

INF 1514

Introdução à Análise de Dados

Material 8



Python

- Python (<http://www.python.org/>) é uma linguagem de programação interpretada de alto nível de propósito geral.
- Foi lançada por Guido van Rossum em 1991 e atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation.
- Enfatiza a legibilidade do código com uso de indentação para delimitar escopos.
- Possui uma sintaxe concisa e clara disponibilizando recursos poderosos através da sua biblioteca padrão e de módulos e frameworks desenvolvidos por terceiros.
- É aplicada em diversas áreas como: desenvolvimento Web e de jogos, Inteligência Artificial, Ciência de Dados e automação de scripts.

Algumas características do Python

- É **case-sensitive**: então “A” e “a” são símbolos diferentes e se referem a diferentes variáveis.
- Comandos são separados por quebras de linha, ao invés de “;” como em outras linguagens.
- Python suporta os tipos primitivos de dados: int, float, complex, boolean e string. Exemplos de outros tipos presentes são:
 - list – usado para agrupar um conjunto de elementos.
 - tupla – semelhante ao tipo list, porém, imutável.
 - dict – para agrupar elementos que serão recuperados por uma chave.
 - set – é uma coleção desordenada de elementos, sem elementos repetidos.
- Comentários de linha começam com “#” e blocos de comentários são delimitados (início e fim) por “""" (3 aspas).

Algumas características do Python

- Como a maioria das linguagens de programação, permite atribuir valores a variáveis.
- A declaração de variáveis não indica seu tipo, sendo este inferido a partir do conteúdo atribuído na inicialização.
- As variáveis possuem tipagem dinâmica, ou seja, suas variáveis podem armazenar dados de diferentes tipos de dados ao longo da execução de um processo.
- Por padrão, constantes no Python devem ser declaradas com todas as letras maiúsculas, porém, por possuir tipagem dinâmica, os valores atribuídos a constantes podem ser alterados.

Algumas funções do Python

Função	Descrição
<code>sqrt()</code>	Do pacote <code>math</code> , calcula a raiz quadrada.
<code>abs()</code>	Valor absoluto.
<code>exp()</code>	Do pacote <code>math</code> , exponencial.
<code>range()</code>	Permite criar listas de números inteiros.
<code>log()</code>	Do pacote <code>math</code> , logaritmo na base <code>e</code> .
<code>sin()</code> <code>cos()</code> <code>tan()</code>	Do pacote <code>math</code> , funções trigonométricas.
<code>asin()</code> <code>acos()</code> <code>atan()</code>	Do pacote <code>math</code> , funções trigonométricas inversas.
<code>sorted()</code>	Ordena os elementos de uma lista.
<code>replace()</code>	Permite substituir partes de uma string por outros caracteres.
<code>max(x)</code>	Retorna o maior valor encontrado em <code>x</code> (lista, etc).
<code>mean(x)</code>	Do pacote <code>statistics</code> , retorna a média dos elementos de <code>x</code> (lista, etc).
<code>upper()</code>	Função do objeto e retorna a string em letras maiúsculas.
<code>hist()</code>	Do pacote <code>matplotlib</code> , cria um histograma.

Instalação do Python

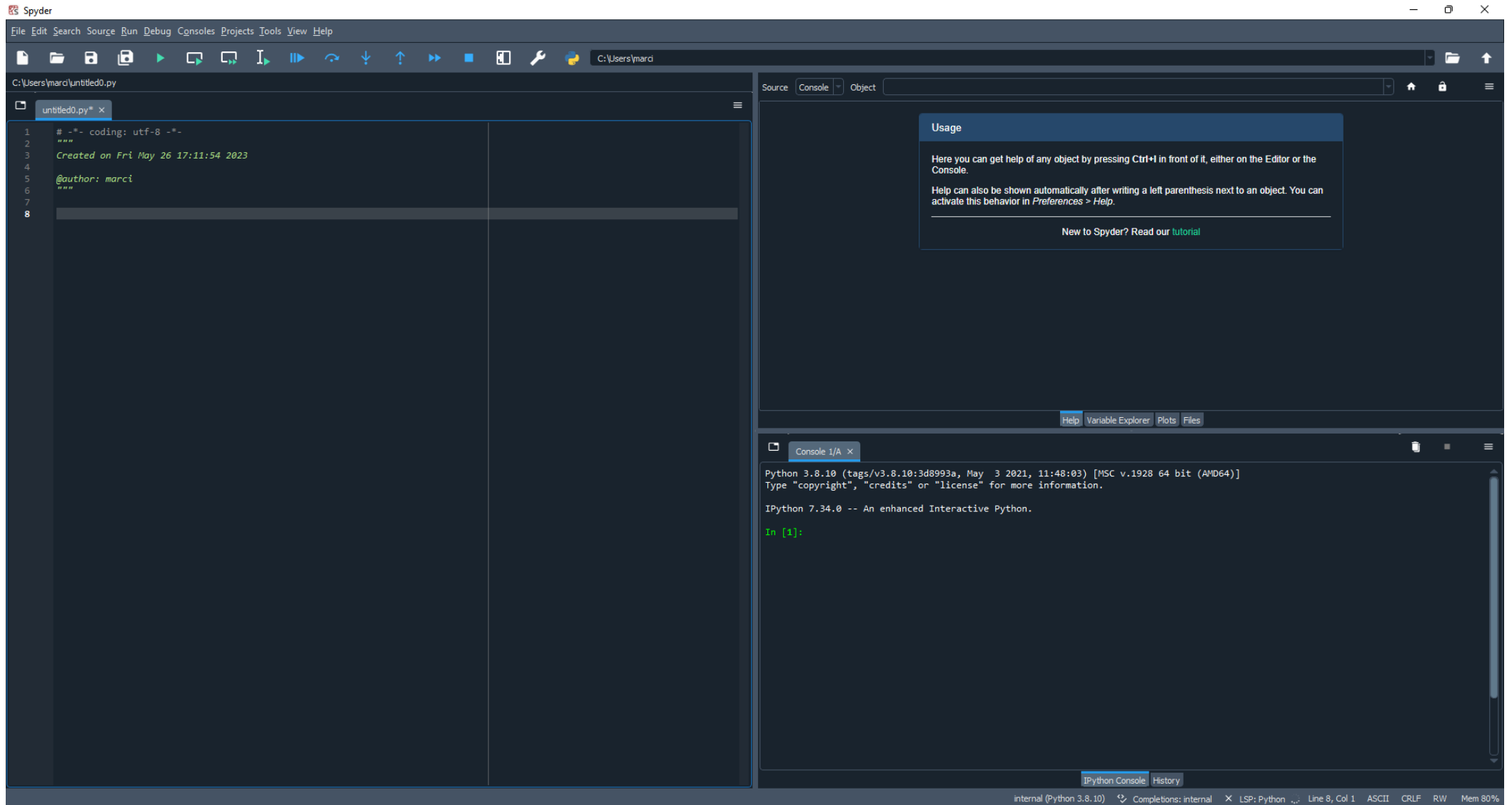
- Existem vários IDEs (*Integrated Development Environment*) de programação que permitem trabalhar com Python.
- Um IDE é um software que combina ferramentas comuns de desenvolvimento em uma única interface gráfica do usuário (GUI), facilitando o desenvolvimento de aplicações.
- Neste curso iremos trabalhar com o **Spyder** cujos links de instalação estão abaixo.
- Para Windows:

https://github.com/spyder-ide/spyder/releases/latest/download/Spyder_64bit_full.exe

- Para Mac:

<https://github.com/spyder-ide/spyder/releases/latest/download/Spyder.dmg>

Spyder



Spyder

- Para conseguir ajuda sobre um comando utilize o conjunto de teclas **Ctrl-I** tendo o mouse posicionado sobre o comando tanto no Editor quanto no IPython console.
- Também é possível obter ajuda sobre um comando através do comando **help** diretamente no console.

`help(comando)`

- Para instalar uma biblioteca no seu ambiente, você pode usar o comando **pip install** seguido do nome da biblioteca.

`pip install nomebiblioteca`

- Para usar a biblioteca em seu programa, use o comando **import** seguido do nome da biblioteca:

`import nomebiblioteca`

Sintaxe R x Python

- Em termos de sintaxe, de forma simplificada, podemos dizer que ambos funcionam de maneira similar, mas com algumas diferenças significativas.
- Importante, indentação é feita com tabulações ou normalmente dois ou quatro espaços, e faz parte da sintaxe!

```
peso <- 80
altura <- 1.60
imc <- (peso / (altura * altura))
print(imc)
```

```
peso = 80
altura = 1.60
imc = (peso / (altura * altura))
print(imc)
```

R x Python – Tipos primitivos de dados

- `class(2L) # integer`
 - `class(2.3) # numeric`
 - `class('PUC-RIO') # character`
 - `class(TRUE) # logical`
 - Exemplos de outros tipos presentes: vetor, lista, matriz, array e data frame.
- `type(2) # int`
 - `type(2.3) # float`
 - `type("PUC-RIO") # str`
 - `type(True) # bool`
 - Exemplos de outros tipos presentes: lista, tupla, dicionário, conjunto, array e DataFrame.

R x Python – Operadores algébricos

Operador	Descrição
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão

Operador	Descrição
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão



Em ambas as linguagens o operador `**` pode ser utilizado para realizar a operação de potência.

R x Python – Operadores relacionais e lógicos

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

Operador	Descrição
<code>==</code>	Operador de igualdade
<code>></code>	Operador “maior que”
<code><</code>	Operador “menor que”
<code><=</code>	Operador “menor ou igual”
<code>!</code>	Operador “Não”
<code>&</code>	Operador lógico “E”
<code> </code>	Operador lógico “OU”

R x Python – Vetores e listas

- Em R, um vetor é uma coleção mutável de elementos do mesmo tipo.
- O índice de um vetor em R começa em **1** e é **inclusivo**.

```
v <- c (6, 7, 2, 8)
print(v[1]) # imprime 6
print(v[1:2]) # imprime 6 e 7
```

- Vetores em Python são guardados em listas.
- Em Python, uma lista é uma coleção mutável de elementos de qualquer tipo.
- O índice de uma lista em Python começa em **0** e é **não inclusivo**.

```
ls = [6, 7, 2, 8]
print(ls[0]) # imprime 6
print(ls[0:1]) # imprime só o 6
```

R x Python – Controle de fluxo if/else

- Usa {} para delimitar o escopo de execução.

```
a <- 19
b <- 29
if (a > b) {
  print("a é maior que b.")
} else if (a == b) {
  print("a é igual a b.")
} else {
  print("b é maior que a.")
}
```

- Usa : e **indentação** para delimitar o escopo de execução.

```
a = 19
b = 29
if a > b:
  print("a é maior que b.")
elif a == b:
  print("a é igual a b.")
else:
  print("b é maior que a.")
```

R x Python – Controle de fluxo if/else

- Mais exemplos em Python.

```
a = 30
b = 30
print("A") if a > b else print("=") if a == b else print("B")
```

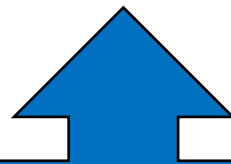
```
x = 5
if x > 0:
    print("x é positivo.")
    if x > 10:
        print("x é maior do que 10.")
    else:
        print("x é menor ou igual a 10.")
else:
    print("x é negativo ou zero.")
```

```
a = 100
b = 20
c = 200
if a > b and c > a:
    print("As duas condições são verdadeiras.")
```

R x Python – Controle de fluxo **for**

```
for (i in 1:10) {# imprime de 1 a 10
  print(i)
}
```

```
for i in range(10): # imprime de 0 a 9
  print(i)
```



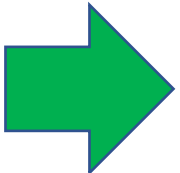
A função **range()** tem como padrão **0** como valor inicial, porém é possível especificar o valor inicial adicionando um parâmetro: **range(2, 10)**, que significa valores de **2** a **10** (**não incluindo 10**).

O padrão da função **range()** é incrementar a sequência em **1**, no entanto, é possível especificar o valor do incremento adicionando um terceiro parâmetro: **range(2, 10, 3)**.

R x Python – Controle de fluxo **for**

- **break** e **next**.

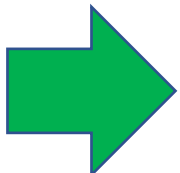
```
for (x in 1:6) {  
  if (x == 4)  
    break  
  if (x == 2)  
    next  
  print(x)  
}
```



1
3

- **break**, **continue** e **else**.

```
for x in range(6):  
  if (x == 4):  
    break  
  if (x == 2):  
    continue  
  print(x)  
else:  
  print("Fim do for!")
```



0
1
3

Python:

Com a instrução **break**, podemos interromper o loop antes que ele percorra todos os itens.

Com a instrução **continue**, podemos interromper a iteração atual do loop e continuar com a próxima.

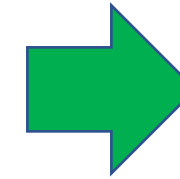
A instrução **else** em um **loop for** especifica um bloco de código a ser executado quando o loop terminar.

O bloco **else** NÃO será executado se o **loop for** for interrompido por uma instrução **break**.

R x Python – Controle de fluxo for

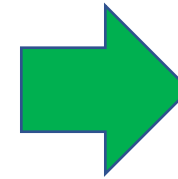
- Mais exemplos em Python.

```
estados = ["Rio de Janeiro", "São Paulo", "Santa Catarina"]  
for estado in estados:  
    print(estado, len(estado))
```



Rio de Janeiro 14
São Paulo 9
Santa Catarina 14

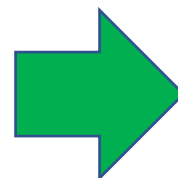
```
for letra in "Rio de Janeiro":  
    print(letra)
```



R
i
o

...

```
lista_estado = [ ["RJ", 15993583, 407123], ["SP", 41252160, 1247596], ["SC", 6249682, 152482] ]  
for estado in lista_estado:  
    for dado_estado in estado:  
        print(dado_estado)
```



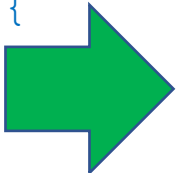
RJ
15993583
407123
SP
41252160
...

A variável **lista_estado** é uma **lista** de **listas**, onde **lista_estado[2]** possui o valor **['SC', 6249682, 152482]**.
A variável **estado** é uma **lista**.

R x Python – Controle de fluxo **while**

- break e next

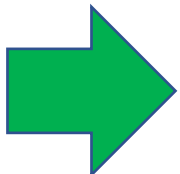
```
x <- 0
while (x <= 6) {
  x <- x + 1
  if (x == 4)
    break
  if (x == 2)
    next
  print(x)
}
```



1
3

- break, continue e else

```
x = 0
while (x <= 6):
  x = x + 1
  if (x == 4):
    break
  if (x == 2):
    continue
  print(x)
else:
  print("Fim do while!")
```



1
3

Python:

Com a instrução **break**, podemos interromper o loop antes que ele percorra todos os itens.

Com a instrução **continue**, podemos interromper a iteração atual do loop e continuar com a próxima.

A instrução **else** em um **loop while** especifica um bloco de código a ser executado quando o loop terminar.

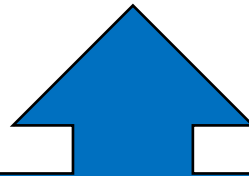
O bloco **else** NÃO será executado se o loop **while** for interrompido por uma instrução **break**.

R x Python – Funções

- A palavra reservada **function** permite definir funções.
- A palavra reservada **def** permite definir funções.

```
nome_da_função <- function (par1, par_2, etc...)  
{  
  corpo_da_função  
}
```

```
def nome_da_função (par1, par_2, etc...):  
    corpo_da_função
```



Python:

A instrução **return** permite que uma função retorne um valor.

Para chamar uma função, deve ser usado o nome da função seguido de parênteses.

Por padrão, se a função espera 2 parâmetros, ela deve ser chamada com 2 argumentos, nem mais nem menos.

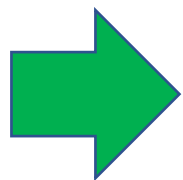
Os parâmetros podem ser de quaisquer tipo de dados, sendo tratados como do mesmo tipo dentro da função.

R x Python – Funções

- Em R e Python é possível enviar argumentos usando a sintaxe **chave = valor**.

A função **str** converte o número inteiro **idade** em string para permitir a concatenação de strings feita com a instrução **+**.

```
def imprimePessoa(nome, idade):  
    print("O nome da pessoa é " + nome + " e sua idade é " + str(idade) + " anos.")  
  
imprimePessoa(idade = 30, nome = "Pedra da Silva")
```



O nome da pessoa é Pedra da Silva e sua idade é 30 anos.

Os parâmetros foram passados via **chave = valor**.
Observe que a ordem dos parâmetros pode não ser a mesma da definição da função.

R x Python – Funções

- Tratamento de **valor padrão** de **parâmetro de função** em Python.

Parâmetros que possuem **valor padrão** devem ficar após os parâmetros que não possuem valor padrão.

```
def calculaHorasTrabalhadas(dias, jornada = 8):  
    return (dias * jornada)
```

```
horas = calculaHorasTrabalhadas(10)
```

O parâmetro **dias** irá receber o valor **10** e o parâmetro **jornada**, não obrigatório, irá assumir o valor **8**, neste exemplo. A variável **horas** irá receber o retorno da função, neste exemplo, **80**.

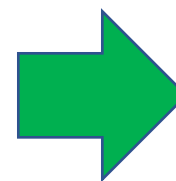
R x Python – Funções

- Em Python, adicionar um ***** antes do nome de um parâmetro na definição da função irá permitir que ele receba uma **tupla** como argumento.

```
def imprimeSorteio(texto, *numeros):  
    print(texto)  
    for numero in numeros:  
        print(numero)
```

```
imprimeSorteio("Números sorteados:", 30, 42, 19)
```

O parâmetro **numeros** possui o modificador ***** na sua definição permitindo que ele receba uma **tupla**.



Números sorteados:

30
42
19

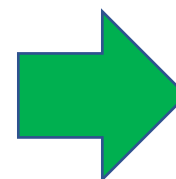
As **tuplas** no Python são tipos de dados semelhantes ao tipo **list**, porém, imutáveis (seus valores não podem ser modificados). No exemplo, o parâmetro **numero** será uma **tupla** com os valores **30, 42 e 19** passados na chamada da função.

R x Python – Funções

- Em Python, adicionar ****** antes do nome de um parâmetro na definição da função irá permitir que ele receba um **dicionário** como argumento.

```
def imprimeSorteio(texto, **numeros):  
    print(texto)  
    for chave in numeros:  
        print(chave, " = ", numeros[chave])
```

O parâmetro **numeros** possui o modificador ****** na sua definição permitindo que ele receba um **dicionário**.



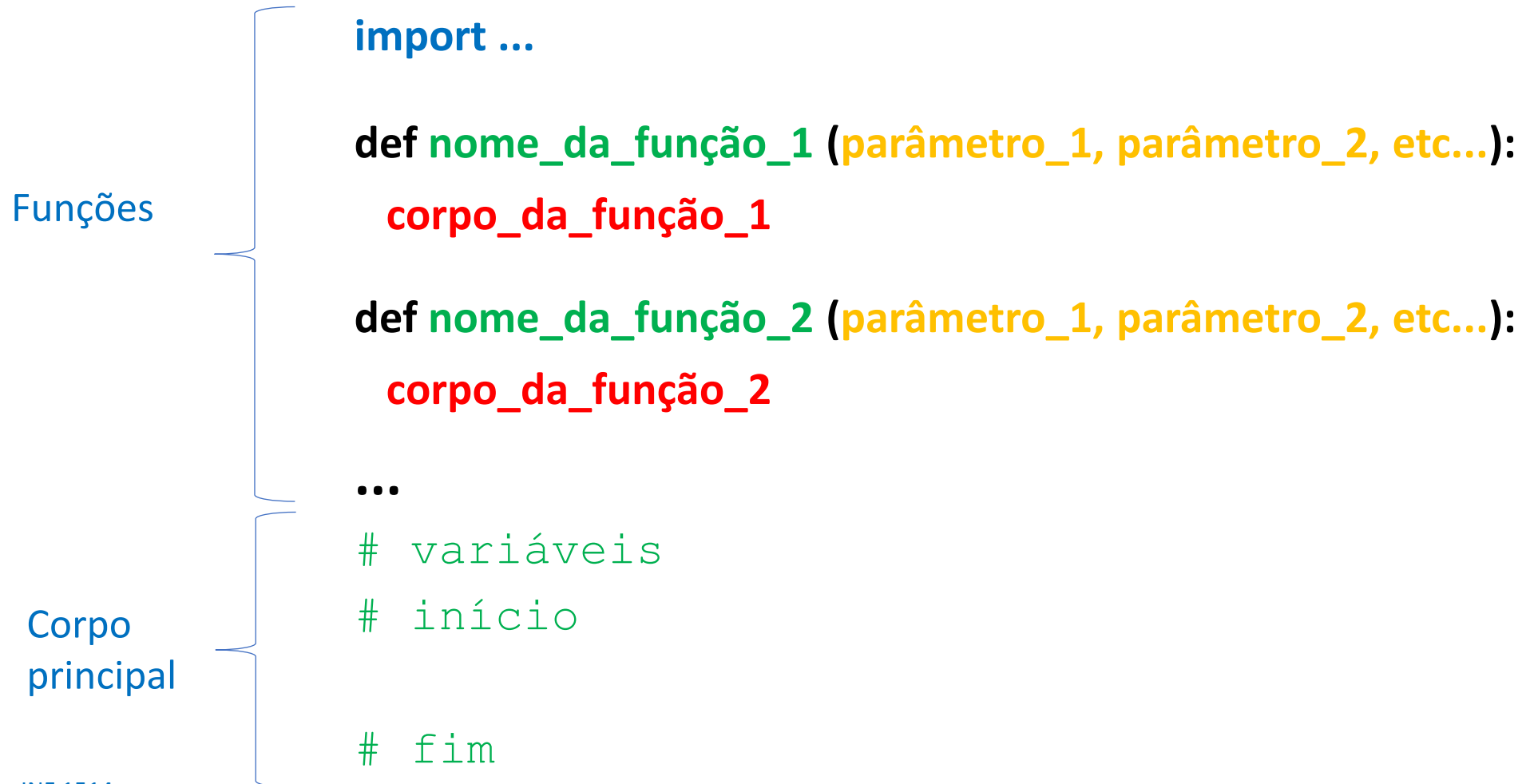
Números sorteados:
Primeiro = 30
Segundo = 42
Terceiro = 19

```
imprimeSorteio("Números sorteados:", Primeiro = 30, Segundo = 42, Terceiro = 19)
```

Os **dicionários** no Python são tipos de dados usados para armazenar dados em pares **chave : valor**.
No exemplo, **numeros** será um dicionário sendo **Primeiro**, **Segundo** e **Terceiro** as chaves para os valores **30**, **42** e **19**, respectivamente.

R x Python – Funções

- Organização básica de um script Python para o nosso curso.



R x Python – Manipulação de dados

- A principal biblioteca para manipulação de dados no R é o **tidyverse**, usando a estrutura baseada em **data.frame**.

```
library(tidyverse)
```

```
df <- read_csv2("C:\\temp\\DadosRJ.csv")
```

```
head(df)
```

```
summary(df)
```

```
media <- mean(df$pibpercapta)
```

```
df.f <- filter(df, pibpercapta > media)
```

```
write_csv2(df.f, "C:\\temp\\FiltroDadosRJ.csv")
```

- Em Python, **pandas** é a biblioteca equivalente ao **tidyverse** e a classe **DataFrame** é a equivalente ao **data.frame** do R.

```
import pandas as pd
```

```
df = pd.read_csv("C:\\temp\\DadosRJ.csv",  
sep = ";", decimal = ",")
```

```
df.head()
```

```
df.describe()
```

```
media = df.pibpercapta.mean()
```

```
df_f = df[df.pibpercapta > media]
```

```
df_f.to_csv("C:\\temp\\FiltroDadosRJ.csv",  
sep = ";", decimal = ",", index = False)
```

R x Python – Manipulação de dados

- Seleccionando colunas.

```
library(tidyverse)

df <- read_csv2("C:\\temp\\DadosRJ.csv")

df.s <- select(df, codigo, populacao)

head(df.s)

summary(df.s)
```

O comando **chdir()** da biblioteca **os** permite definir um diretório de trabalho e o comando **getcwd()** permite visualizar qual é o diretório de trabalho corrente.

- O **pandas** possui funções semelhantes às do **tidyverse**.

```
import pandas as pd
import os

os.chdir("C:\\temp")

df = pd.read_csv("DadosRJ.csv", sep = ";",
decimal = ",")

df_s = df[["codigo", "populacao"]]

df_s.head()

df_s.describe()
```

R x Python – Plotando gráficos

- Uma das bibliotecas mais usadas é a **ggplot2**.
- Uma das bibliotecas mais usadas é a **Matplotlib**.

```
library(ggplot2)

y <- rnorm(mean = 0, sd = 10, n = 500)

ggplot() + geom_histogram(aes(y)) +
ggtitle("Histograma")
```

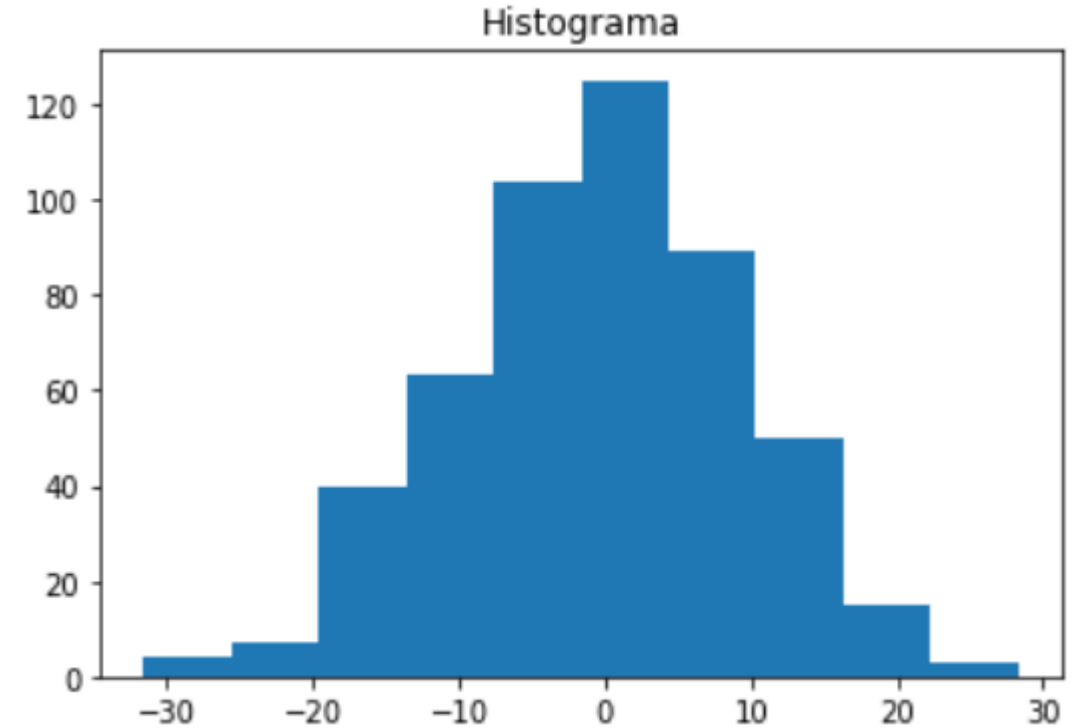
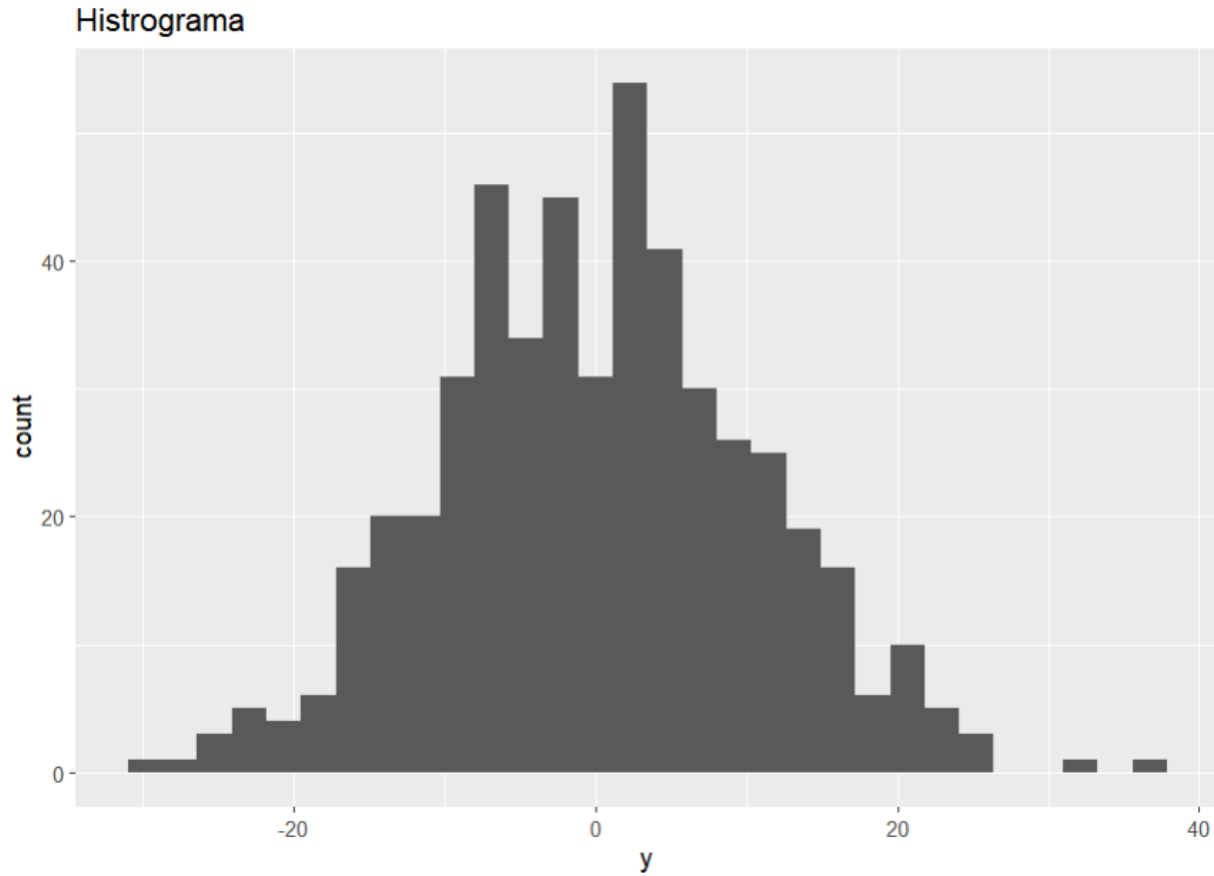
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(0, 10, 500)

plt.title("Histograma")
plt.hist(x)
plt.show()
```

NumPy, abreviação de Numerical Python, é uma biblioteca de código aberto destinada a realizar operações em arrays multidimensionais.

R x Python – Plotando gráficos



R x Python – Acesso a banco de dados

- Lendo os dados da tabela **NotaFiscal**.
- São importados os pacotes **DBI** e **RSQLite**.
- É necessário importar além do **pandas**, o pacote **sqlite3**.
- A variável **nota** no exemplo é um **DataFrame**.

```
library(DBI)
library(RSQLite)

con <- dbConnect(RSQLite::SQLite(), dbname =
"c:\\temp\\Financeiro.db")

nota <- dbGetQuery(con, "select * from
NotaFiscal;")

dbDisconnect(con)

head(nota)
```

```
import pandas as pd
import sqlite3

con =
sqlite3.connect("c:\\temp\\Financeiro.db")

nota = pd.read_sql_query("select * from
NotaFiscal;", con)

con.close()

nota.head()
```

Python

- Os objetos possuem métodos próprios.

```
texto = "BBAS3F"  
  
print(texto.endswith("G")) # False  
  
print(texto.endswith("F")) # True
```

Python

- Índices de vetores começam com **0**, sendo as **strings** tratadas como **vetores de caracteres**.

```
texto = "BBAS3F"

print(texto[0]) # Primeira letra, B

print(texto[2]) # Terceira letra, A

print(texto[-1]) # Última letra, F

print(texto[-6]) # Primeira letra, B
```


Python

- Referência parcial a objetos e concatenação.

```
texto = "BBAS3F"
```

```
print(texto[:2]) # Posição 0 a 1, BB
```

```
print(texto[2:]) # Posição 2 até o final, AS3F
```

```
print(texto[:2] + texto[2:]) # Texto inteiro, BBAS3F
```

Python

- Vetores em Python são guardados em listas.

```
x = [1, 7, 10, 5, 20]
print(x)                # [1, 7, 10, 5, 20]

print(x[-2:])           # Últimos 2 elementos, [5, 20]]

x = x + [12, 30, 25]
print(x)                # Concatenação, [1, 7, 10, 5, 20, 12, 30, 25]

print(sorted(x))        # Ordenação, [1, 5, 7, 10, 12, 20, 25, 30]

x[1] = 4
print(x)                # Modificação de valores
                        # [1, 4, 10, 5, 20, 12, 30, 25]

del x[3]
print(x)                # Remove valor
                        # [1, 4, 10, 20, 12, 30, 25]
```

Python

- Em Python, uma **matriz** pode ser representada como uma **lista de listas**, onde um elemento da lista contém uma linha da matriz, que por sua vez corresponde a uma lista com os elementos das colunas da matriz.

```
m = [[1, 2, 3], [4, 5, 6]] # Duas linhas e três colunas.  
print(m[0][0]) # Elemento da posição [1, 1], ou seja, 1.  
print(m[1][2]) # Elemento da posição [2, 3], ou seja, 6.
```

- O pacote **NumPy** facilita operações com **matrizes** e torna o seu processamento mais eficaz. Ele trabalha com um objeto de array especial de **N dimensões** chamado de **ndarray** (N-Dimensional Array), e possui diversas funções e métodos para sua manipulação.

```
import numpy as np  
m = np.array([[1, 2, 3], [4, 5, 6]])  
print(m[0, 0]) # Elemento da posição [1, 1], ou seja, 1.  
print(m[1, 2]) # Elemento da posição [2, 3], ou seja, 6.
```