpy`.



```
temp2.py - /tmp/temp2.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array,copy
v = array([1,1],int)
w = copy(v)
print("O vetor v é",v,"e o vetor w é",w)
v[0],w[1] = 2,3
print("Agora o vetor v é",v,"e o vetor w é",w)

===== RESTART: /tmp/temp2.py =====
0 vetor v é [1 1] e o vetor w é [1 1]
Agora o vetor v é [2 1] e o vetor w é [1 3]
>>> |
```

A função `copy` pode tornar a execução do programa muito lenta se a `array` a ser copiada for muito grande.

# Exercício 2

Suponha que dois vetores  $a$  e  $b$  sejam definidos da forma abaixo:

```
from numpy import array  
a = array([1,2,3,4],int)  
b = array([2,4,6,8],int)
```

Que resposta o computador dará a cada uma das seguintes instruções? Verifique após pensar.

- a) `print (b/a+1)`
- b) `print (b/ (a+1) )`
- c) `print (1/a)`

Essas operações (e suas respostas) correspondem a alguma operação matemática útil em física?

# Exercício 2: solução

```
temp2.py - /tmp/temp2.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import array
a = array([1,2,3,4],int)
b = array([2,4,6,8],int)
print(b/a+1)
print(b/(a+1))
print(1/a)|

===== RESTART: /tmp/temp2.py =====
[ 3.  3.  3.  3.]
[ 1.          1.33333333  1.5          1.6          ]
[ 1.          0.5          0.33333333  0.25          ]
>>>
```

Essas operações (e suas respostas) correspondem a alguma operação matemática útil em física?

# Para a próxima aula

- Exercícios no moodle (não são ainda um dos trabalhos utilizados para a nota no curso)