

# Python Científico

**Carlos Rodrigues Rocha**

`carlos.rocha@riogrande.ifrs.edu.br`

- Conceitos e motivações
- Usando o Spyder como IDE base
- Dados em listas e em arrays numpy
- Gráficos com Matplotlib
- *Curve fitting* usando polinômios
- Resolução de ODEs
- O que mais é possível fazer?

# Conceitos e motivações

- O Problema
  - Resolução de problemas analíticos e numéricos em exatas
  - Modelagem e simulação
  - Análise de dados (quantitativa e qualitativa)
  - Visualização e apresentação de resultados
- As Possíveis Soluções
  - Planilhas eletrônicas
  - Desenvolvimento de software
  - *Computer Algebra Systems* (CAS)
  - Modeladores/simuladores dedicados

# Conceitos e motivações

- Todas as soluções são válidas
- O uso depende do tipo de problema a resolver
- O volume de dados também é relevante
- Custo da solução é um fator importante
- Empregar dois ou mais tipos de solução é usual
- *Pipelining* é uma técnica comum
- Coexecução é mais complexa, mas muito útil
- Ligação/comunicação entre softwares

# Conceitos e motivações

- Python

- Linguagem de programação de alto nível **interpretada**
- Estruturada, modular e orientada a objetos
- Tipagem dinâmica e forte
- Criada por Guido van Rossum em 1990

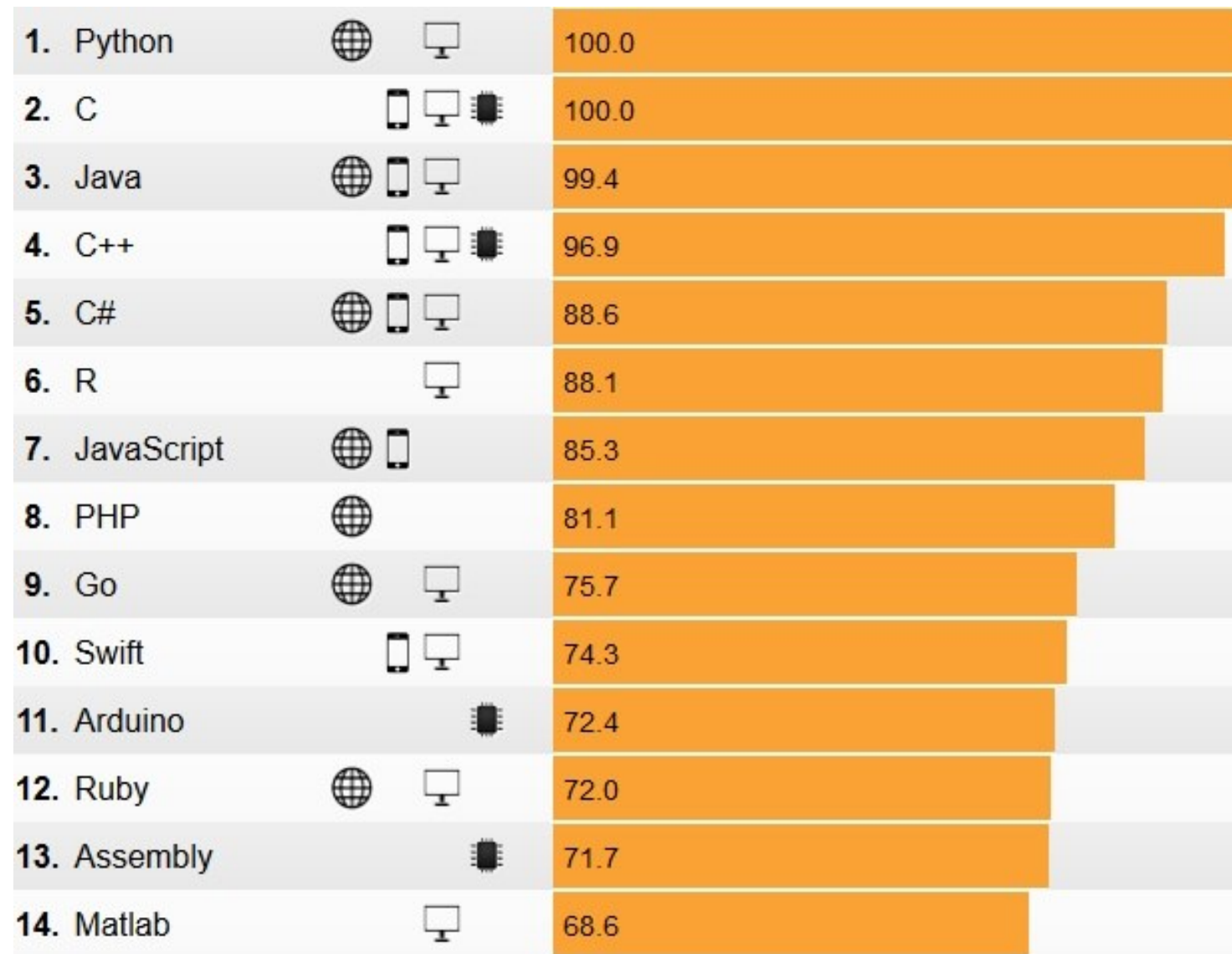


- Características

- Uso geral
- Multiplataforma
- Grande base de recursos *built-in* e extensa oferta de bibliotecas
- Comunidade de usuários e desenvolvedores imensa
- *Flerte* com a comunidade *open source* e software livre

# Conceitos e motivações

- Ranking das Linguagens de Programação



Fonte: IEEE Spectrum(2017)

# Conceitos e motivações



- Onde e como obter?
  - Direto do site [www.python.org](http://www.python.org)
  - Obtendo uma *distribuição*:
    - Python-xy ([python-xy.github.io](http://python-xy.github.io))
    - WinPython ([winpython.github.io](http://winpython.github.io))
    - Enthought Canopy ([store.enthought.com/downloads](http://store.enthought.com/downloads))
  - Ou, se é um iluminado Linux, é só selecionar os pacotes
- APIs
  - Várias são disponíveis no próprio Python
  - APIs de terceiros disponíveis de acordo com a área
  - Python Package Index (PyPI)
- Ambientes de Desenvolvimento (IDE)
  - Além do editor (básico), ferramentas de apoio integradas a ele
  - Diversas IDEs, independentes do Python

# Algumas APIs

- APIs científicas





# Algunas APIs

- APIs gráficas



PyQt, PySide



PyGTK



wxPython



kivy

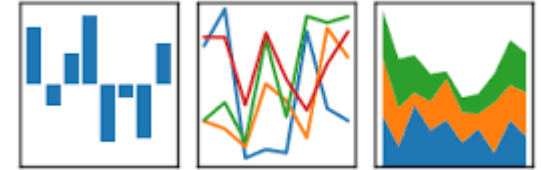
# Algumas APIs

- Diversas



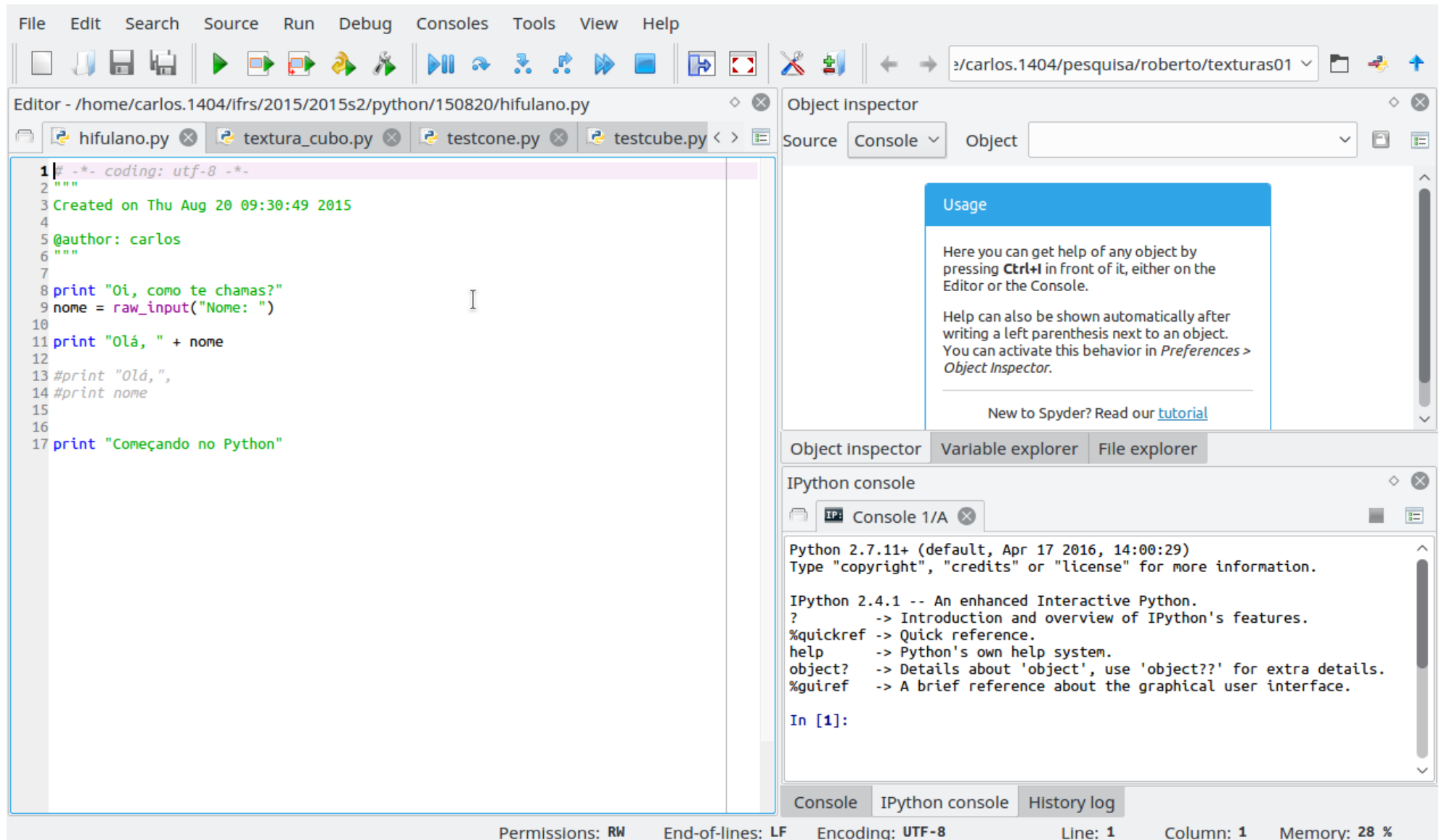
pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

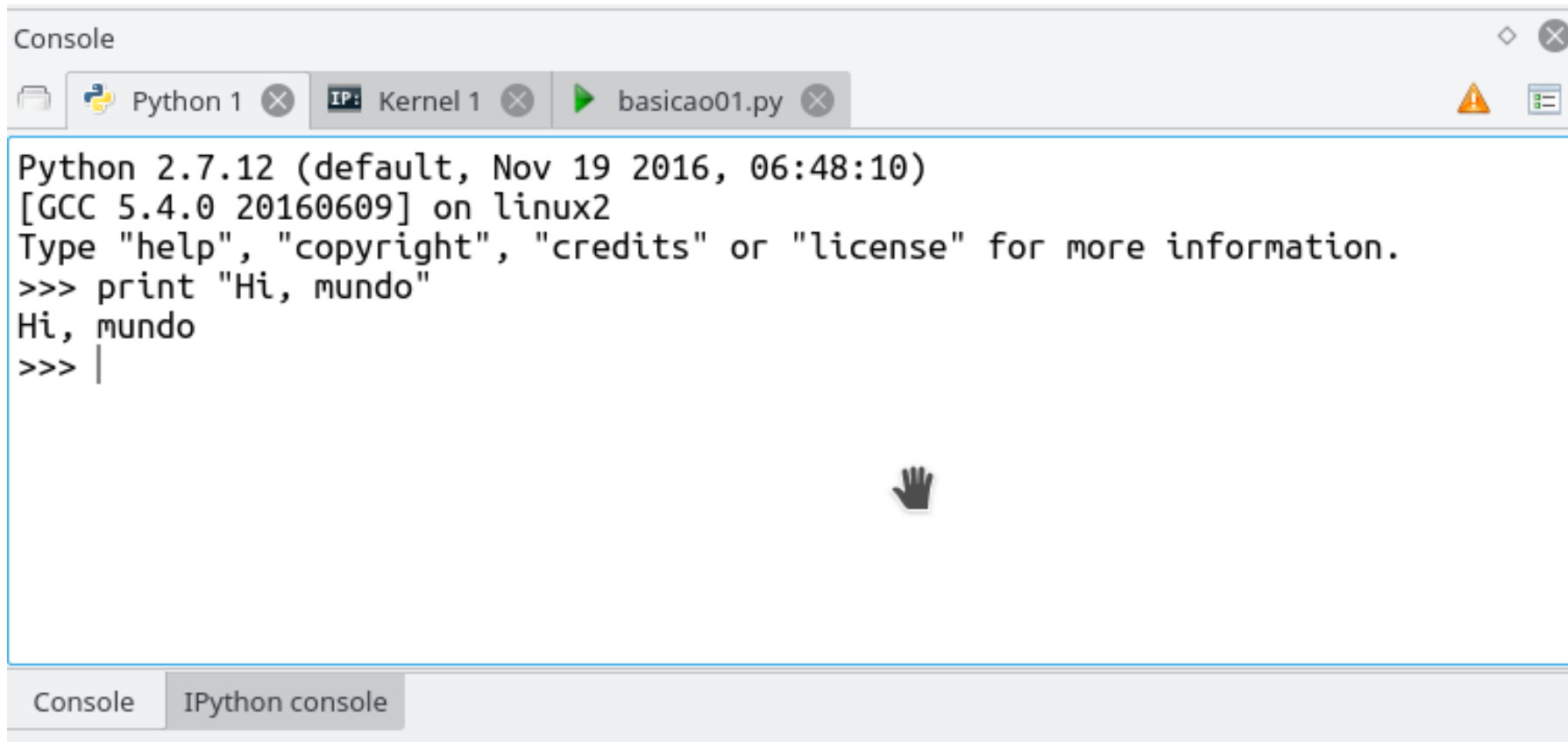


# Conhecendo a IDE

- Spyder2: IDE para Python 2.x que lembra o MATLAB



# Conhecendo a IDE



The screenshot shows a JupyterLab interface with a 'Console' window. The window has a title bar with 'Console' and standard window controls. Below the title bar is a tab bar with three tabs: 'Python 1', 'IP: Kernel 1', and 'basicao01.py'. The 'IP: Kernel 1' tab is active. The console area displays the following text:

```
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hi, mundo"
Hi, mundo
>>> |
```

A mouse cursor is visible over the console area. At the bottom of the window, there is a tab bar with 'Console' and 'IPython console' tabs, with 'IPython console' being the active tab.

# Conhecendo a IDE

The screenshot displays the JupyterLab IDE interface. The main editor window on the left shows a Python script named `basicao01.py` with the following content:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Jun  2 10:17:17 2016
4
5 @author: carlos
6 """
7
8 a = input(u"Entre com o dividendo: ")
9 b = input(u"Entre com o divisor: ")
10
11 print u"A divisão de a por b é " + str(a/b)
```

The right-hand side of the interface contains the Object Inspector, Variable Explorer, and File Explorer panels. The Object Inspector is currently active, displaying a "Usage" message:

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Object Inspector*.

Below the Object Inspector is the Console, which shows the output of the script:

```
Entre com o dividendo: 5
Entre com o divisor: 2
A divisão de a por b é 2
>>> a
5
>>> b
2
>>> |
```

The bottom status bar indicates the current file is `basicao01.py`, the encoding is UTF-8, and the cursor is at Line 11, Column 1. The memory usage is 13 %.

# Usando a IDE

Exemplos 01 a 04

# Dados & Operadores - Numéricos

- `int`: `int(obj, base)` – long C

10, 010, 0x10, -10, -010

- `long`: `long(obj, base)` – ilimitado

10L, 0x100L

- `float`: `float(obj)` – double C

10.0, .5

- `complex`: `complex(re, im)`

10j, 1+.5j

+

-

\*

/ % //

`abs(nro)`

`pow(nro, exp) **`

`nro.conjugate()`

`round(nro, casas)`

|

&

^

<<

>>

~

# Dados & Operadores

- Lógicos

False: 0, .0, None, '', (), [], {}

True: tudo que não False

and  
not or

< <=  
== !=

> >=  
is is not

in not in

- Textuais: str(obj)

'exemplo 1'

"exemplo 2"

"""

Exemplo 3

"""

u"unicode str"

in not in

+ \*

[i] [i:j] [i:j:k]

len(s)

min(s) max(s)

s.index(c)

s.count(c)



# Dados e Operadores - Sequências

- Tuplas: imutáveis

`(1,2,3)`    `('a','b')`  
`('ncc',1701)`

```
l[i]= x  
l[i:j]= t
```

```
l.append(t)  
l.insert(t,i)
```

- Listas: mutáveis

`[1,2,3]`        `['a','b']`  
`['ncc',1701]`

```
del s[i:j]  
l.remove(x)
```

```
l.index(x)  
l.count(x)  
l.reverse()  
l.sort()
```

- Dicionários: mutáveis

`{ 'Enterprise': 'nx-01', 'Columbia': 'nx-02' }`  
`{ 10: 'Rose', 11: 'Amy', 12: 'Clara' }`

# Listas x arrays Numpy

- Listas são úteis para agrupamentos
- Muitas APIs utilizam listas e tuplas
- Porém, a manipulação aritmética é limitada
- Numpy: API para manipulação numérica
- Numpy array: matrizes multidimensionais
- Numpy matrix: matrizes bidimensionais
- Operadores array-wise
- Métodos de criação a partir de listas

# Listas x Arrays Numpy – exemplo

## Usando listas

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

```
b=[3,4,5]
```

```
c=a+b
```

```
c=[0,1,2,3,4,5]
```

## Usando arrays numpy

```
import numpy as np
```

```
a=np.array([0,1,2])
```

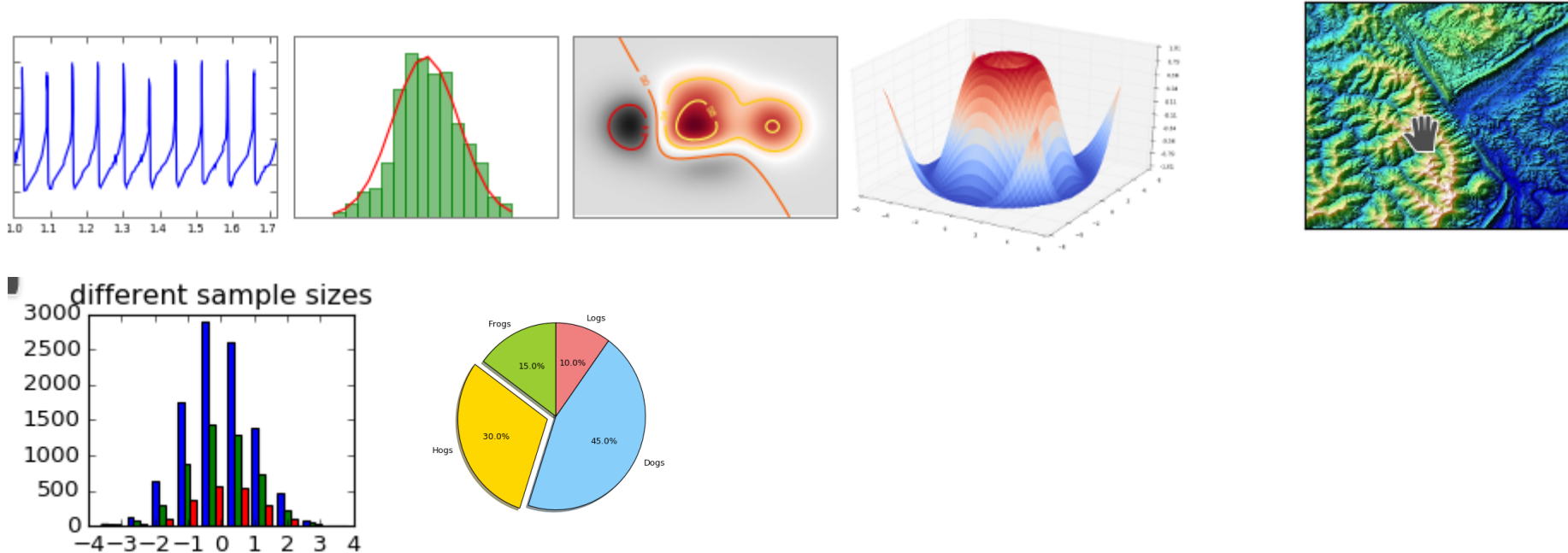
```
b=np.array([3,4,5])
```

```
c=a+b
```

```
c=array([3,5,7])
```

# Criando gráficos com Matplotlib

- Matplotlib foi criada por John Hunter
- Baseia-se em parte na simplicidade do Matlab para gráficos
- Suporta diferentes tipos de gráficos



- Recursos integradores com APIs gráficas
- Exporta para diferentes formatos gráficos

# Gráficos com Matplotlib

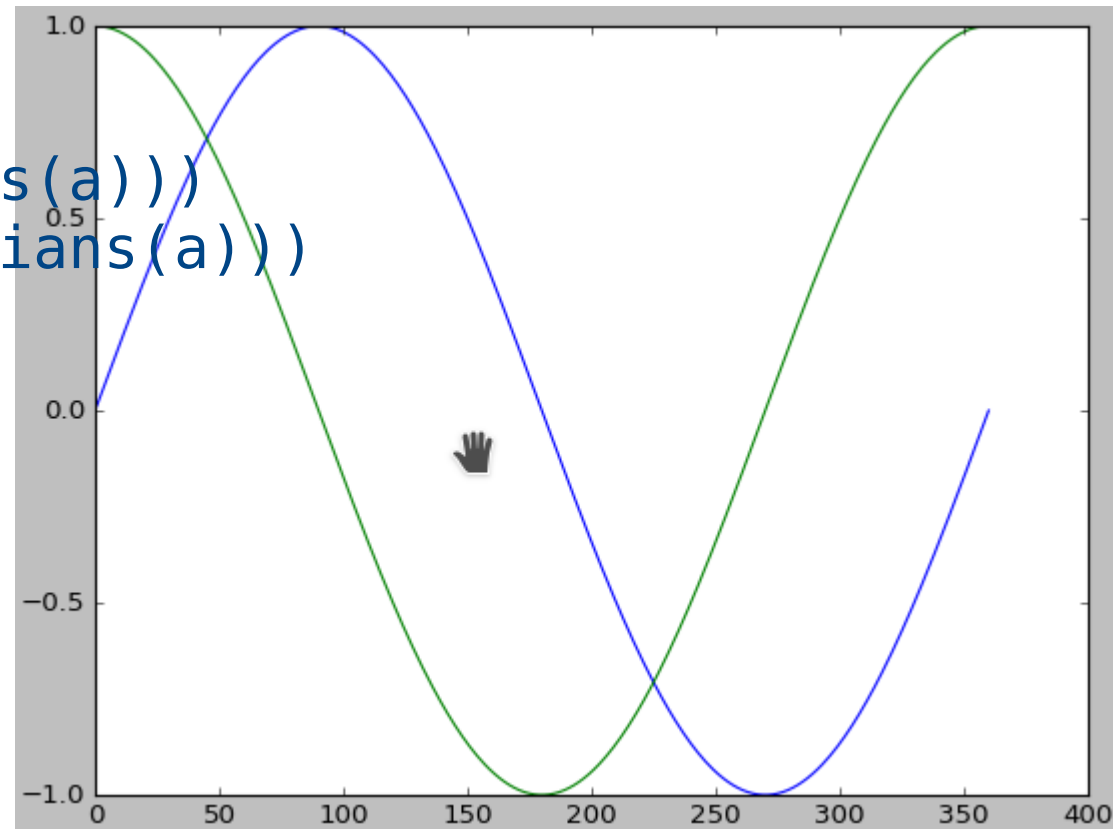
## Usando listas

```
from math import sin, cos, radians
from pylab import *
```

```
ang = range(0,361)
seno = []
cosseno = []
```

```
for a in ang:
    seno.append(sin(radians(a)))
    cosseno.append(cos(radians(a)))
```

```
plot (ang, seno)
plot (ang, cosseno)
```



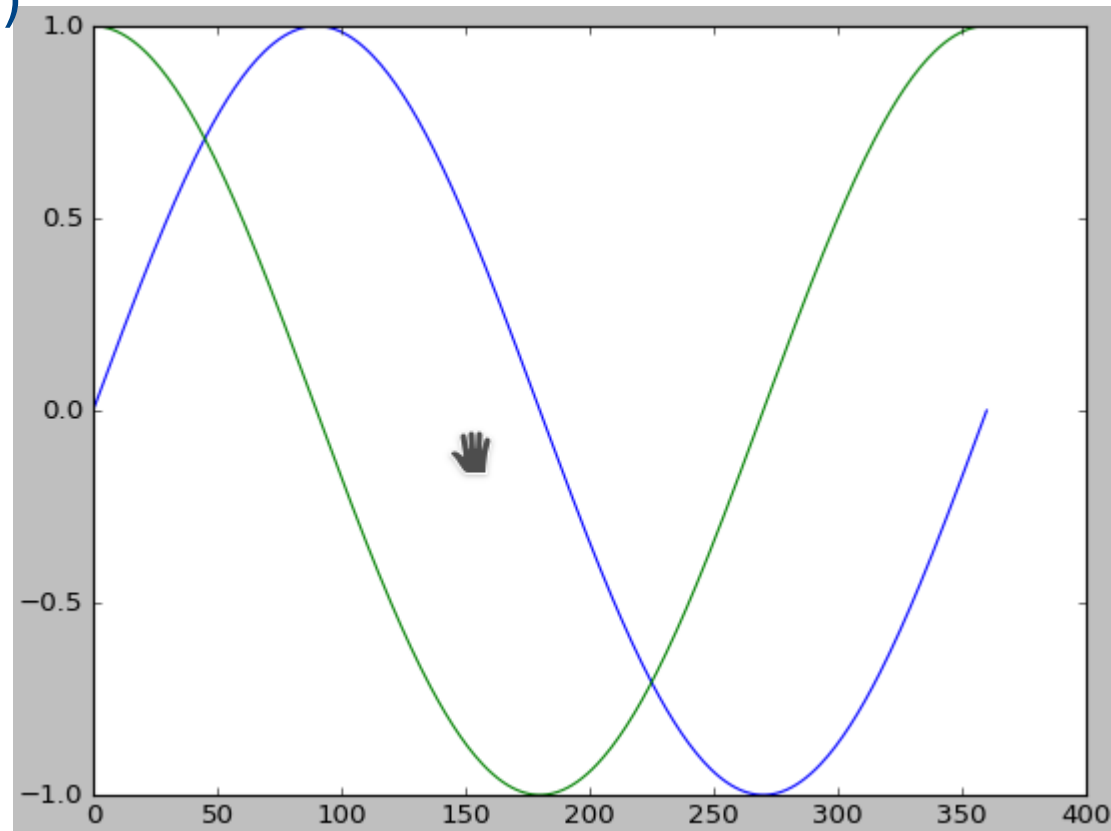
# Gráficos com Matplotlib

## Usando numpy

```
from numpy import sin, cos, radians, arange  
from pylab import *
```

```
ang = arange (0.0, 361.0)  
seno    = sin(radians(ang))  
cosseno = cos(radians(ang))
```

```
plot (ang, seno)  
plot (ang, cosseno)
```



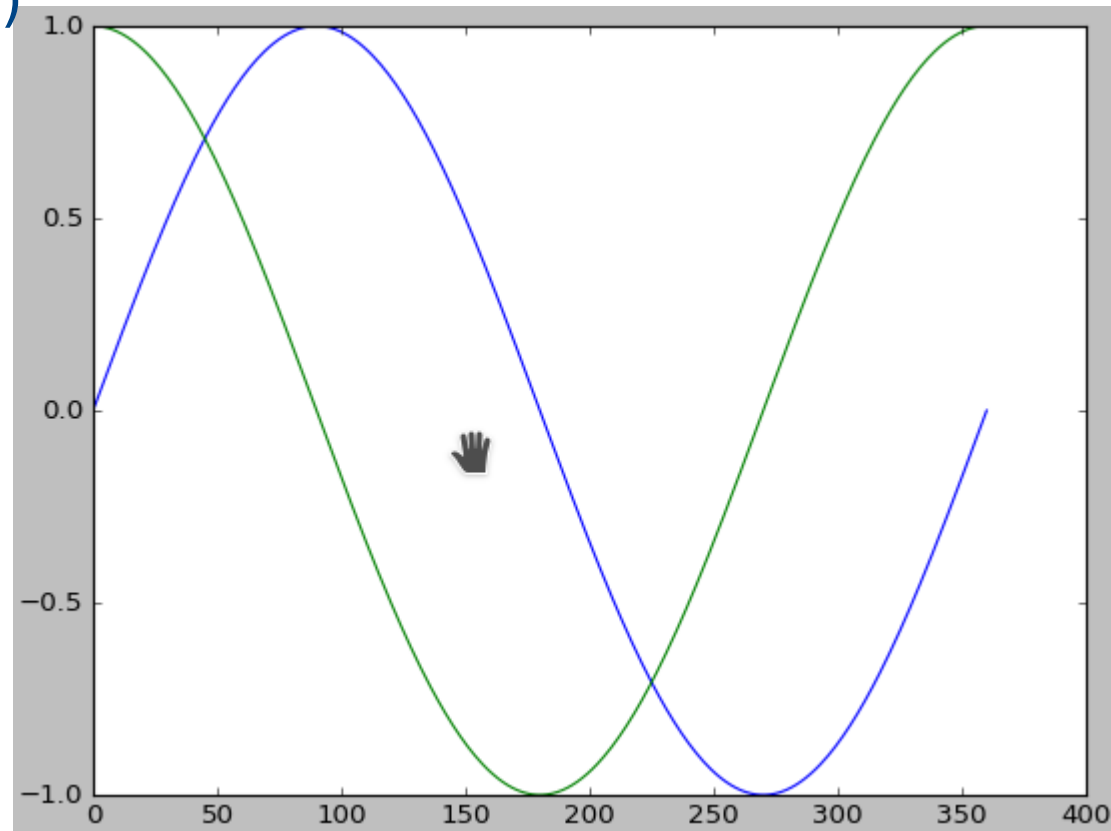
# Gráficos com Matplotlib

## Ainda usando numpy

```
from numpy import sin, cos, radians, arange  
from pylab import *
```

```
ang = arange (0.0, 361.0)  
seno    = sin(radians(ang))  
cosseno = cos(radians(ang))
```

```
plot (ang, seno)  
plot (ang, cosseno)
```



# Gráficos com Matplotlib

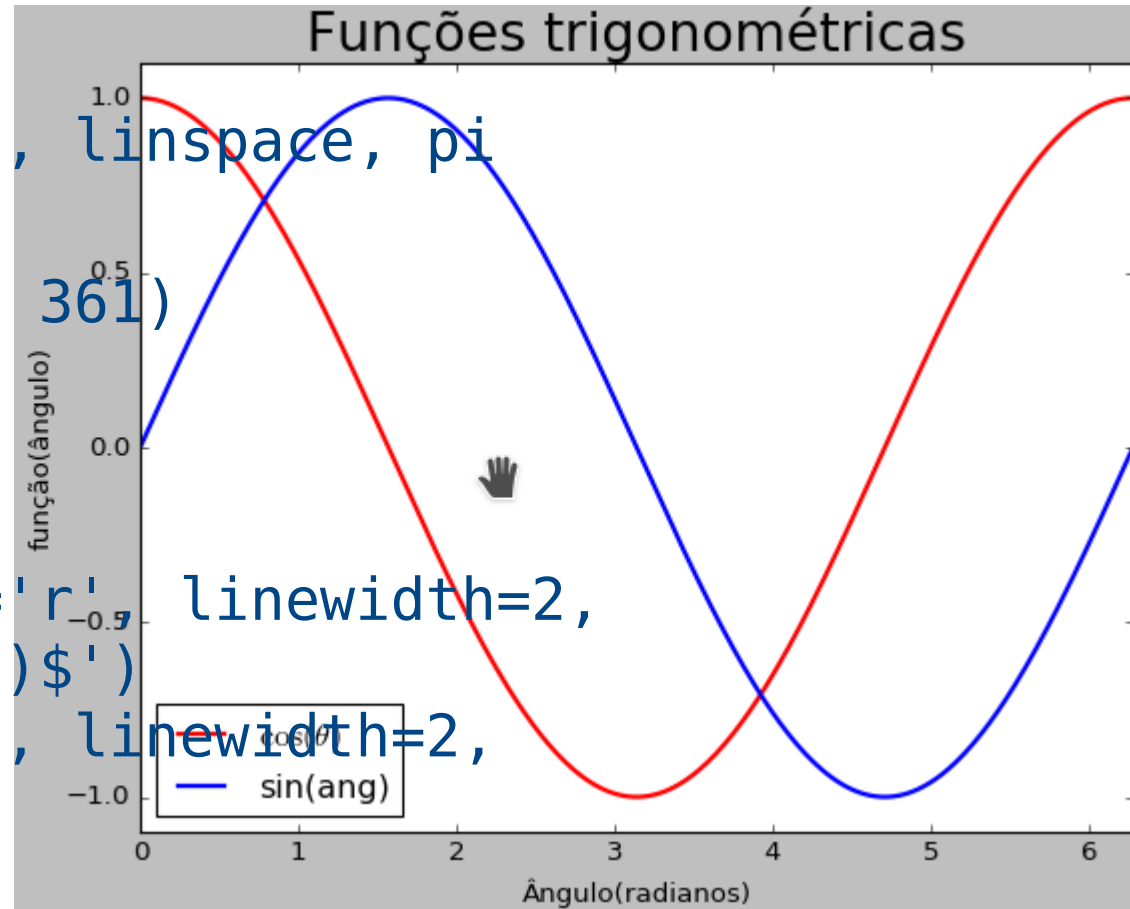
## Ainda usando numpy

```
from numpy import sin, cos, linspace, pi
from pylab import *

ang = linspace (0.0, 2*pi, 361)
seno      = sin(ang)
cosseno   = cos(ang)

cla()
plot (ang, cosseno, color='r', linewidth=2,
      label=r'$\cos(\theta)$')
plot (ang, seno, color='b', linewidth=2,
      label='sin(ang)')

ylim (-1.1,1.1)
xlim (0,2*pi)
title(u"Funções trigonométricas", fontsize=24)
xlabel(u"Ângulo(radianos)")
ylabel(u"função(ângulo)")
legend(loc=3)
```





# Gráficos com Matplotlib

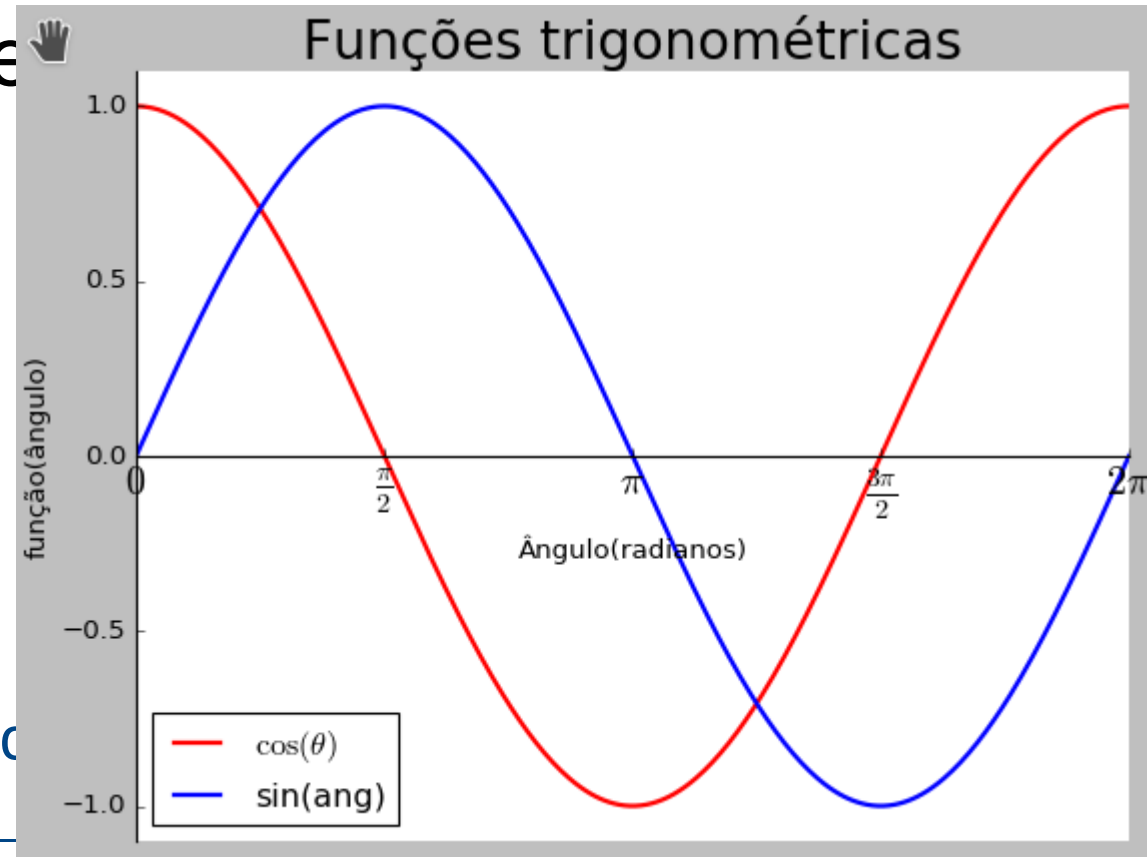
## Numpy e matplotlib oste

```
ang = linspace (0.0, 2*pi, 361)
seno = sin(ang)
cosseno = cos(ang)

cla()
plot (ang, cosseno, color='r', linewidth=2,
      label=r'$\cos(\theta)$')
plot (ang, seno, color='b', linewidth=2,
      label='sin(ang)')

ylim (-1.1,1.1)
xlim (0,2*pi)
title(u"Funções trigonométricas", fontsize=24)
xlabel(u"Ângulo(radianos)")
ylabel(u"função(ângulo)")
legend(loc=3)
```

```
eixos = pl.gca()
eixos.spines['top'].set_color('black')
eixos.spines['right'].set_color('black')
eixos.yaxis.set_ticks_position('left')
eixos.spines['bottom'].set_position (('data',0))
eixos.xaxis.set_ticks_position('bottom')
pl.xticks([0,np.pi/2,np.pi,3*np.pi/2,2*np.pi],
          [r'$0$',r'$\frac{\pi}{2}$', r'$\pi$',
           r'$\frac{3}{2} \pi$',r'$2 \pi$'],
          fontsize=18)
```



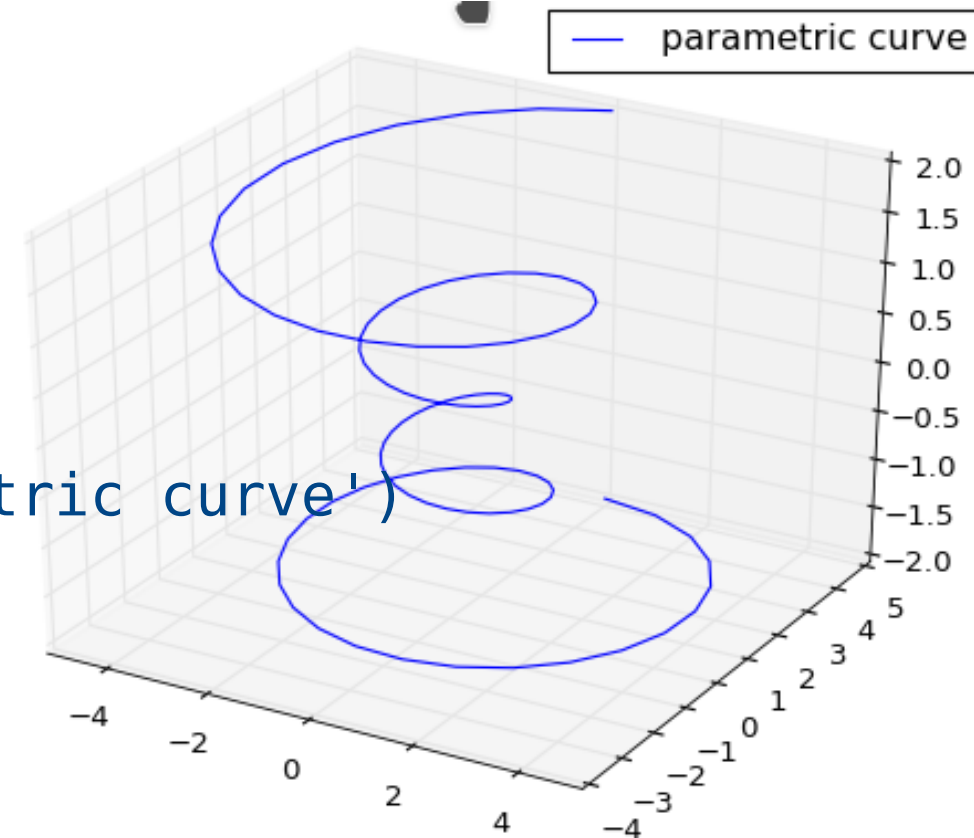
# Gráficos com Matplotlib

## Gráficos 3D

```
from pylab import figure, show
from numpy import pi, linspace, sin, cos
from mpl_toolkits.mplot3d import Axes3D
```

```
theta = linspace(-4*pi, 4*pi, 100)
z = linspace(-2, 2, 100)
r = z**2 + 1
x = r*sin(theta)
y = r*cos(theta)
r = None
```

```
fig = figure()
ax = fig.gca(projection='3d')
ax.plot(x, y, z, label='parametric curve')
ax.legend()
show()
```



# Gráficos com Matplotlib

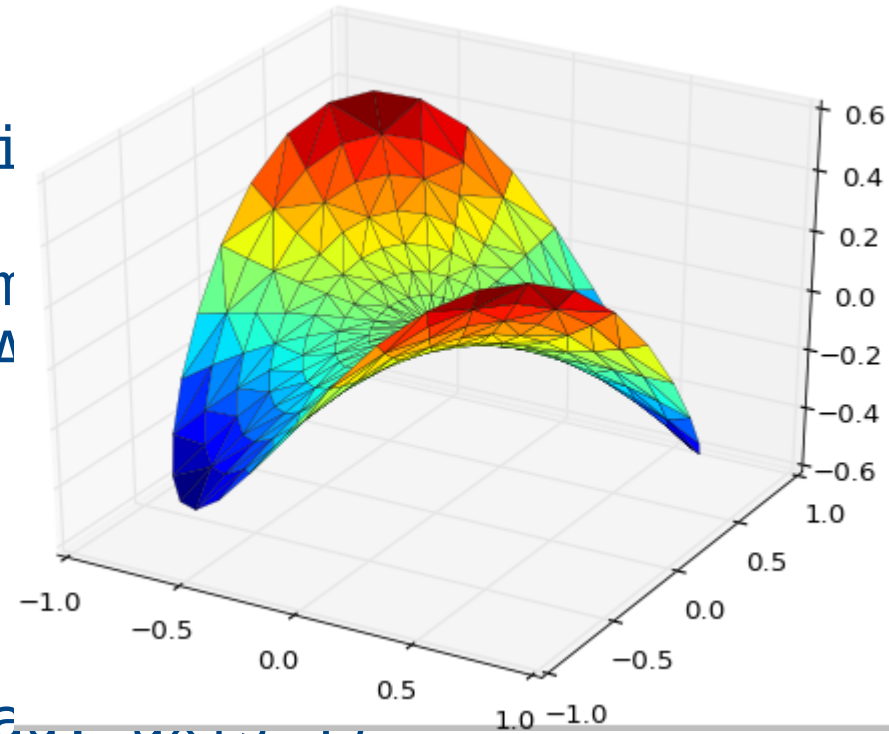
## Mais Gráficos 3D

```
from numpy import pi, sin, cos, linspace, repeat, append,
from pylab import figure, show, cm
from mpl_toolkits.mplot3d import Axes3D

nAng = 36
nRad = 8
rad = linspace(0.125, 1.0, nRad)
ang = linspace(0, 2*pi, nAng)
ang = repeat(ang[...], nRad, axis=-1)

x = append(0, (rad*cos(ang)).flatten())
y = append(0, (rad*sin(ang)).flatten())
z = sin(-x*y)

fig = figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(x, y, z, cmap=cm.jet, linewidth=0.2)
show()
```



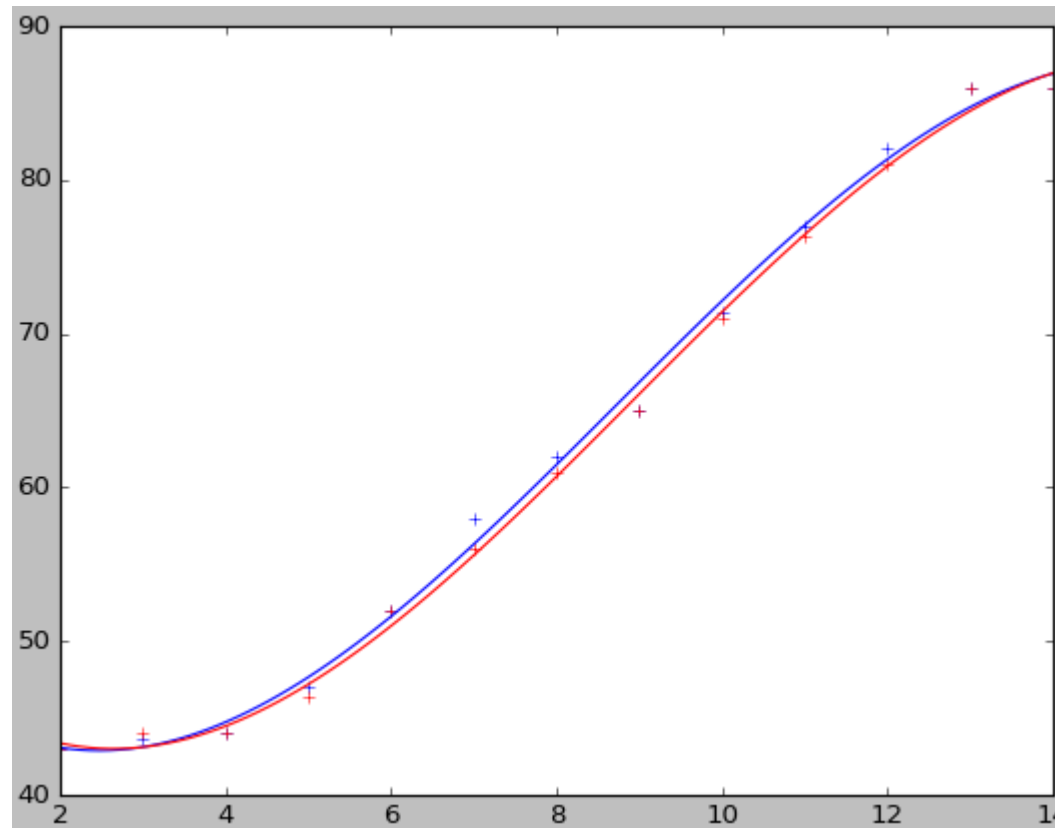
# Curve fitting

- Problema
  - Construção de um sensor de interface água-óleo
  - Sinal analógico 0V-5V, lido por um Arduino (0-1023)
  - 3 ciclos de carga e descarga de água em leitura
  - Média de carga e descarga



# Ajuste de curvas

- Identificação dos pontos
  - Array de valores de nível, média de subidas, média de descidas
  - Plotagem dos pontos
  - Uso da função `polyfit()` para estimar coeficientes
  - Uso da função `poly1d()` para criar um polinômio



# Ajuste de curvas

```
from numpy import array, linspace, polyfit, poly1d
from pylab import plot
```

```
nivel = array([2,3,4,5,6,7,8,9,10,11,12,13,14],dtype=float)
l1s   = array([43,44,44,47,52,58,62,65,71,77,82,86,86],dtype=float)
l1d   = array([43,44,44,47,52,56,61,65,71,77,81,86,86],dtype=float)
l2s   = array([43,44,44,47,52,58,62,65,72,77,82,86,86],dtype=float)
l2d   = array([43,44,44,46,52,56,61,65,71,77,81,86,86],dtype=float)
l3s   = array([43,43,44,47,52,58,62,65,71,77,82,86,86],dtype=float)
l3d   = array([43,44,44,46,52,56,61,65,71,75,81,86,86],dtype=float)
```

```
lsMed = (l1s+l2s+l3s)/3
```

```
ldMed = (l1d+l2d+l3d)/3
```

```
plot(nivel, lsMed, 'b+', linewidth=2)
```

```
plot(nivel, ldMed, 'r+', linewidth=2)
```

# Ajuste de curvas

```
from numpy import array, linspace, polyfit, poly1d
from pylab import plot

nivel = array([2,3,4,5,6,7,8,9,10,11,12,13,14],dtype=float)
l1s = array([43,44,44,47,52,58,62,65,71,77,82,86,86],dtype=float)
l1d = array([43,44,44,47,52,56,61,65,71,77,81,86,86],dtype=float)
l2s = array([43,44,44,47,52,58,62,65,72,77,82,86,86],dtype=float)
l2d = array([43,44,44,46,52,56,61,65,71,77,81,86,86],dtype=float)
l3s = array([43,43,44,47,52,58,62,65,71,77,82,86,86],dtype=float)
l3d = array([43,44,44,46,52,56,61,65,71,75,81,86,86],dtype=float)

lsMed = (l1s+l2s+l3s)/3
ldMed = (l1d+l2d+l3d)/3

plot(nivel, lsMed, 'b+', linewidth=2)
plot(nivel, ldMed, 'r+', linewidth=2)

grau = 3
coefS = polyfit(nivel, lsMed, grau)
curvaS = poly1d(coefS)

coefD = polyfit(nivel, ldMed, grau)
curvaD = poly1d(coefD)

x=linspace(nivel[0],nivel[-1],101)
yS = curvaS(x)
yD = curvaD(x)

plot(x, yS, 'b')
plot(x, yD, 'r')
```

# Resolução de equações diferenciais

- Problema: pêndulo
- $\ddot{\theta} + b\dot{\theta} + c\sin(\theta) = 0$
- Passando para um sistema de 1ª ordem

$$\dot{\theta} = \omega$$

$$\dot{\omega} = -b\omega - c\sin(\theta)$$

- Condições iniciais

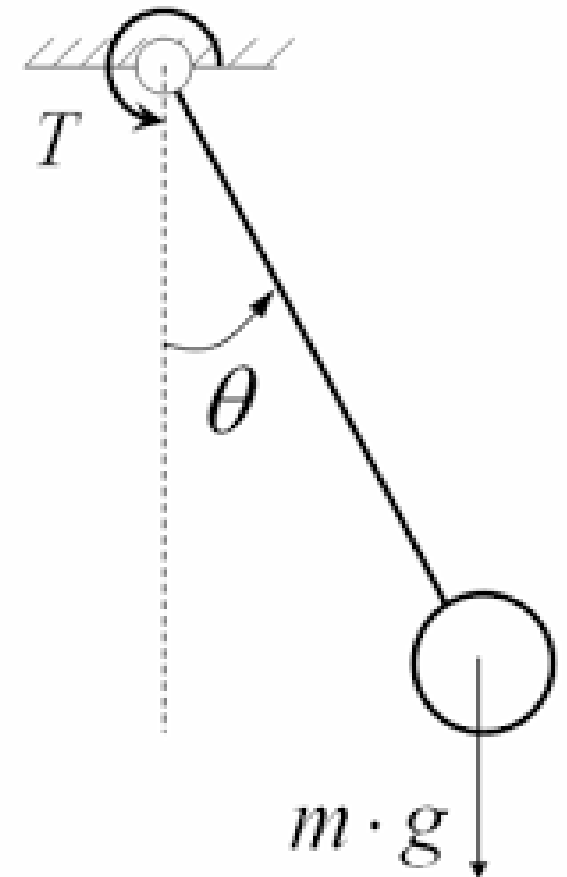
$$\theta = \pi - 0.1$$

$$\omega = 0$$

$$b = 0.25$$

$$c = 0.5$$

t entre 0 e 10, passo 0.01





# Resolução de equações diferenciais

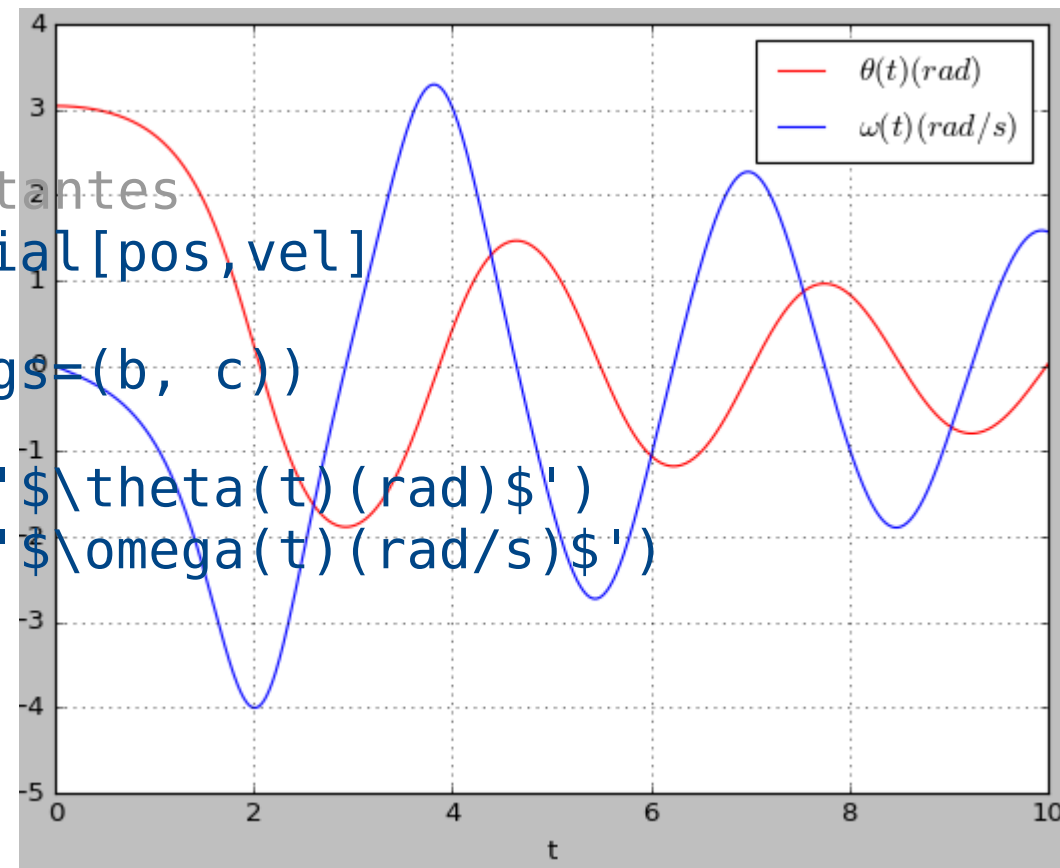
```
from numpy import sin, pi, linspace
from scipy.integrate import odeint
from pylab import plot, legend, xlabel, grid
```

```
def pendulo(y, t, b, c):
    theta, omega = y
    dydt = [omega, -b*omega - c*sin(theta)]
    return dydt
```

```
b = 0.25 # constantes
c = 5.0  # constantes
t = linspace(0, 10, 1001) # instantes
y0 = [pi-0.1, 0.0] # ponto inicial [pos, vel]
```

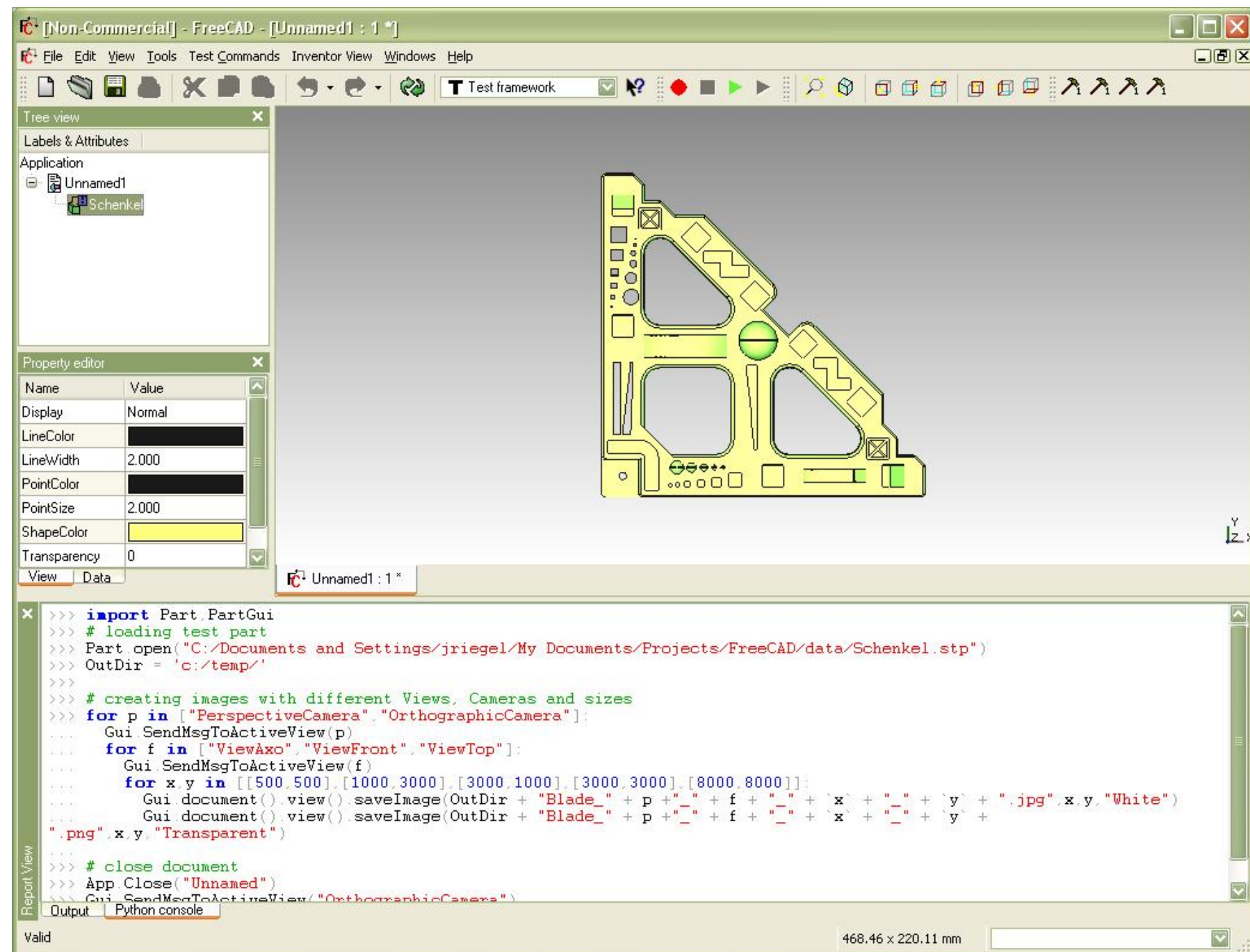
```
res = odeint(pendulo, y0, t, args=(b, c))
```

```
plot(t, res[:, 0], 'r', label=r'$\theta(t)(rad)$')
plot(t, res[:, 1], 'b', label=r'$\omega(t)(rad/s)$')
legend(loc='best')
xlabel('t')
grid()
```

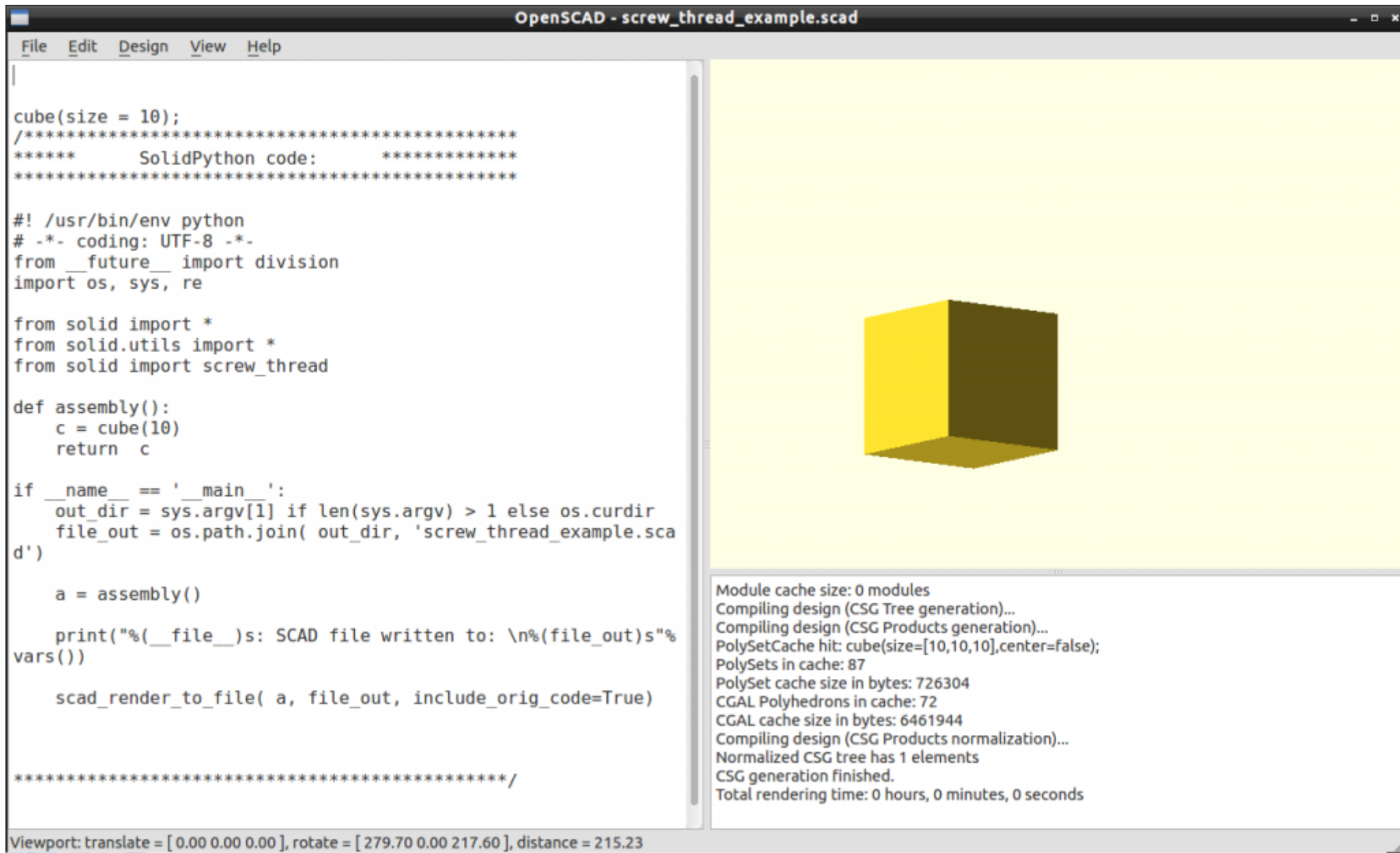


# Outras possibilidades

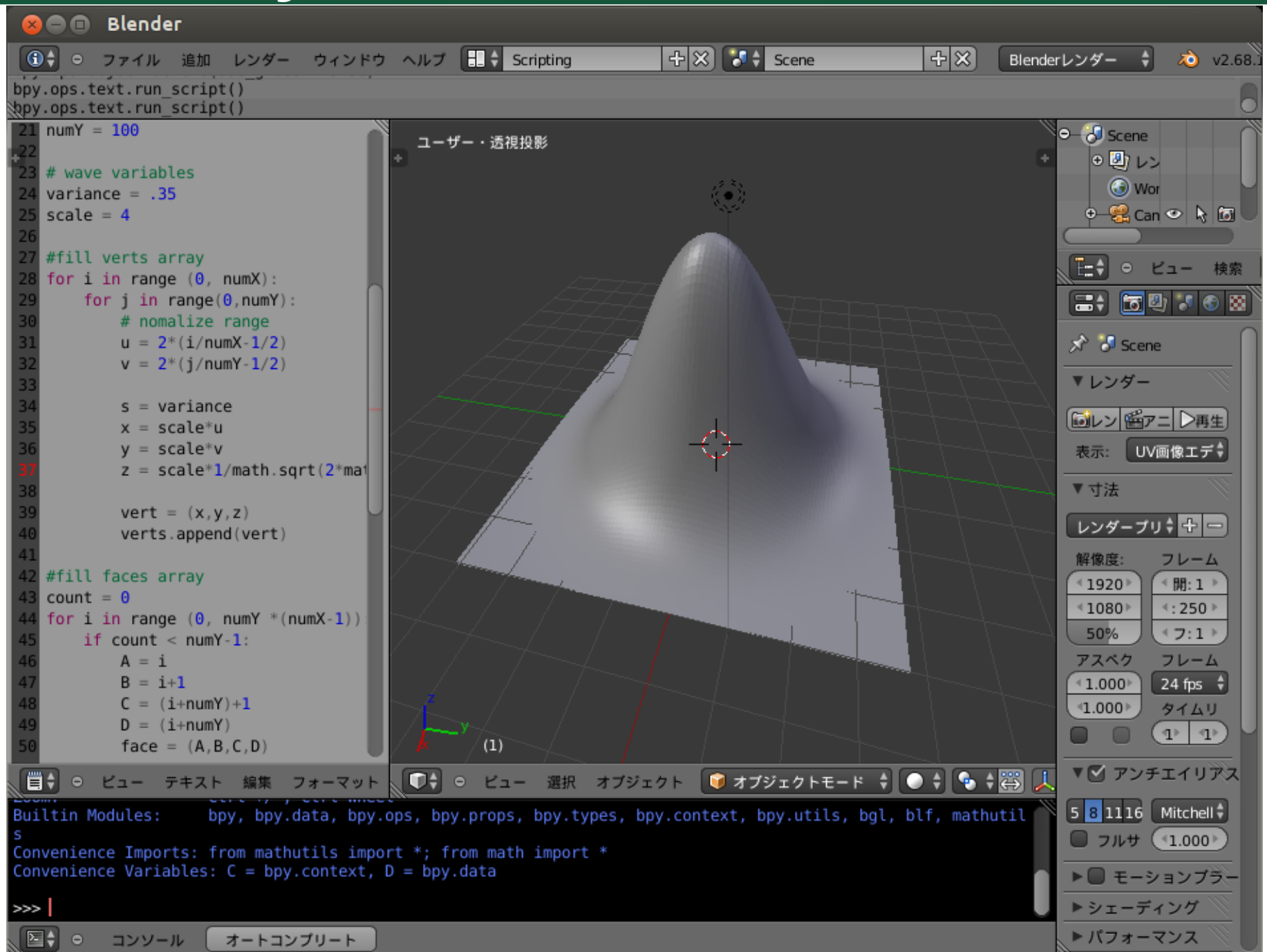
- Comunicação com dispositivos
  - pyFirmata, minimalmodbus, paho, opcua
  - MicroPython
- Interface com outros softwares
  - Blender, FreeCAD, OpenSCAD
  - ROS
  - Gazebo, OpenRave
  - OpenCascade
- Bindings/Links com outras linguagens
  - C/C++, R, Julia



# OpenSCAD & Python



# Blender & Python



# Python Científico

**Carlos Rodrigues Rocha**

[carlos.rocha@riogrande.ifrs.edu.br](mailto:carlos.rocha@riogrande.ifrs.edu.br)

# Obrigado