

Introdução à Física Computacional I (4300218)

Prof. André Vieira
apvieira@if.usp.br
Sala 3120 – Edifício Principal

Aula 3

Programação em Python para físicos:
laços `for` e funções definidas pelo usuário

Laços `for`

- Um laço `for` repete instruções enquanto percorre os elementos de uma lista ou array.



```
teste.py - /tmp/teste.py (3.6.8)
File Edit Format Run Options Window Help
r = [ 1, 3, 5 ]
for n in r:
    print(n,2*n)
print ("Programa finalizado")

===== RESTART: /tmp/teste.py =====
1 2
3 6
5 10
Programa finalizado
>>>
```

Note a indentação do que se quer repetir no laço.

Laços `for`

- Se quisermos repetir os comandos no interior do laço um número definido de vezes, é mais prático utilizar a função `range`.

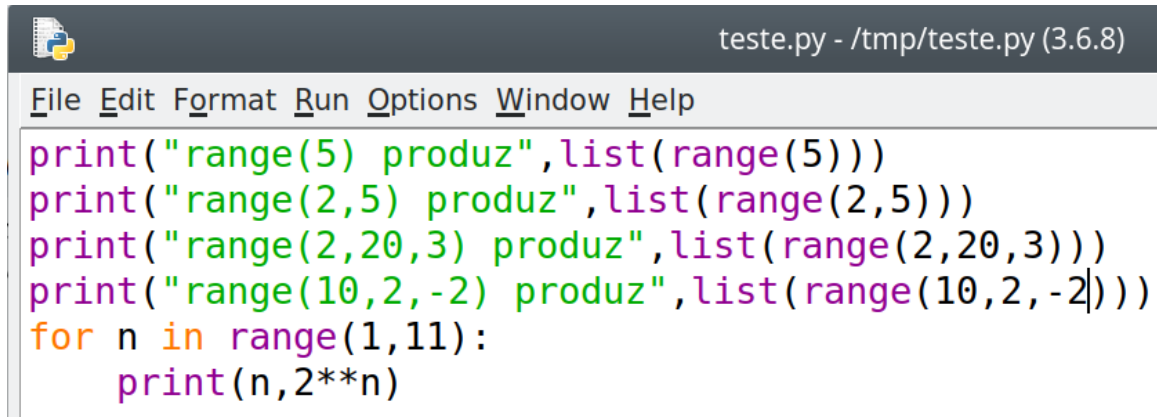
```
teste.py - /tmp/teste.py (3.6.8)
File Edit Format Run Options Window Help
N = 10
print(range(N))
print(list(range(N)))
for n in range(N):
    if n == 5:
        print("Alô!!!!")
        continue
    print("Olá!")
print("Programa finalizado")

===== RES
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Olá!
Olá!
Olá!
Olá!
Olá!
Alô!!!!
Olá!
Olá!
Olá!
Olá!
Programa finalizado
>>>
```

Note que a função `range` gera um iterador, que é aceito pelo `for` como se fosse uma lista. Note também o uso de `continue`, como nos laços `while`. Também se poderia usar `break`.

Laços `for`

- A função `range` aceita até 3 argumentos **inteiros**: <o primeiro número do intervalo>, o “último” número do intervalo e <o passo>.



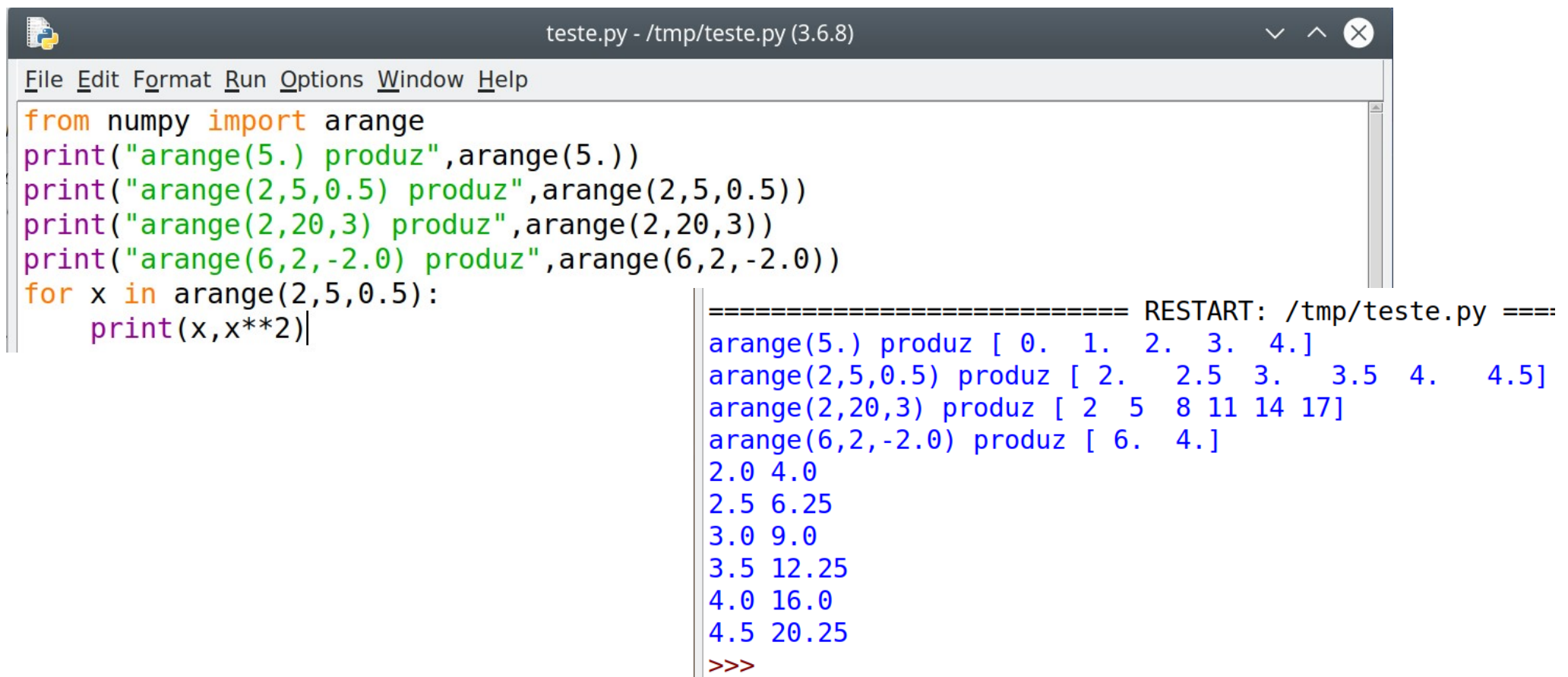
```
File Edit Format Run Options Window Help
print("range(5) produz",list(range(5)))
print("range(2,5) produz",list(range(2,5)))
print("range(2,20,3) produz",list(range(2,20,3)))
print("range(10,2,-2) produz",list(range(10,2,-2)))
for n in range(1,11):
    print(n,2**n)
```

```
===== RESTART: /tmp/
range(5) produz [0, 1, 2, 3, 4]
range(2,5) produz [2, 3, 4]
range(2,20,3) produz [2, 5, 8, 11, 14, 17]
range(10,2,-2) produz [10, 8, 6, 4]
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024
>>>
```

Note que a lista produzida é interrompida no penúltimo elemento. Se a função é invocada com um único argumento, este é interpretado como sendo o “último” número do intervalo, o primeiro número sendo definido como 0 e o passo, como 1.

Laços for

- Para produzir listas com números reais, utilize a função `arange` do pacote `numpy`.



The screenshot shows a Python IDE window titled "teste.py - /tmp/teste.py (3.6.8)". The code in the editor is as follows:

```
from numpy import arange
print("arange(5.) produz", arange(5.))
print("arange(2,5,0.5) produz", arange(2,5,0.5))
print("arange(2,20,3) produz", arange(2,20,3))
print("arange(6,2,-2.0) produz", arange(6,2,-2.0))
for x in arange(2,5,0.5):
    print(x, x**2)
```

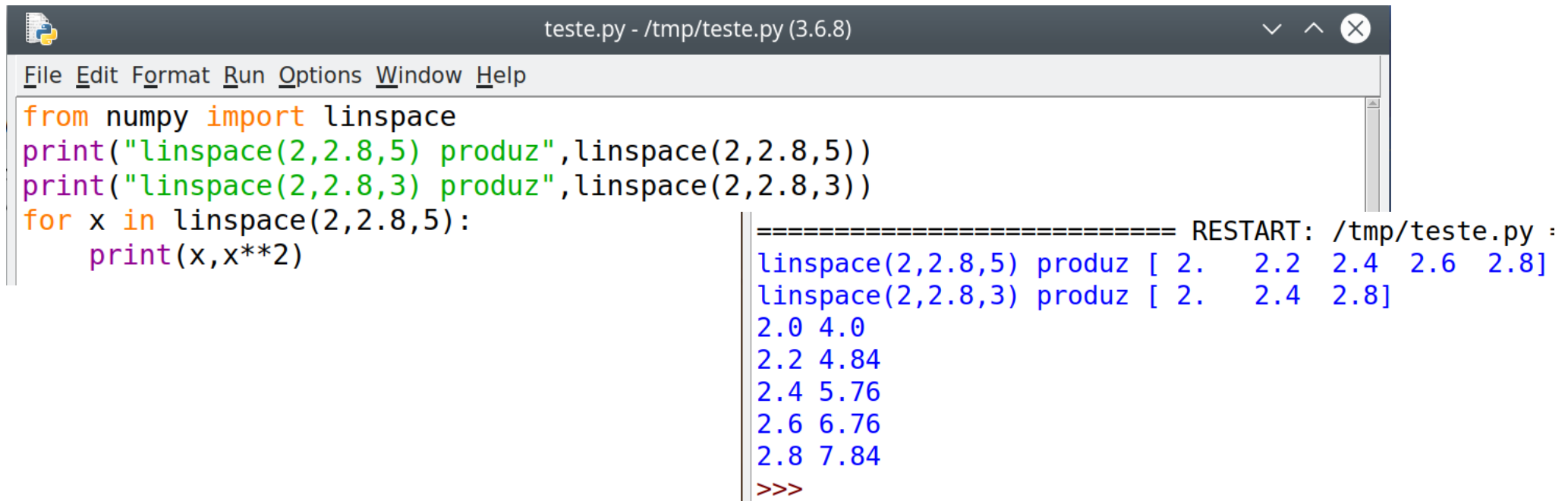
The output of the program is displayed on the right side of the window, separated by a "RESTART: /tmp/teste.py ===" line. The output shows the arrays produced by `arange` and the results of the `for` loop:

```
arange(5.) produz [ 0.  1.  2.  3.  4.]
arange(2,5,0.5) produz [ 2.   2.5  3.   3.5  4.   4.5]
arange(2,20,3) produz [ 2  5  8 11 14 17]
arange(6,2,-2.0) produz [ 6.  4.]
2.0 4.0
2.5 6.25
3.0 9.0
3.5 12.25
4.0 16.0
4.5 20.25
>>>
```

Note que `arange` produz uma array, não uma lista, mas o laço `for` percorre sem problemas os elementos dessa array.

Laços for

- Outra opção é a função `linspace`, também do pacote `numpy`.



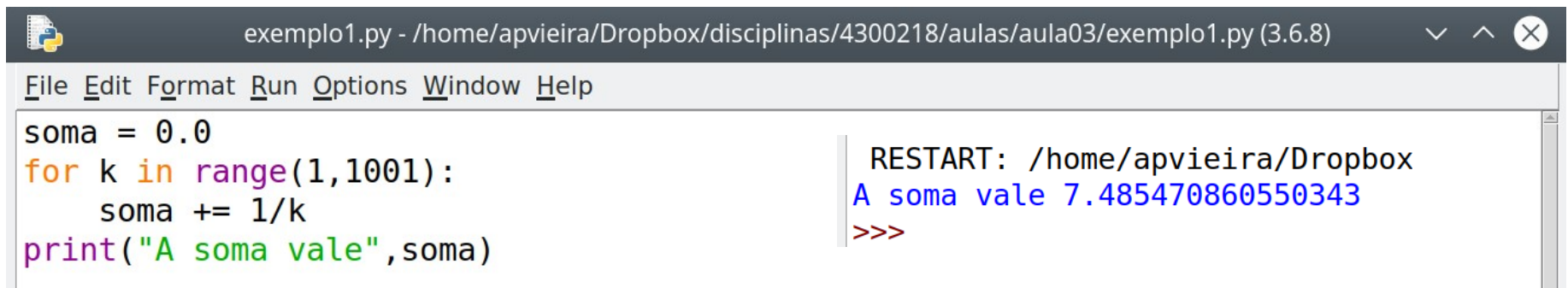
```
teste.py - /tmp/teste.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import linspace
print("linspace(2,2.8,5) produz",linspace(2,2.8,5))
print("linspace(2,2.8,3) produz",linspace(2,2.8,3))
for x in linspace(2,2.8,5):
    print(x,x**2)
```

===== RESTART: /tmp/teste.py :
linspace(2,2.8,5) produz [2. 2.2 2.4 2.6 2.8]
linspace(2,2.8,3) produz [2. 2.4 2.8]
2.0 4.0
2.2 4.84
2.4 5.76
2.6 6.76
2.8 7.84
>>>

Note que `linspace` utiliza três argumentos: o valor do elemento inicial, o valor do elemento final (que é de fato alcançado na array produzida) e a quantidade de elementos a produzir. (Se omitido o último argumento, são produzidos 50 elementos.)

Laços `for`

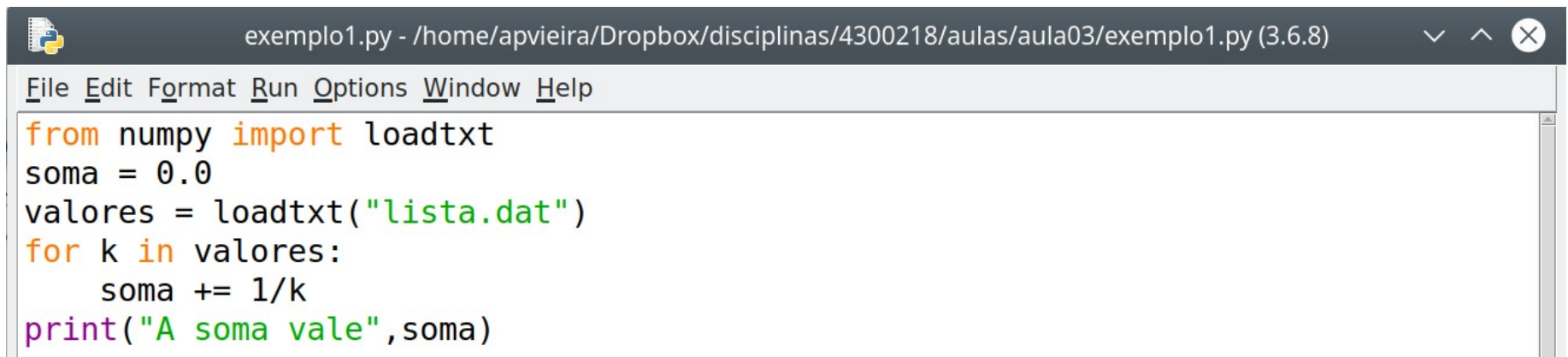
- Exemplo 1: vamos somar o inverso de todos os números entre 1 e 1000.



```
exemplo1.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo1.py (3.6.8)
File Edit Format Run Options Window Help
soma = 0.0
for k in range(1,1001):
    soma += 1/k
print("A soma vale",soma)

RESTART: /home/apvieira/Dropbox
A soma vale 7.485470860550343
>>>
```

Também poderíamos ler a lista de valores da variável a partir de um arquivo:



```
exemplo1.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo1.py (3.6.8)
File Edit Format Run Options Window Help
from numpy import loadtxt
soma = 0.0
valores = loadtxt("lista.dat")
for k in valores:
    soma += 1/k
print("A soma vale",soma)
```

Laços for

- Exemplo 2: linhas de emissão do átomo de hidrogênio.

$$\frac{1}{\lambda} = R \left(\frac{1}{m^2} - \frac{1}{n^2} \right)$$

```
exemplo2.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/
File Edit Format Run Options Window Help
R = 1.0968e-2
for m in range(1,4):
    print("Série para m =",m)
    for n in range(m+1,m+6):
        invlambda = R*(1/m**2 - 1/n**2)
        print("  ",1/invlambda," nm")
```

```
RESTART: /home/apvieira/Dr
Série para m = 1
121.56576707999027 nm
102.57111597374178 nm
97.25261366399222 nm
94.9732555312424 nm
93.77930603313536 nm
Série para m = 2
656.4551422319474 nm
486.26306831996106 nm
434.16345385710815 nm
410.28446389496713 nm
397.11483912796825 nm
Série para m = 3
1875.5861206627073 nm
1282.1389496717727 nm
1094.0919037199126 nm
1005.1969365426696 nm
954.8438432464691 nm
>>>
```


Exercício 1

Escreva um programa que faça uso de um laço `for` para calcular π usando os primeiros vinte termos da série de Madhava:

$$\pi = \sqrt{12} \left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \dots \right).$$

Exercício 1: solução

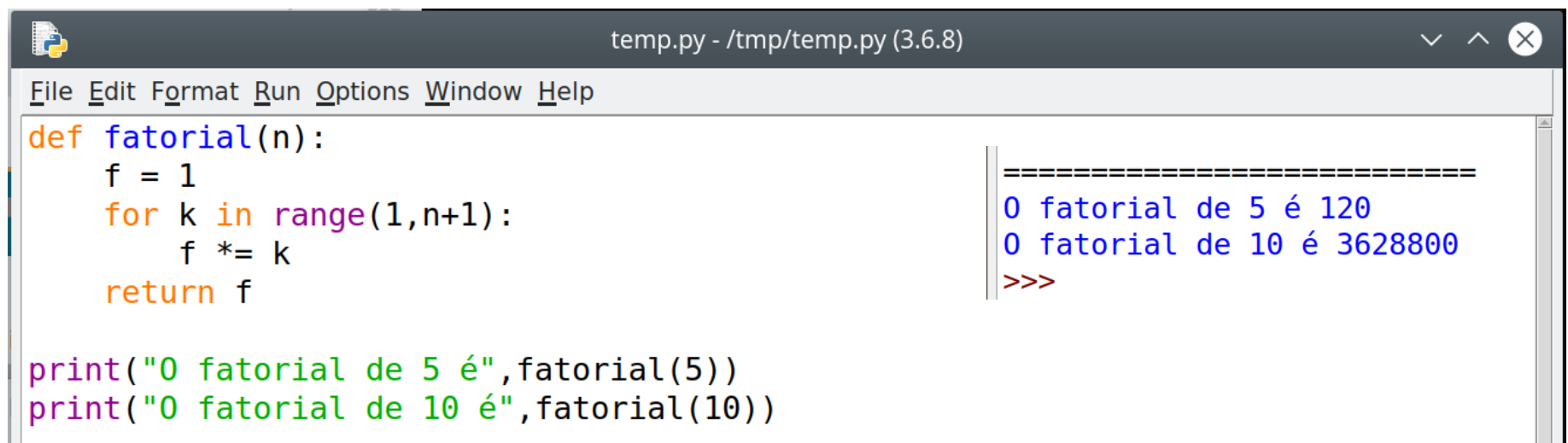
$$\pi = \sqrt{12} \left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \dots \right)$$

```
exercicio1.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exercicio1.py (3.6.8)
File Edit Format Run Options Window Help
from math import pi,sqrt
soma = 0.0
for n in range(0,20):
    soma += (-1)**n/((2*n+1)*3**n)
soma *= sqrt(12)
print("A estimativa para pi é",soma)
print("O valor exato é",pi)

RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exercicio1.py
A estimativa para pi é 3.1415926535714034
O valor exato é 3.141592653589793
>>>
```

Funções definidas pelo usuário

- Python permite que, além das funções intrínsecas e daquelas importadas a partir de pacotes, o usuário defina suas próprias funções, como no exemplo abaixo.



The screenshot shows a Python IDE window titled "temp.py - /tmp/temp.py (3.6.8)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
def fatorial(n):  
    f = 1  
    for k in range(1,n+1):  
        f *= k  
    return f  
  
print("0 fatorial de 5 é",fatorial(5))  
print("0 fatorial de 10 é",fatorial(10))
```

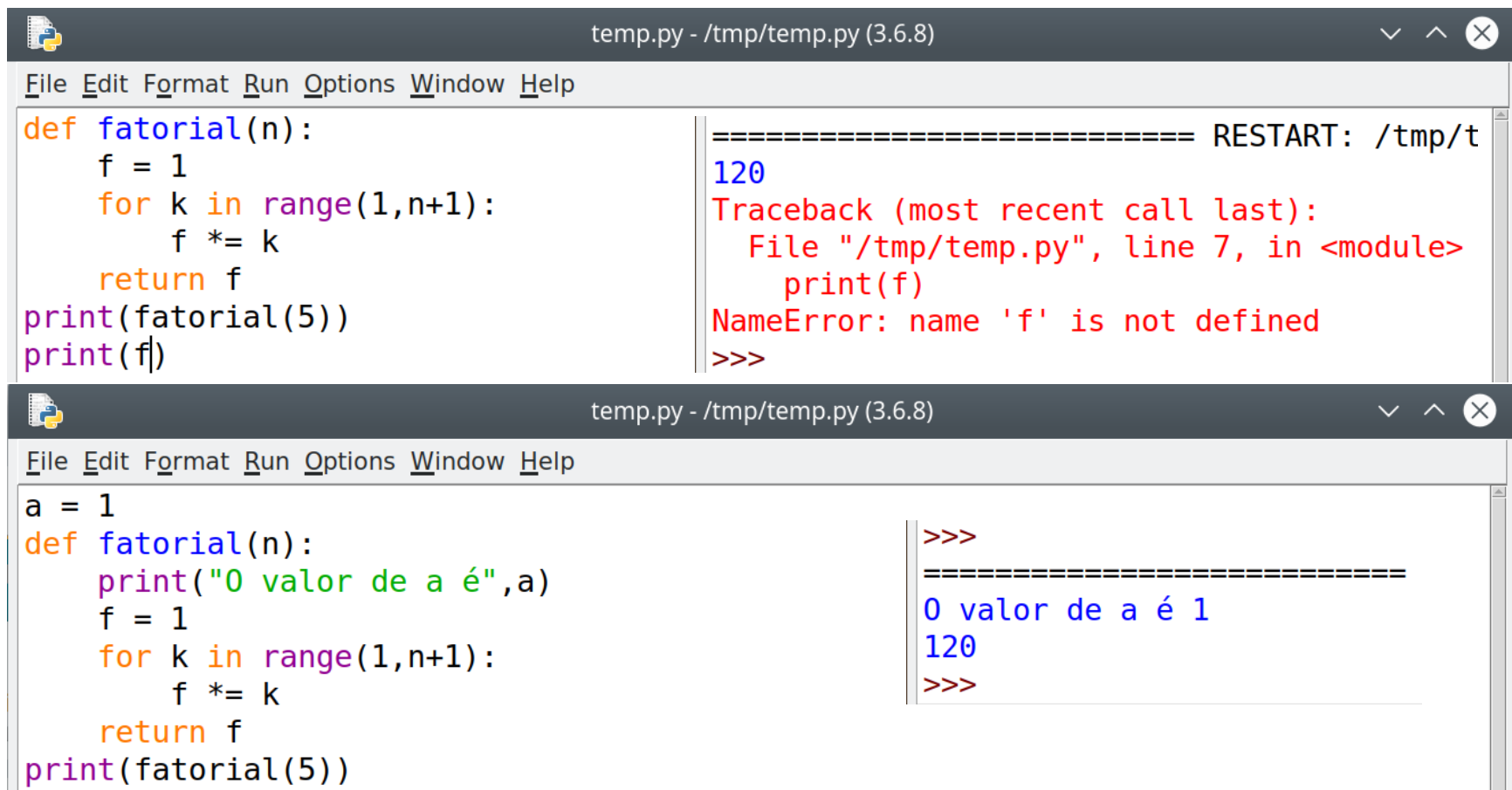
To the right of the code editor, a console window displays the output of the program:

```
=====  
0 fatorial de 5 é 120  
0 fatorial de 10 é 3628800  
>>>
```

O valor da função é o valor da variável que se segue à instrução `return` sob a instrução `def`. Note a sintaxe semelhante à de um laço.

Funções definidas pelo usuário

- As variáveis no interior da definição são independentes daquelas no corpo do programa, e não são “vistas” fora da definição. Mas o contrário não é verdadeiro.



The image shows two screenshots of a Python IDE window titled "temp.py - /tmp/temp.py (3.6.8)".

The top screenshot shows a function definition for `fatorial(n)` with a local variable `f`. The code is:

```
def fatorial(n):  
    f = 1  
    for k in range(1,n+1):  
        f *= k  
    return f  
print(fatorial(5))  
print(f)
```

The output on the right shows a `NameError: name 'f' is not defined`, indicating that the local variable `f` is not accessible outside the function.

The bottom screenshot shows the same function definition, but with a global variable `a` defined before the function. The code is:

```
a = 1  
def fatorial(n):  
    print("O valor de a é",a)  
    f = 1  
    for k in range(1,n+1):  
        f *= k  
    return f  
print(fatorial(5))
```

The output on the right shows the function being called, and the global variable `a` is printed as "O valor de a é 1", indicating that the function can access global variables.

Funções definidas pelo usuário

- Uma função definida pelo usuário aceita vários argumentos e pode retornar qualquer número de variáveis, ou outros objetos, como listas.

```
*temp.py - /tmp/temp.py (3.6.8)*
File Edit Format Run Options Window Help
from math import cos,sin,pi
def esf_cart(r,theta,phi):
    x = r*sin(theta)*cos(phi)
    y = r*sin(theta)*sin(phi)
    z = r*cos(theta)
    return x,y,z
x,y,z = esf_cart(2.0,pi/3,pi/4)
print(x,y,z)
```

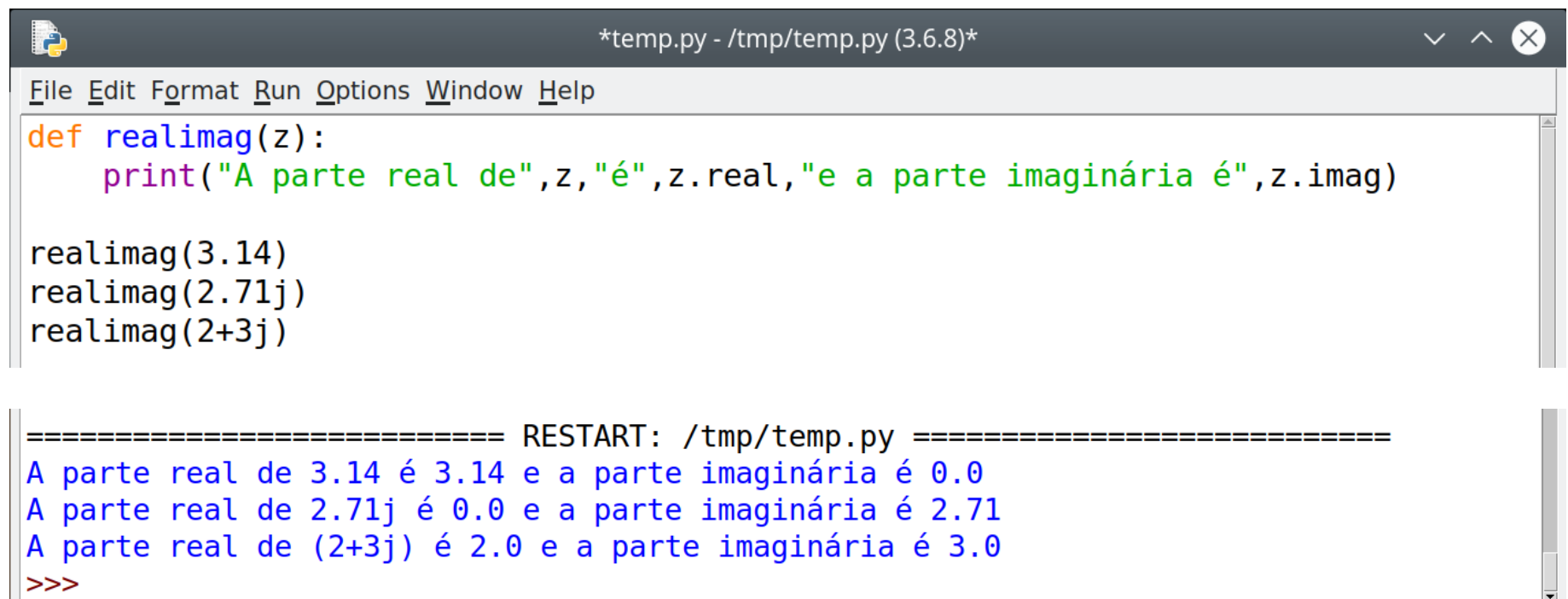
```
===== RESTART: /tmp/temp.py =====
1.2247448713915892 1.224744871391589 1.0000000000000002
>>>
```

```
temp.py - /tmp/temp.py (3.6.8)
File Edit Format Run Options Window Help
from math import cos,sin,pi
def esf_cart(r,theta,phi):
    x = r*sin(theta)*cos(phi)
    y = r*sin(theta)*sin(phi)
    z = r*cos(theta)
    return [x,y,z]
xyz = esf_cart(2.0,pi/4,pi/3)
print(xyz)
```

```
===== RESTART: /tmp/temp.py =====
[0.7071067811865476, 1.224744871391589, 1.4142135623730951]
>>>
```

Funções definidas pelo usuário

- Não é obrigatório que uma função retorne um valor. Veja no exemplo abaixo.



```
*temp.py - /tmp/temp.py (3.6.8)*
File Edit Format Run Options Window Help
def realimag(z):
    print("A parte real de",z,"é",z.real,"e a parte imaginária é",z.imag)

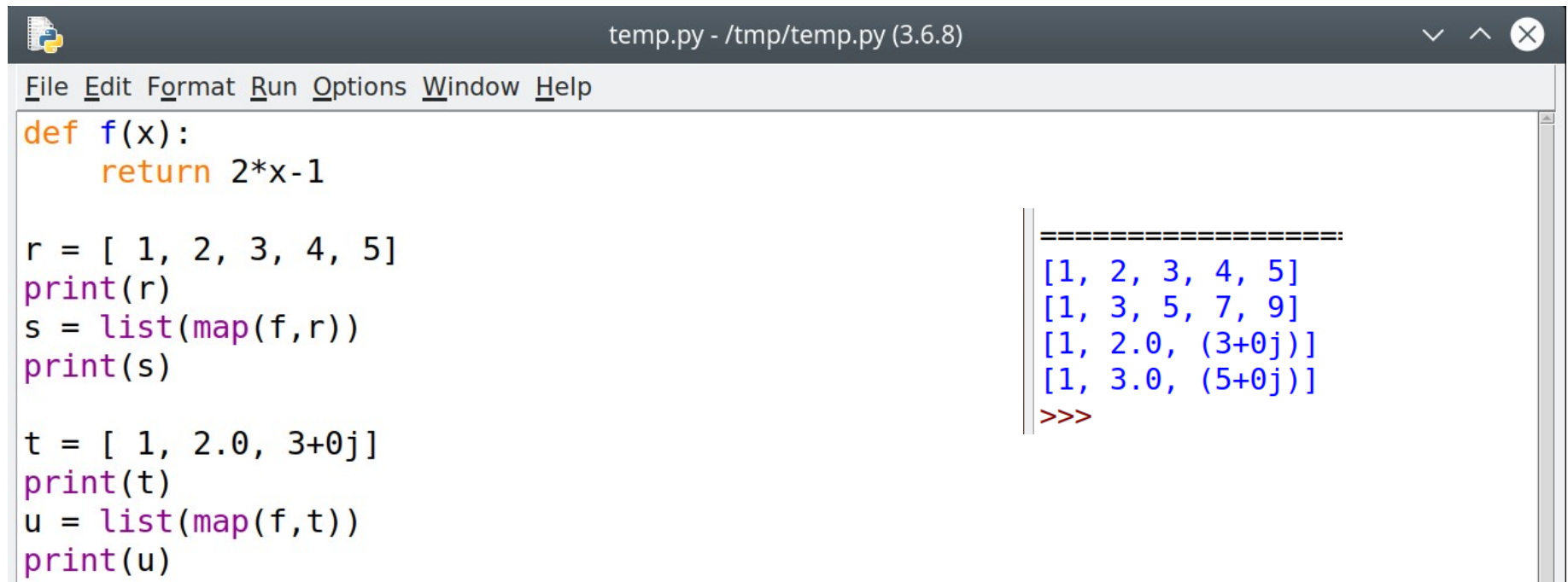
realimag(3.14)
realimag(2.71j)
realimag(2+3j)

===== RESTART: /tmp/temp.py =====
A parte real de 3.14 é 3.14 e a parte imaginária é 0.0
A parte real de 2.71j é 0.0 e a parte imaginária é 2.71
A parte real de (2+3j) é 2.0 e a parte imaginária é 3.0
>>>
```

Note a ausência da instrução `return` e o fato de que não é preciso invocar a função a partir de uma atribuição de variável.

Funções definidas pelo usuário

- Uma função definida pelo usuário pode ser utilizada juntamente com a instrução `map`.



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

def f(x):
    return 2*x-1

r = [ 1, 2, 3, 4, 5]
print(r)
s = list(map(f,r))
print(s)

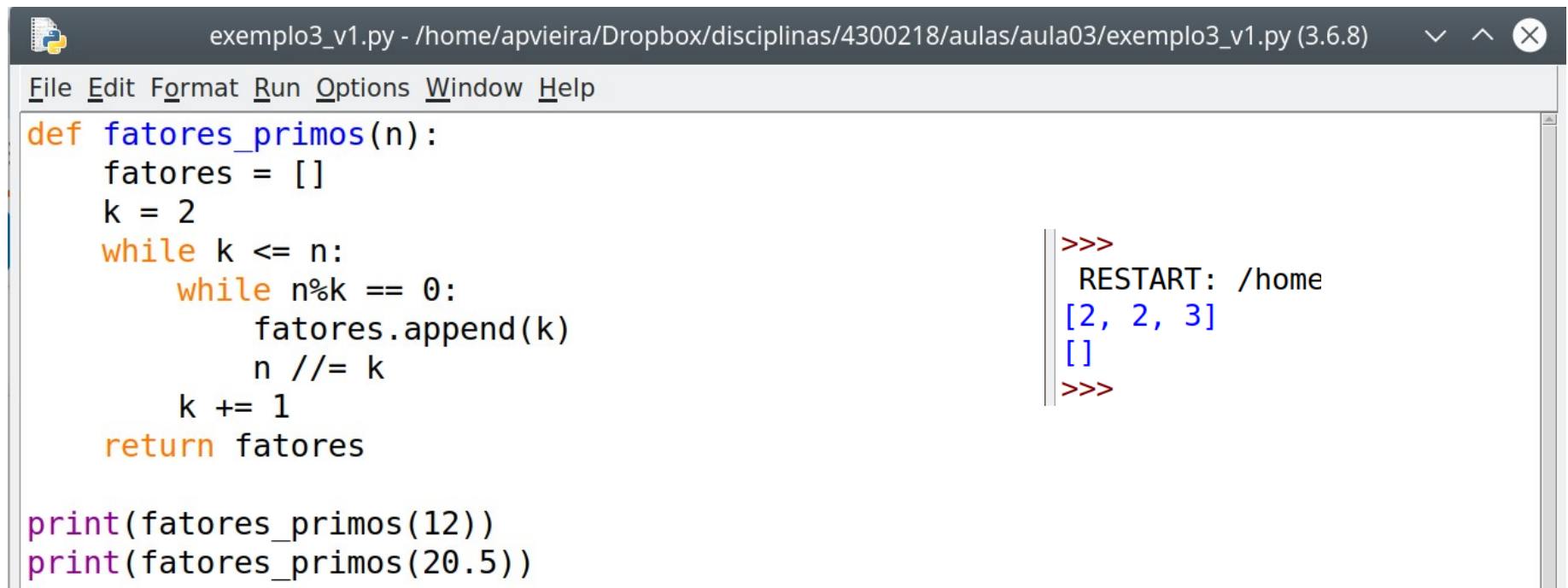
t = [ 1, 2.0, 3+0j]
print(t)
u = list(map(f,t))
print(u)
```

```
=====
[1, 2, 3, 4, 5]
[1, 3, 5, 7, 9]
[1, 2.0, (3+0j)]
[1, 3.0, (5+0j)]
>>>
```

Note que o tipo de uma variável interna reflete o tipo do argumento da função.

Funções definidas pelo usuário

- Exemplo 3: uma função para calcular os fatores primos de um número inteiro.



The screenshot shows a Python IDE window titled "exemplo3_v1.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo3_v1.py (3.6.8)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code editor contains the following Python code:

```
def fatores_primos(n):  
    fatores = []  
    k = 2  
    while k <= n:  
        while n%k == 0:  
            fatores.append(k)  
            n //= k  
        k += 1  
    return fatores  
  
print(fatores_primos(12))  
print(fatores_primos(20.5))
```

On the right side of the editor, the interactive Python shell (RESTART) shows the output of the code:

```
>>>  
RESTART: /home  
[2, 2, 3]  
[]  
>>>
```

Note que a função retorna uma lista vazia se o argumento não é inteiro.

Funções definidas pelo usuário

- Exemplo 3: uma função para calcular os fatores primos de um número inteiro.

```
exemplo3_v2.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo3_v2.py (3.6.8)
File Edit Format Run Options Window Help

def fatores_primos(n):
    if not isinstance(n,int):
        print("fatores_primos: convertendo o argumento para inteiro")
        n = int(n)
        print("fatores_primos: argumento convertido para",n)
    fatores = []
    k = 2
    while k <= n:
        while n%k == 0:
            fatores.append(k)
            n //= k
        k += 1
    return fatores

print(fatores_primos(12))
print(fatores_primos(20.5))
print(fatores_primos(20+1j))
```

```
[2, 2, 3]
fatores_primos: convertendo o argumento para inteiro
fatores_primos: argumento convertido para 20
[2, 2, 5]
fatores_primos: convertendo o argumento para inteiro
Traceback (most recent call last):
  File "/home/apvieira/Dropbox/disciplinas/4300218/aula03/exemplo3_v2.py", line 17, in <module>
    print(fatores_primos(20+1j))
  File "/home/apvieira/Dropbox/disciplinas/4300218/aula03/exemplo3_v2.py", line 4, in fatores_primos
    n = int(n)
TypeError: can't convert complex to int
>>>
```

Funções definidas pelo usuário

- Exemplo 3: uma função para calcular os fatores primos de um número inteiro.

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

def fatores_primos(n):
    if isinstance(n, complex):
        return "fatores_primos: não é possível fatorar um número complexo"
    if not isinstance(n, int):
        print("fatores_primos: convertendo o argumento para inteiro")
        n = int(n)
        print("fatores_primos: argumento convertido para", n)
    fatores = []
    k = 2
    while k <= n:
        while n%k == 0:
            fatores.append(k)
            n //= k
        k += 1
    return fatores

print(fatores_primos(12))
print(fatores_primos(20.5))
print(fatores_primos(20+1j))
```

```
===== RESTART: /tmp/temp.py =====
[2, 2, 3]
fatores_primos: convertendo o argumento para inteiro
fatores_primos: argumento convertido para 20
[2, 2, 5]
fatores_primos: não é possível fatorar um número complexo
>>>
```

Funções definidas pelo usuário

- Exemplo 3: uma função para calcular os fatores primos de um número inteiro.

```
exemplo3_v3.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo3_v3.py (3.6.8)
File Edit Format Run Options Window Help

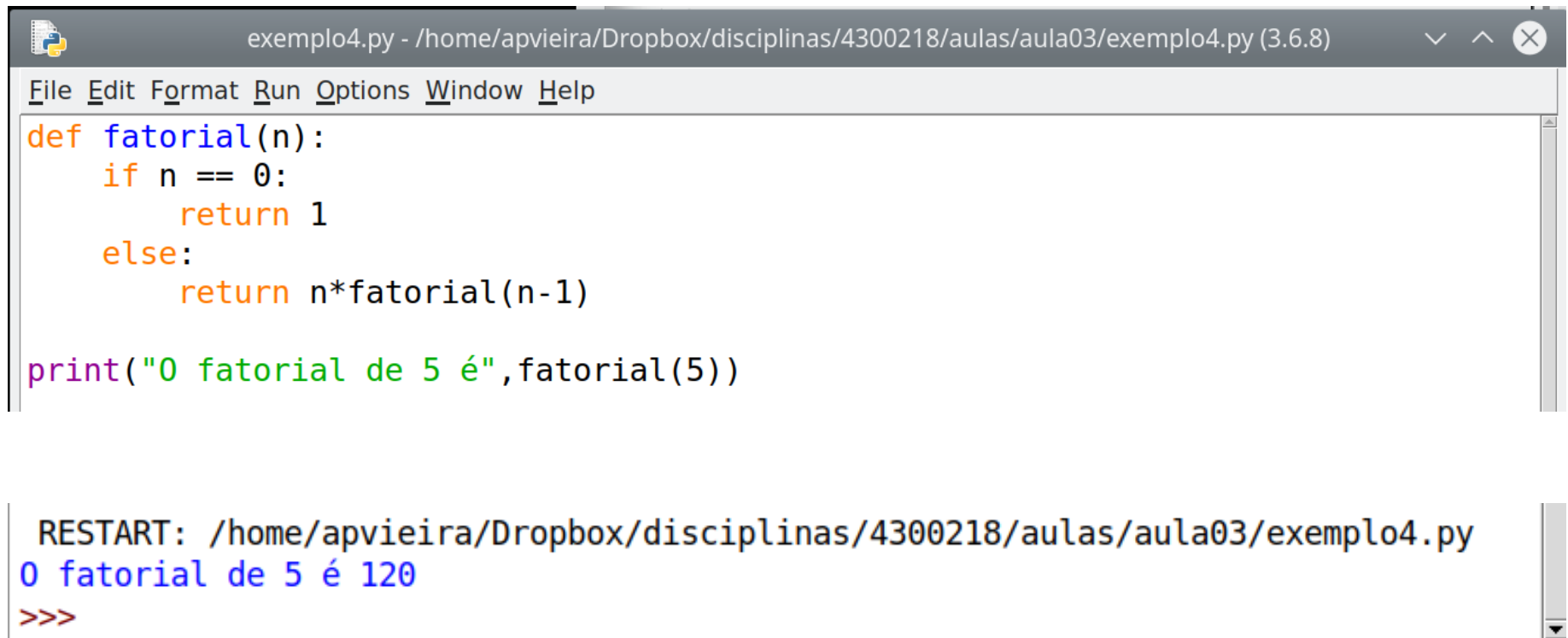
def fatores_primos(n):
    if isinstance(n, complex):
        return "fatores_primos: não é possível fatorar um número complexo"
    if not isinstance(n, int):
        print("fatores_primos: convertendo o argumento para inteiro")
        n = int(n)
        print("fatores_primos: argumento convertido para", n)
    fatores = []
    k = 2
    while k <= n:
        while n%k == 0:
            fatores.append(k)
            n //= k
        k += 1
    return fatores

print(fatores_primos(13))
for n in range(2, 50):
    if len(fatores_primos(n)) == 1:
        print(n)
```

RESTART: [13]
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
>>>

Funções definidas pelo usuário

- Exemplo 4: a função fatorial definida recursivamente.



```
exemplo4.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo4.py (3.6.8)
File Edit Format Run Options Window Help
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n*fatorial(n-1)

print("O fatorial de 5 é",fatorial(5))

RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exemplo4.py
O fatorial de 5 é 120
>>>
```

Python permite que uma função invoque a si própria. Nem todas as linguagens de programação têm essa facilidade.

Exercício 2

O coeficiente binomial é um inteiro igual a

$$C_{n,k} \equiv \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}$$

com a convenção de que $0!=1$.

- (a) Escreva um programa que defina uma função `binomial(n, k)` para calcular o coeficiente binomial dados n e k . Certifique-se de que o valor retornado seja inteiro (não real) e que retorne 1 quando $k=0$.
- (b) Use seu programa para imprimir as primeiras 10 linhas do triângulo de Pascal. A n -ésima linha do triângulo contém $n+1$ números, correspondentes aos coeficientes binomiais $C_{n,0}$ até $C_{n,n}$.
- (c) A probabilidade de obter cara k vezes lançando n vezes uma moeda honesta é $C_{n,k}/2^n$. Use sua função para calcular a probabilidade de obter cara 60 vezes ou mais ao lançar 100 vezes uma moeda honesta.

Exercício 2: solução

```
exercicio2.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exercicio2.py (3.6.8)
File Edit Format Run Options Window
def binomial(n,k):
    if k == 0:
        return 1
    else:
        num, den = n, 1
        for m in range(1,k):
            num *= (n-m)
            den *= m+1
        return num//den

for n in range(1,11):
    linha = []
    for k in range(n+1):
        linha.append(binomial(n,k))
    print(linha)

def probcaras(n,mink,maxk):
    prob = 0.0
    for k in range(mink,maxk+1):
        prob += binomial(n,k)
    return prob/2**n

print("A probabilidade de obter 60 caras ou mais é",probcaras(100,60,100))
print("A probabilidade de obter 100 caras é",probcaras(100,100,100))
print("A probabilidade de obter 59 caras ou menos é",probcaras(100,0,59))

RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]
A probabilidade de obter 60 caras ou mais é 0.028443966820490392
A probabilidade de obter 100 caras é 7.888609052210118e-31
A probabilidade de obter 59 caras ou menos é 0.9715560331795097
>>> |
```

Exercício 2: solução recursiva

exercicio2.py - /home/apvieira/Dropbox/disc

File Edit Format Run Options Window Help

```
def binomial(n,k):  
    if k == 0:  
        return 1  
    else:  
        return binomial(n-1,k-1)*n//k
```

```
for n in range(10):  
    linha = []  
    for k in range(n+1):  
        linha.append(binomial(n,k))  
    print(linha)
```

```
def probcaras(n,mink,maxk):  
    prob = 0.0  
    for k in range (mink,maxk+1):  
        prob += binomial(n,k)  
    return prob/2**n
```

```
print("A probabilidade de obter 60 caras ou mais é",probcaras(100,60,100))  
print("A probabilidade de obter 100 caras é",probcaras(100,100,100))  
print("A probabilidade de obter 59 caras ou menos é",probcaras(100,0,59))
```

RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03

```
[1, 1]  
[1, 2, 1]  
[1, 3, 3, 1]  
[1, 4, 6, 4, 1]  
[1, 5, 10, 10, 5, 1]  
[1, 6, 15, 20, 15, 6, 1]  
[1, 7, 21, 35, 35, 21, 7, 1]  
[1, 8, 28, 56, 70, 56, 28, 8, 1]  
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]  
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]  
A probabilidade de obter 60 caras ou mais é 0.028443966820490392  
A probabilidade de obter 100 caras é 7.888609052210118e-31  
A probabilidade de obter 59 caras ou menos é 0.9715560331795097  
>>> |
```

Exercício 3

Euclides mostrou que o máximo divisor comum de dois inteiros não negativos m e n satisfaz

$$g(m, n) = \begin{cases} m & \text{se } n = 0 \\ g(n, m \bmod n) & \text{se } n > 0 \end{cases}$$

Escreva um programa que utilize recursão para calcular e imprimir o máximo divisor comum de m e n utilizando o algoritmo de Euclides. Teste sua função com os números 108 e 192.

Exercício 3: solução

$$g(m,n)=\begin{cases} m & \text{se } n=0 \\ g(n, m \bmod n) & \text{se } n>0 \end{cases}$$

```
exercicio3.py - /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exercicio3.py (3.6.8)
File Edit Format Run Options Window Help

def maxdivcomum(m,n):
    if m<0 or n<0:
        return "Números devem ser não negativos"
    if n==0:
        return m
    else:
        return maxdivcomum(n,m%n)

print(maxdivcomum(108,192))

RESTART: /home/apvieira/Dropbox/disciplinas/4300218/aulas/aula03/exercicio3.py
12
>>> |
```

Bom estilo de programação

- Ao escrever um programa, procure satisfazer os três seguintes requisitos:
 - o programa deve ter uma **estrutura simples**;
 - deve ser **fácil de ler e entender**;
 - sua **velocidade de execução** deve ser a **maior possível**.
- Para ajudá-lo a satisfazer os requisitos, tenha em mente as observações dos próximos slides.

Bom estilo de programação

- Utilize **comentários** (#) abundantemente, para registrar seu raciocínio ao elaborar o programa e ajudá-lo a compreender o programa no futuro.
- Dê **nomes relevantes às variáveis**, que lembrem seu papel, como `E` ou `energia`, `t` ou `tempo` e `B` ou `campo_magnetico`.
- Utilize os **tipos apropriados de variáveis**. Tratar uma variável inteira (`int`) como real (`float`), por exemplo, torna seu programa mais lento.

Bom estilo de programação

- **Importe as funções no início do programa.** Isso facilita a depuração (*debugging*).
- **Dê nomes às suas constantes** e a combinações que se repitam em laços. Isso facilita a leitura e a depuração e pode acelerar seu programa, evitando o cálculo repetido de funções, por exemplo.
- Se um trecho de código é utilizado várias vezes, utilize-o para **definir uma função**.
- **Imprima resultados parciais** para acompanhar o andamento do programa.

Bom estilo de programação

- **Inclua espaços e divida linhas** (usando a barra invertida, `\`) para deixar seu programa mais legível.

```
energia = massa * (vx**2 + vy**2) / 2 + massa*g*y \  
          + momento_de_inercia * omega**2 / 2
```

Para a próxima aula

- Primeiro trabalho para nota.