# Introdução ao Python

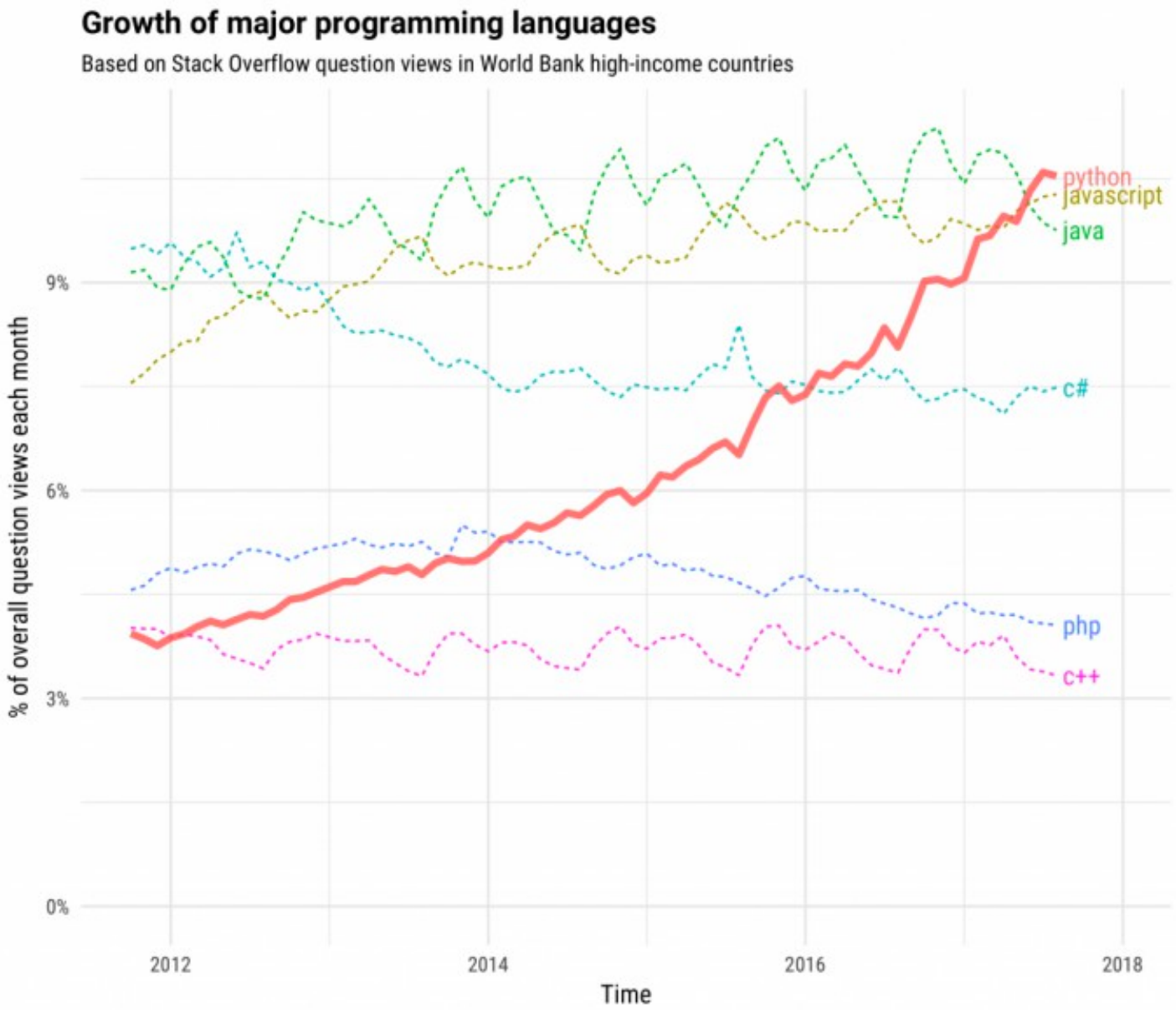
Pedro Campelo

2019

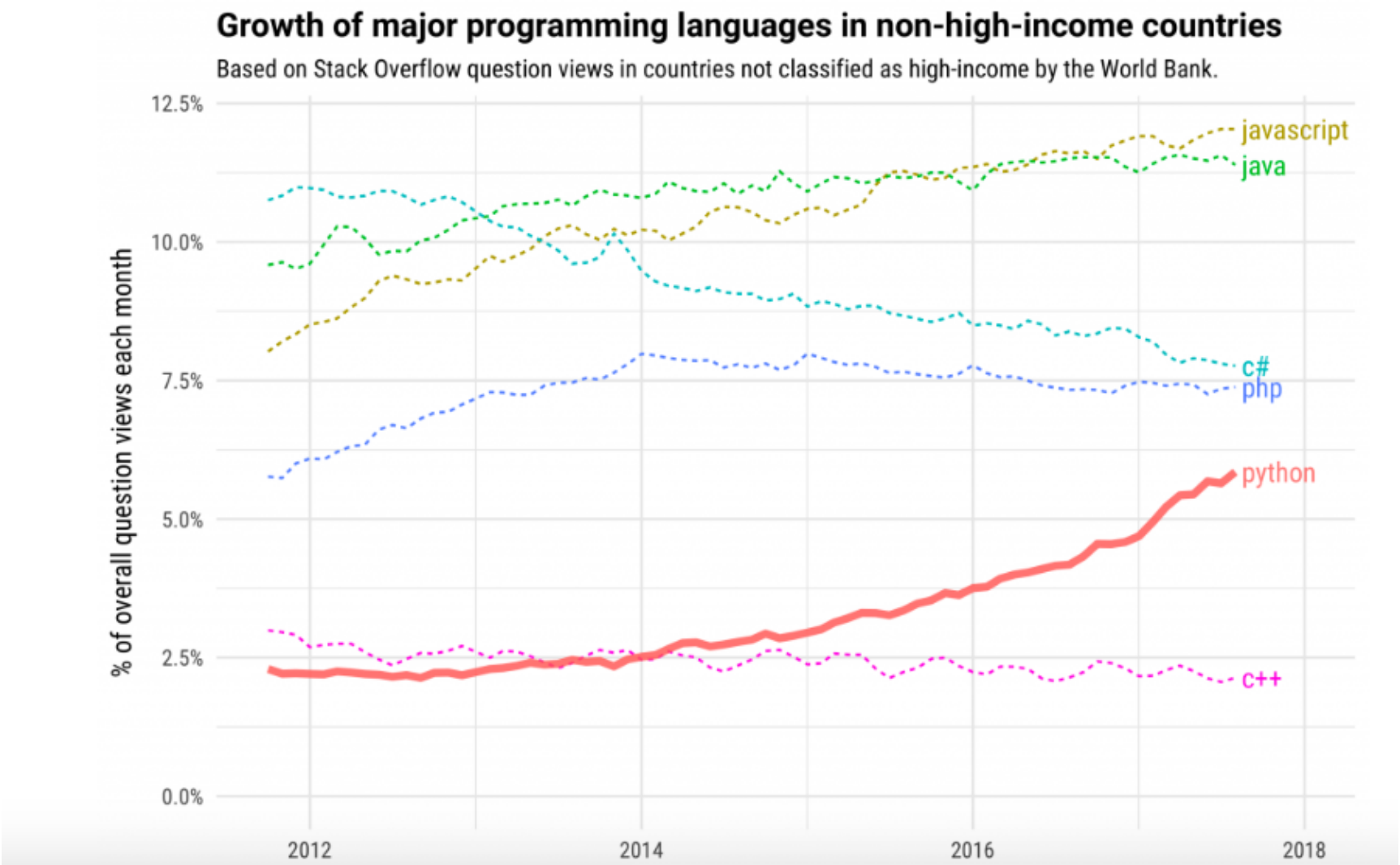
# Sumário

# Sumário

# Por que Python?



# Por que Python?



# Python

- Interfaces:
  - IDLE
  - Spyder (Anaconda)
  - Jupyter (Anaconda)
  - Rodeo, PyCharm, etc

- Algumas práticas:
  - Ler bastante código do seu nível para os seus objetivos, seja documentação ou exemplos
  - Ler com calma mensagens de erro para identificar o que aconteceu
  - Comentar o código
  - Não misturar línguas na hora de comentar
  - Respeitar 79 caracteres por linha
  - Python PEP-8 para mais informações

# Códigos

```
print("Hello World")
```

```
Hello World
```

# Códigos

```
def my_python():
    print("Python eh bonito")

if __name__ == '__main__':

    my_python()

    Python eh bonito
```

# Códigos

```
def funcao1(x):
    return x*x

def funcao2(x):
    y=x*x
    return y

def parouimpar(x):
    if(x%2==0):
        print("par")
    else:
        print ("impar")
```

# Códigos

```
if __name__ == '__main__':

    funcao1(5)
    25

    funcao2(4)
    16

    parouimpar(234)
    par
```

# Códigos

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
if __name__ == '__main__':

    print('Com utf-8 podemos usar acentos nos
    scripts\n')
    print('Vamos multiplicar a1 por a2:')

    a1 = input('a1: ')
    a2 = input('a2: ')
    print('Cuidado, a1 e a2 entraram como strings:')
    print('tipo de a1: %s' % type(a1))
    print('tipo de a2: %s' % type(a2))

    # continua no proximo slide
```

# Códigos

```
# continuacao

a1 = float(a1)
a2 = float(a2)

# print(a1*a2)
print('Resultado: %.2f' % (a1*a2))
```

# Códigos

```
# Diferencas entre listas e tuplas

if __name__ == '__main__':

    tupla = (1,2,3,4,5)
    lista = []

    for i in range(len(tupla)):
        lista.append(tupla[i])

    print(type(tupla),type(lista))
    print(tupla, lista)
```

# Códigos

```
# Algumas coisas sobre indexacao listas

if __name__ == '__main__':

    lista = ['O', 'S', 'R', 'E', 'V', 'E', 'R']

    for _ in reversed(lista):
        print(_)

    print(lista[1:4])
    print(lista[1:-3])

    print(lista[::-1])
```

# Sumário

# Bibliotecas

- Numpy
- Pandas
- Matplotlib
- Datetime

# Bibliotecas - Numpy

```
import numpy as np

    arranjo = np.array([(1.5,2,3), (4,5,6)])
    matriz0 = np.zeros( (3,4) )
    matrix1 = np.ones( (3,4) )
    lista = np.linspace( 0, 2, 9 )
    coluna=np.random.randn(100,1)
    raiz2=np.sqrt(16)
```

# Bibliotecas - Pandas

```
import pandas as pd

    data = {
    'Pais': ['Belgica', 'India', 'Brasil'],
    'Capital': ['Bruxelas', 'Nova Delhi', 'Brasilia'],
    'Populacao': [123465, 456789, 987654]
            }
    df = pd.DataFrame(data,
         columns=['Pais','Capital','Populacao'])

    df.drop('Pa s', axis=1)
    df.shape()
    df.sum()
    df.describe()
    df['Nova Coluna']=0
```

# Bibliotecas - Pandas

```
import pandas as pd

from pandas import Series
from pandas import read_csv

    dados = read_csv('dados.csv', sep=';',header=0,
                     index_col='Time', squeeze=True)
    dados.head()

    df = pd.DataFrame(index=range(100))
    df[var1]=np.random.randn(100,1)
    df[var2]=3*np.random.randn(100,1)
```

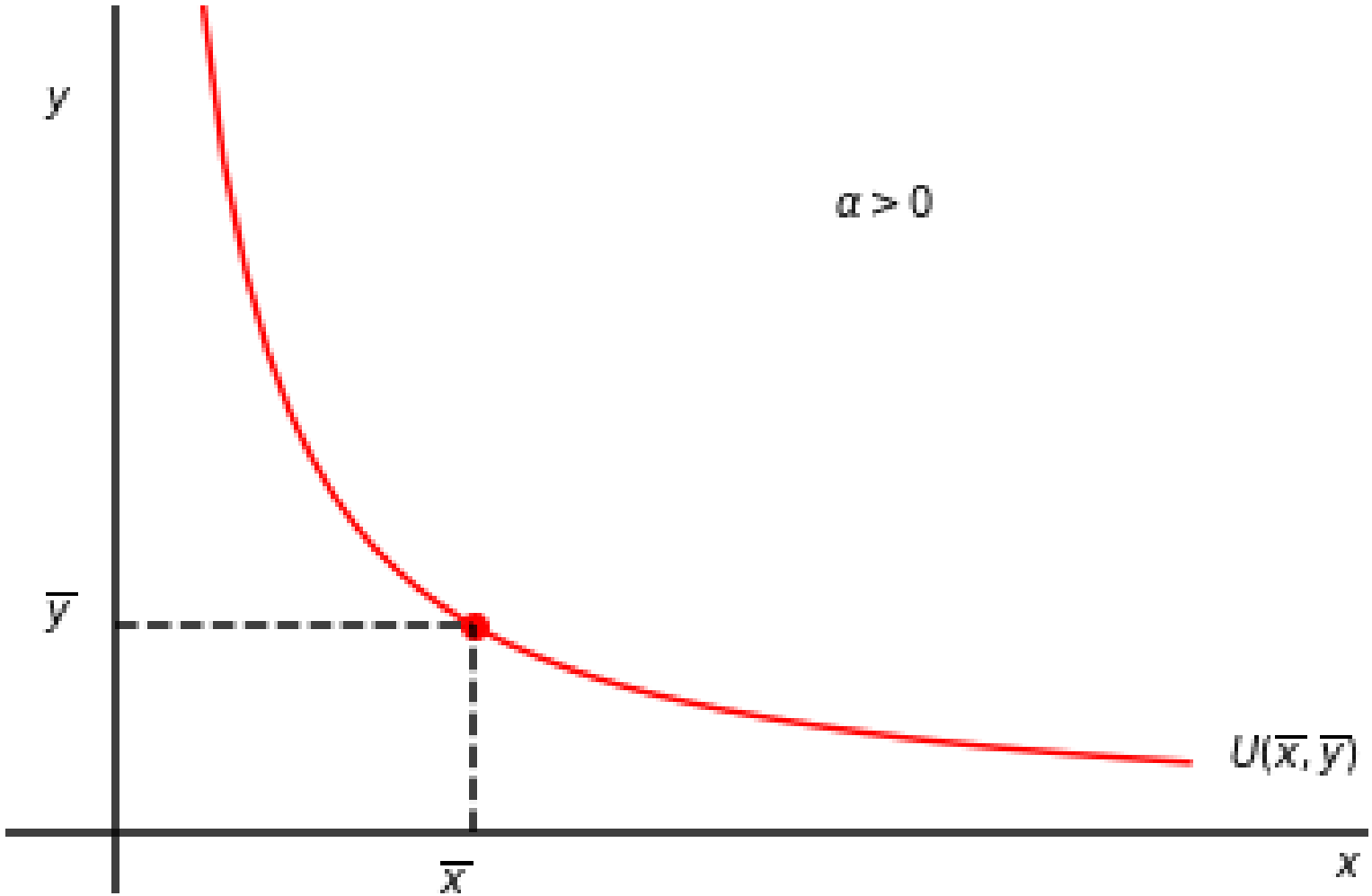
# Bibliotecas - MatPlotLib

```
x = np.linspace(0,3)
y= 1/x

xp = (1)
yp = (1)

xv1=[1,1,1,1]
yv1=[0,0.25,0.5,1]

xv2=[0,0.25,0.5,1]
yv2=[1,1,1,1]
```

# Bibliotecas - MatPlotLib

```
import matplotlib.pyplot as plt
    plt.axis([-0.3, 3.5, -0.3, 4])
    plt.axhline(0, color='black')
    plt.axvline(0, color='black')
    plt.plot(x, y, color='red')
    plt.plot(xp, yp, 'ro')
    plt.plot(xv1, yv1, 'r--', color='black')
    plt.plot(xv2, yv2, 'r--', color='black')
    plt.text(-0.2, 3.5, '$y$')
    plt.text(3.4, -0.2, '$x$')
    plt.text(0.9, -0.3, r'$\overline{x}$')
    plt.text(-0.2, 1, '$\overline{y}$')
    plt.text(3.1,0.3, '$U(\overline{x},\overline{y})$'
    plt.text(2,3, r'$ \alpha > 0$')
    plt.axis('off')
```

# Bibliotecas - MatPlotLib



# Sumário

# For e While

- Uma das tarefas que um computador faz muito bem sem ficar chateado é fazer atividades repetidas.

- As duas estruturas mais comuns são:
  - For
  - While

# Algoritmos

```
def fibonacci(n):
    if(n==1):
        print ("1st Fibonacci number is", 0)
    elif (n==2):
        print ("2nd Fibonacci number is", 1)
    else:
        fib2=0
        fib1=1
        for i in range(3,n+1):
            print ("i: "),i
            fib=fib2+fib1
            fib2=fib1
            fib1=fib
            print (i,"nth Fibonacci number is", fib)
```

# Algoritmos

```
import numpy as np

def is_prime(n):
    prime=True
    count=2
    sqrtn=np.floor(np.sqrt(n))
    while (prime and count<=sqrtn):
        print (count)
        if (np.mod(n,count)==0):
            prime=False
        count=count+1
    if(prime):
        print("Eh primo")
    else:
        print("Nao eh primo")
```

# Algoritmos

```
def fatorial (n):
    if (n>1):
        return n*fatorial(n-1)
    else:
        return 1

def fatorial_2 (n):
    x = 1
    for i in range (2,n+1):
        x=x*i
    return x
```

# Algoritmos

```
def fatoracao (n):
    if (n==0):
        return None
    elif (n==1):
        return 1
    else:
        Fatores =[]
        for i in range(2,n+1):
            while n % i == 0:
                n= n/i
                Fatores.append(i)
        return Fatores
```

# Recursão

- Iteração: É uma técnica para a solução de problemas computacionais onde a mesma tarefa é repetida várias vezes.

- Recursão: É uma técnica para a solução de problemas computacionais em que problemas grandes são reduzidos a problemas menores da mesma forma.

# Algoritmos

```
def fib(n):
    f2=0
    f1=1
    for i in range (2,n):
        f=f2+f1
        f2=f1
        f1=f
    return f
```

# Algoritmos

```
def fib_recursive(n):
    print ("n: ", n)
    if (n>2):
        return fib2(n-1)+fib2(n-2)
    else:
        if(n==2):
            return 1
        else:
            return 0
```

# Algoritmos

```
def fib_recursive2(n, x1=0, x2=1):
    if (n==1):
        return x1
    else:
        if (n==2):
            return x2
        else:
            return fib3(n-1, x2, x1+x2)
```

# Sumário

# Monte Carlo - Círculo

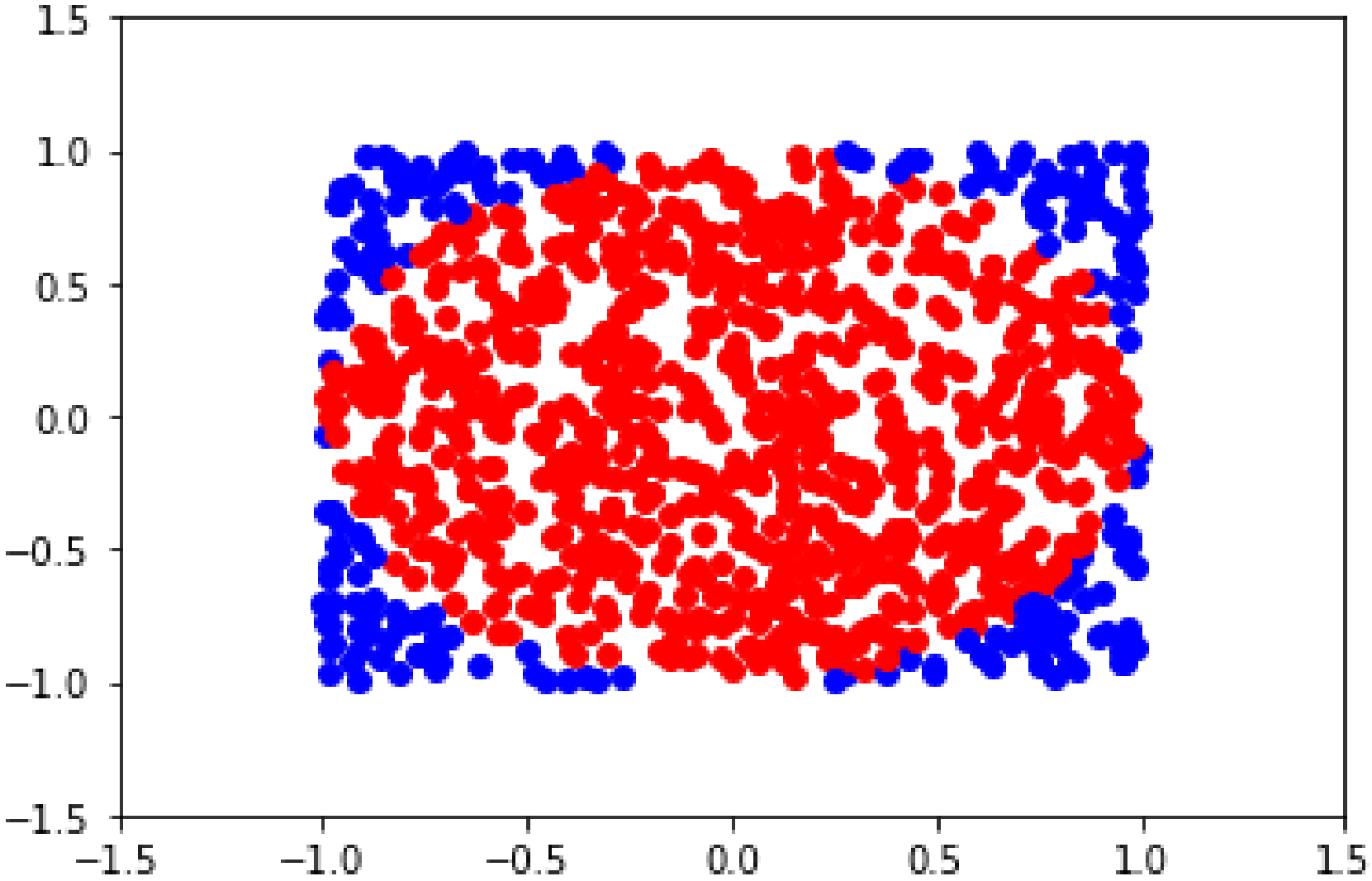
- Considere por exemplo o problema simples de calcular a relação entre as áreas de um quadrado e um círculo que está inscrito nesse quadrado e tangencia os quatro lados dele. Assuma por exemplo que o círculo tem raio r $= 1$ e, consequentemente, o quadrado tem lado l $= 2$. A solução desse problema é óbvia, pois sabemos que a relação entre as áreas é dada por:

$$\frac{\pi r^2}{l^2} = \frac{\pi}{4}$$

# Algoritmos - Monte Carlo

```
import random
import matplotlib.pyplot as plt
def quadradoCirculo(n):
    nc=0
    plt.axis([-1.5, 1.5, -1.5, 1.5])
    plt.hold(True)
    for i in range(0,n):
        x=random.uniform(-1.0,1.0)
        y=random.uniform(-1.0,1.0)
        if(x**2+y**2<1.0):
            plt.plot([x],[y],'ro')
            nc=nc+1.0
        else:
            plt.plot([x],[y],'bo')
    return nc/n
```

# Algoritmos - Monte Carlo



# Monte Carlo - Regressão Linear

- Suponha que você quer analisar o que ocorre com a distribuição do estimador de mínimos quadrados do coeficiente angular da reta dada por $y = \alpha + \beta x + u$ quando aumentamos o tamanho da amostra.

- Suponha que $u$ é uma variável aleatória normal padrão e $\alpha = 2$ e $\beta = 3$. Vamos fazer o papel de Deus e gerar os valores de y e usar os vetores $[x; y]$ para estimar os coeficientes $\alpha$ e $\beta$ (embora estejamos aqui particularmente interessados nas estimativas de $\beta$).

# Algoritmos - Monte Carlo

```
import random
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np

def geracaoDados(n):
    alpha=2
    beta=3
    x=np.random.normal(0,4,n)
    u=np.random.normal(0,1,n)
    y=alpha*np.ones(n)+beta*x+u
    return x,y
```

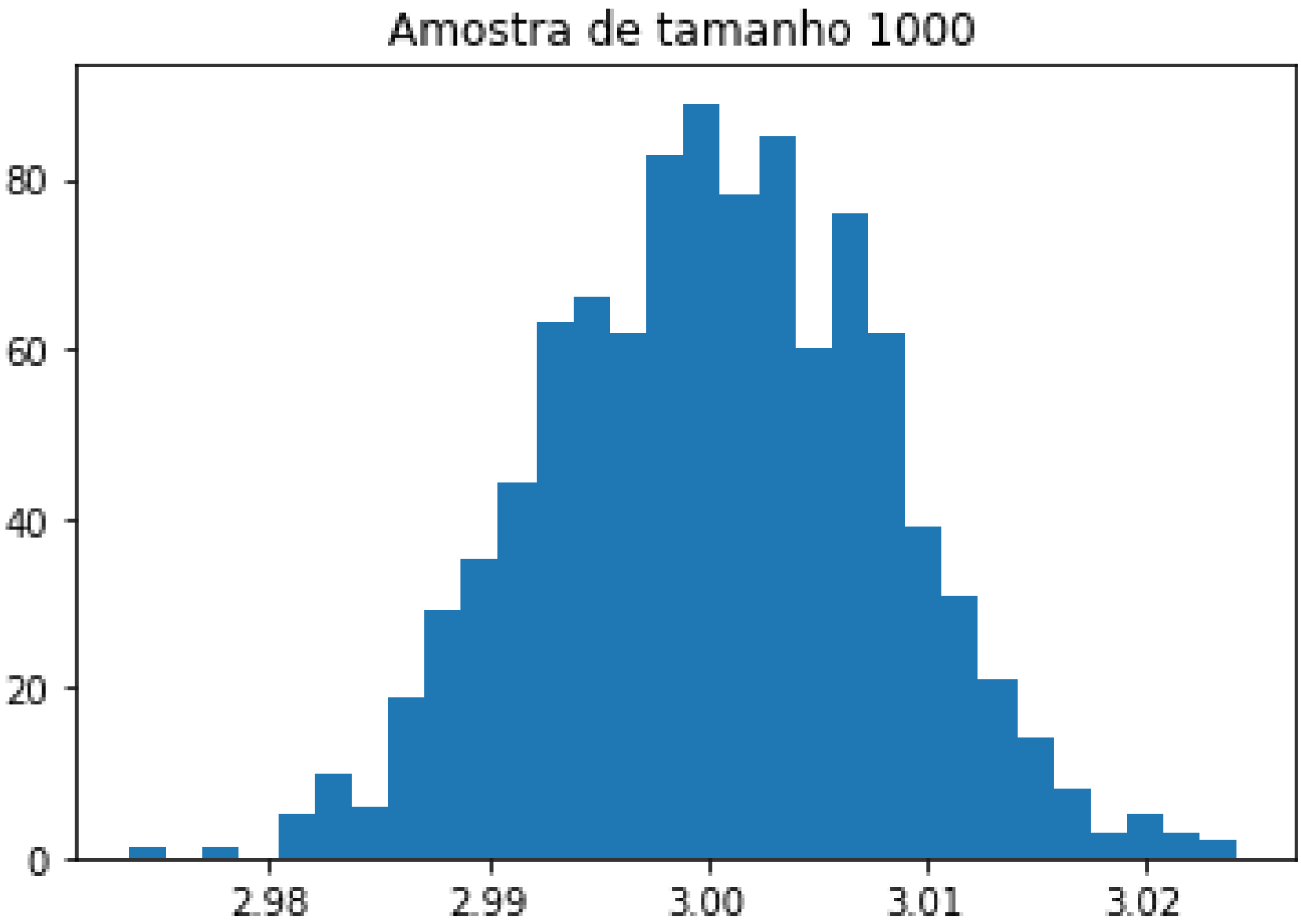
# Algoritmos - Monte Carlo

```
def estimacao(x,y):
    beta, alpha, r_value, p_value,
                std_err = stats.linregress(x,y)
    return alpha,beta
```

# Algoritmos - Monte Carlo

```
if __name__ == '__main__':

    numeroAmostras=1000
    n=1000 # numero de observacoes
    vetor=np.empty([numeroAmostras])
    for i in range(0,numeroAmostras):
        [x,y]=geracaoDados(n)
        [alpha,beta]=estimacao(x,y)
        vetor[i]=beta
    plt.hist(vetor,bins=30)
    plt.title('Amostra de tamanho 1000')
```

# Algoritmos - Monte Carlo



# Algoritmos - Monte Carlo