Básico de Python

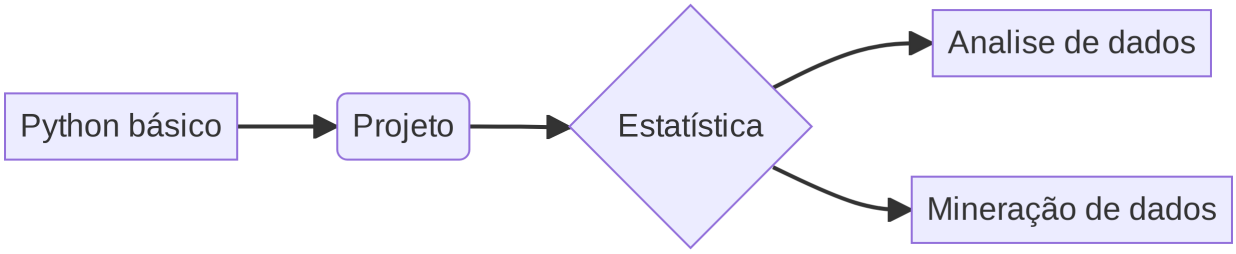
Sergio Pedro Rodrigues Oliveira

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 Diagrama de estudo



2 Variáveis e tipos de dados simples

2.1 `print()`

Print é uma função que exibe uma string na tela.

Exemplo:

```
print("string")
```

```
string
```

2.2 `print()` com variáveis

Podemos usar a função `print()` para imprimir uma variável string.

Exemplo:

```
message = "Hello world!"
print(message)
```

```
Hello world!
```

2.3 Regras de nomes de variáveis

Regras ou diretrizes para usar variáveis em Python.

- Nomes de variáveis deve conter apenas letras, números e underscores. Podemos começar a variável com letra ou underscore, mas nunca com um número.
- Espaços não são permitidos em nomes de variáveis, mas underscores podem ser usados para separar palavras.
- Evite usar palavras reservadas e nome de funções em Python como nome de variáveis.
- Nomes de variáveis devem ser concisos, porém descritivos.
- Tome cuidado ao usar a letra l e a letra maiuscula O, pois podem ser confundidas com os números 1 e 0.

2.4 Concatenando strings

Podemos usar o simbolo de (+) para combinar strings (concatenar).

Exemplo:

```
first_name = "ada"
last_name = "lovelace"
full_name = first_name + " " + last_name
print("Hello, " + full_name.title() + "!")
```

```
Hello, Ada Lovelace!
```

Os espaços em branco entre aspas servem para criar espaços na string.

2.5 Métodos auxiliares da função `print()`

1. `.title()`

Coloca apenas as primeiras letras em maiúsculas de cada palavra e o resto em minúscula.

Exemplo:

```
full_name = "ada lovelace"
print(full_name.title())
```

```
Ada Lovelace
```

2. `.upper()`

Coloca todas as letras em maiúsculas.

Exemplo:

```
full_name = "ada lovelace"
print(full_name.upper())
```

```
ADA LOVELACE
```

3. `.lower()`

Coloca todas as letras em minusculas. O método `.lower()` é particularmente útil para armazenar dados. Converter os dados em minúscula antes de armazenar.

Exemplo:

```
full_name = "ada lovelace"
print(full_name.lower())
```

```
ada lovelace
```

2.6 Caracteres de escape

Podemos inserir alguns caracteres de escape no texto para executar alguma ação, como pular linha, gerar tabulação e etc. Alguns caracteres podem ser vistos na Table 1.

Todos os caracteres de escape começam com barra(\) + complemento.

Table 1: Caracteres de escape

Caracteres de escape	Descrição
\t	Gera tabulação (tab).
\n	Gera quebra de linha.

Exemplo:

```
print("Language:\nPython\nJava\nC\nJavaScript")
```

```
Language:
Python
Java
C
JavaScript
```

2.7 Removendo espaços em branco `print()`

1. `.rstrip()`

Remove espaço em branco do lado direito.

Exemplo:

```
favorite_linguage = 'python '
favorite_linguage.rstrip()
```

```
'python'
```

2. `.lstrip()`

Remove espaço em branco do lado esquerdo.

Exemplo:

```
favorite_linguage = ' python'
favorite_linguage.lstrip()
```

```
'python'
```

3. `.strip()`

Remove os espaços em branco dos dois lados ao mesmo tempo.

Exemplo:

```
favorite_linguage = ' python '
favorite_linguage.strip()
```

```
'python'
```

- Os metodos usados não removem os espaços em branco em definitivo, para remover em definitivo é necessario armazenar o valor novo na variável.

```
favorite_linguage = ' python '
favorite_linguage = favorite_linguage.strip()
favorite_linguage
```

```
'python'
```

2.8 Números

A linguagem Python faz tipagem automática (dinâmica), tipa a variável de acordo com o uso. E o Python contém uma tipagem forte, não faz converção automática do tipo de uma variável para executar uma ação (operação).

Em resumo, python tem é uma linguagem de tipagem dinâmica e forte.

Os principais tipos de dados no Python são estão presentes na Table 2.

Table 2: Principais tipos de dados

Nome	Abreviação	Descrição
Inteiro	int	Números inteiros
Ponto flutuante	float	Números com ponto decimal

2.9 Funções de conversão de tipo

Podemos converte variáveis para determinado tipo especificado usando funções de conversão de tipo, como pode ser obeservado na Table 3.

Converter uma variável não é permanente, a não ser que a ação seja armazenada na variável explicitamente.

Table 3: Funções de conversão de tipo

Tipo para converter	Função	Descrição
int	`int()`	Converte variável para o tipo inteiro(int)
float	`float()`	Converte variável para o tipo float
string	`str()`	Converte variável para o tipo string

A função `str()` é deveras importante, pois pode auxiliar na função `print()`. A função `print()` só imprime na tela veriáveis string, sendo assim, precisamos converter as variáveis de outros tipos para string (pelo menos, momentaneamente), para comprir essa condição.

Exemplo:

```
age = 23
print("Happy " + str(age) + "rd Birthday!")
```

```
Happy 23rd Birthday!
```

2.10 Descobrindo o tipo da variável usando a função `type()`

Podemos usar a função `type()` para descobrir o tipo de determinada variável.

```
age = 23
print (type(age))
```

```
<class 'int'>
```

É uma **boa pratica** usar a função `type()`, para conferir o tipo da variável, antes de manipular alguma variável. Assim o programador terá o controle sobre as variáveis que esta trabalhando. Essa boa prática evita erros.

Também é uma **boa prática**, ao identificar/observar um erro, conferir os tipos das variáveis envolvidas. É um dos erros mais comuns: erro de tipagem.

2.11 Operações básicas

A Table 4 apresenta as principais operações básicas do python.

Table 4: Operações básicas

Operação	Símbolo	Exemplo
Soma	+	2+2=4
Subtração	-	3-2=1
Multiplicação	*	2*3=6
Divisão	/	5/4=1.25
Divisão inteira	//	5//4=1
Resto da divisão (módulo)	%	10%8=2
Potência	**	3**2=9
Raiz	**	4**0.5=2

2.12 Biblioteca `math` para ampliar operações matematicas

Podemos usar o pacote `math` para ampliar as funções matemáticas do Python (básicas, trigonométricas e estatísticas). A Table 5 apresenta as principais funções básicas da biblioteca `math`.

Table 5: Algumas operações da biblioteca `math`

Operação	Símbolo	Exemplo
Soma	`math.add(x,y)`	`math.add(2,2) = (2+2)=4`
Subtração	`math.subtract(x,y)`	`math.subtract(2,2) = (2-2)=0`
Raiz quadrada	`math.sqrt()`	`math.sqrt(4)=2`
Potência	`math.pow(x,y)`	`math.pow(2,3) = (2**3)=8`
Seno	`math.sin()`	`math.sin()`, retorna um ângulo em radianos.
Cosseno	`math.cos()`	`math.cos()`, retorna um ângulo em radianos.
Tangente	`math.tan()`	`math.tan()`, retorna um ângulo em radianos.
potencia de Euler	`math.exp(x)`	`math.exp(x) = math.pow(math.e**x)`
Logaritmo natural, ou log neperiano	`math.log(x)`	`math.log(2)=0.69`
Logaritmo	`math.log(x[,base])`	`math.log(2,10)=0.3`

Para converter o ângulo para radianos podemos usar a função `math.radians()`.

```
import math
#Seno do ângulo de 45°
#Resultado em Radianos
print(str(math.sin(math.radians(45))))
```

```
0.7071067811865475
```

Para converter de radiano para grau podemos usar a função `math.degrees()`.

```
import math
#Seno do ângulo de 45°
#Resultado em ângulo
print(str(math.degrees(math.sin(math.radians(45)))))
```

```
40.51423422706977
```

2.13 Operações lógicas básicas

A Table 6 apresenta as principais operações lógica básica do python. As operações lógicas retornam `True` ou `False`.

Table 6: Operações Lógicas

Operação	Nome	Função	Exemplo
==	Igual a	Varifica se um valor é igual ao outro.	1==1 = True
!=	Diferente de	Varifica se um valor é diferente ao outro.	1!=2 = True
>	Maior que	Varifica se um valor é maior que outro.	5>1 = True
>=	Maior ou igual	Varifica se um valor é maior ou igual a outro.	5>=5 = True
<	Menor que	Varifica se um valor é menor que outro.	1<5 = True
<=	Menor ou igual	Varifica se um valor é menor ou igual a outro.	1<=4 = True
and	E	Retorna True se ambas as afirmações forem verdadeiras.	(1==1) and (4<5)
or	Ou	Retorna True se uma das afirmações for verdadeiras.	(1==1) or (2<1) =
not	Negação	Retorna Falso se o resultado for verdadeiro, ou o contrario.	not (1==1) = Fals

2.14 Operadores de identidade

Os operadores de identidade, Table 7, são utilizados para comparar objetos, se os objetos testados referenciam o mesmo objeto.

Table 7: Operadores identidade

Operador	Definição
is	Retorna `True` se ambas as variáveis são o mesmo objeto.
is not	Retorna `True` se ambas as variáveis não são o mesmo objeto.

Exemplo de operações de identidade:

```
lista = [1,2,3]
outra_lista = [1,2,3]
recebe_lista = lista

print(f"São o mesmo objeto: {lista is outra_lista}")
```

```
São o mesmo objeto: False
```

```
lista = [1,2,3]
outra_lista = [1,2,3]
recebe_lista = lista

print(f"São o mesmo objeto: {lista is recebe_lista}")
```

```
São o mesmo objeto: True
```

2.15 Operações de associação

Os operadores de associação, Table 8, servem para verificar se determinado objeto esta **associado** ou **pertence** a determinada estrutura de dados.

Table 8: Operadores de associação

Operação	Função
in	Retorna `True` caso valor seja encontrado na sequência.
not in	Retorna `True` caso valor não seja encontrado na sequência.

Exemplos de operações de associação:

```
lista = ["Python", 'Academy', "Operadores", 'Condições']
print('Python' in lista)
```

```
True
```

```
lista = ["Python", 'Academy', "Operadores", 'Condições']
print('SQL' not in lista)
```

```
True
```

2.16 Comentários

Um comentário permite escrever notas em seus programas em liguagem natural. Em Python, o caractere sustenido (#) indica um comentário. Tudo que vier depois de um caractere sustenido en seu código será ignorado pelo interpretador Python.

Boas práticas em comentários:

1. Explicar o que o código deve fazer.
2. Como faz para funcionar.

2.17 Zen Python

É um guia de **boas práticas**.

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Principais pontos:

1. Bonito é melhor do que feio.
2. Simples é melhor que complexo.
3. Complexo é melhor que complicado.
4. Legibilidade conta.
5. Deve haver uma - e, de preferência, apenas uma - maneira óbvia de fazer algo.
6. Agora é melhor que nunca.

3 Listas

3.1 Lista

Uma lista é uma coleção de itens em uma ordem em particular. Os colchetes([]) indicam uma lista e os elementos individuais de uma lista são separados por vírgula. [ver 1, p. 71]

Exemplo:

```
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

3.2 Acessando elementos de uma lista

Podemos acessar a qualquer item de uma lista informando a posição, ou índice. As posições de uma lista começam no 0, e não no 1.

Para acessar um elemento de uma lista, informamos o nome nome da lista seguido do índice do item entre colchetes.

Exemplo:

```
#Acessando o primeiro item da lista
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles[0].title())
```

```
Trek
```

Para acessar a lista de trás pra frente podemos usar a posição invertida seguida do símbolo de menos na frente. Sendo assim, a posição do último item é -1, do penúltimo -2 e assim sucessivamente.

Exemplo:

```
#Acessando o último item da lista
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles[-1].title())
```

```
Specialized
```

3.3 Alterando, acrescentando e removendo elementos

Dado que a lista é um elemento dinâmico (pode, e provavelmente ocorrerá, de sofrer modificações com o uso), este tópico comentará os principais formas de modificação de listas.

3.3.1 Modificando elementos de uma lista

Para alterar um elemento que você quer modificar, use o nome da lista seguido do índice do elemento que quer modificar, e então forneça um novo valor.

```
#Alterando o item 1 da lista (índice 0)
motorcycles = ['honda','yamaha','suzuki']
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['ducati', 'yamaha', 'suzuki']
```

3.3.2 Acrescentando elementos em uma lista

Existem diversas formas de adicionar elementos a uma lista:

3.3.2.1 Concatenando elementos no final de uma lista, método `.append()`

Adiciona um novo elemento no final da lista usando o método `.append()`.

Exemplo:

```
#Adicionando elemento ao final da lista
motorcycles = ['honda','yamaha','suzuki']
motorcycles.append('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
```

3.3.2.2 Inserindo elementos em uma lista, método `.insert()`

Este método insere um elemento em determinada posição da lista, usando o método `.insert(índice,elemento)`.

Exemplo:

```
#Adicionando um item na segunda posição da lista (índice 1)
motorcycles = ['honda','yamaha','suzuki']
motorcycles.insert(1,'ducati')
print(motorcycles)
```

```
['honda', 'ducati', 'yamaha', 'suzuki']
```

3.3.3 Removendo elementos de uma lista

Os métodos para remover um item, ou um conjunto de itens, de uma lista.

3.3.3.1 Instrução `del`

Se a posição do item que você quer remover de uma lista for conhecida, a instrução `del` remove (deleta) um item em qualquer determinada posição. Depois de removido (deletado) não podemos mais acessar o valor, quando usado a instrução `del`.

```
# Remover (deletar) primeiro item da lista, indíce 0
motorcycles = ['honda','yamaha','suzuki']
print(motorcycles)

del motorcycles[0]
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

3.3.3.2 Método `.pop()`

Existem duas formas de usar o método `.pop()`:

1. `.pop()`

As vezes há necessidade de usar o valor de um item depois de removê-lo de uma lista. O método `.pop()` remove o **último** item de uma lista, mas permite que você trabalhe com esse item depois da remoção.

Remove o primeiro item de uma pilha, ou seja, o último item de uma lista.

Para usarmos o item removido é necessário, salva-lo numa variável.

Exemplo:

```
# Uso do método .pop()
# Removendo último item da lista e
# Trabalhando com o item removido.
motorcycles = ['honda','yamaha','suzuki']
print(motorcycles)

pop_motorcycle = motorcycles.pop()
```

```
print(motorcycles)
print(pop_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

2. `.pop(índice)`

Podemos usar o `.pop()` para remover um item em qualquer posição em uma lista, se incluirmos o índice do item que você deseja remove entre parênteses.

Exemplo:

```
# Uso do método .pop()
# Removendo o segundo item da lista e
# Trabalhando com o item removido.
motorcycles = ['honda','yamaha','suzuki']
print(motorcycles)

pop_motorcycle = motorcycles.pop(1)
print(motorcycles)
print(pop_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
yamaha
```

3.3.3.3 Método `.remove()`

Remove um item de acordo com o valor. É usado quando sabemos o valor do item, mas não a posição.

O método `.remove()` apaga apenas a primeira ocorrência do valor especificado. Para apagar mais de uma ocorrência será necessario o uso de um laço, para cada ocorrência.

Exemplo:

```
# Uso do método .remove()
# Removendo um item da lista pelo valor
motorcycles = ['honda','yamaha','ducati']
print(motorcycles)
```

```
too_expensive = 'ducati'
motorcycles.remove(too_expensive)
print(motorcycles)
```

```
['honda', 'yamaha', 'ducati']
['honda', 'yamaha']
```

3.4 Organizando uma lista

Dado que com frequência, as listas são organizadas numa ordem imprevisível, se torna necessario organizar as informações em uma ordem particular. O Python tem mecanismos para organizar listas. São eles:

3.4.1 Método `.sort()`

Ordena uma lista em ordem alfabética, ou alfabetica inversa.

Para ordenar uma lista em ordem alfabética inversa, basta passar o argumento `reverse = True` para o método `.sort()`.

Uma vez ordenada pelo método `.sort()` a lista não retorna a ordem original (ordenação permanente).

Exemplo:

```
# Ordenando a lista cars usando o método .sort()
cars = ['bmw','audi','toyota','subaru']
print(cars)
cars.sort()
print(cars)
cars.sort(reverse=True)
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
```

3.4.2 A função `sorted()`

A função `sorted()` ordena uma lista de forma temporaria, não altera a lista original, em ordem alfabetica. Ou seja, a lista volta a forma orginal ao final do uso da função.

Assim como no médodo `.sort()`, podemos ordenar a lista em ordem alfabética inversa adicionando o argumento `reverse=True`.

Exemplo:

```
# Ordenando temporariamente a lista cars usando a função sorted()
cars = ['bmw','audi','toyota','subaru']
print(cars)
print(sorted(cars))
print(sorted(cars,reverse=True))
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
['bmw', 'audi', 'toyota', 'subaru']
```

3.4.3 Método `.reverse()`

Para inverter a ordem original de uma lista, podemos usar o método `.reverse()`.

O método `.reverse()` não organiza a lista em ordem alfabética inversa, o método inverte a lista original.

O método `.reverse()` ordena de forma permanente a lista, porém se usarmos o método novamente, teremos a lista original. Logo, é fácil reverter o uso do método `.reverse()`.

Exemplo:

```
# Método .reverse() para inverte, de modo permanete, a ordem da lista.
cars = ['bmw','audi','toyota','subaru']
print(cars)
cars.reverse()
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
['bmw', 'audi', 'toyota', 'subaru']
```

3.5 Descobrindo o tamanho de uma lista - `len()`

Podemos descobrir o tamanho de uma lista usando a função `len()`.

Exemplo:

```
cars = ['bmw','audi','toyota','subaru']
len(cars)
```

```
4
```

4 Trabalhando com listas

4.1 Percorrendo uma lista inteira com um laço

Podemos usar um laço `for` para percorrer toda uma lista, podendo assim entre outras coisas, efetuar tarefas em cada item da lista.

A estutura básica do `for` é:

```
for variável_nova in lista :
  tarefas
```

O laço diz para a cada iteração pegar um elemento da lista e armazenar na nova variável, e executar uma tarefa a cada iteração. Toda tarefa indentada depois dos dois pontos é considerada dentro do laço.

No Python o `for`, usa indentação para determinar o que esta dentro do laço.

Qual quer linha após o laço que não for indentada é considerada fora do laço.

Exemplo:

```
#Executando um laço com base numa lista
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
  print(magician)
```

```
alice
david
carolina
```

4.2 Erros comuns de indentação

- Esquecer de indentar.
- Esquecer de indentar linhas adicionais do laço.
- Indentação desnecessaria.
- Indentando desnecessariamente após o laço.
- Esquecer os dois-pontos do laço `for`.

4.3 Listas numéricas

4.3.1 Gerando série de números com a função `range()`

A função `range()` é usada para gerar uma série de números, de uma determinada sequência numérica.

A função `range()` faz o Python começar a contar no primeiro valor definido (limite inferior) e parar quando atingir o segundo valor definido (limite superior). Como o for para no segundo valor, a saída não conterá o valor final. Também podemos definir um intervalo, pulando alguns valores.

Estrutura da função `range()`:

```
range(limite_inferior, limite_superior, intervalo)
```

Exemplo:

```
for value in range(1,5):
  print(value)
```

```
1
2
3
4
```

4.3.2 Usando `range()` para gerar uma lista - `list()`

Podemos usar para criar uma lista de números, combinando a função `range()`, que gera uma série númerica, com a função `list()`, que cria um lista.

Exemplo:

```
numbers = list(range(1,6))
print(numbers)
```

```
[1, 2, 3, 4, 5]
```

Exemplo 2:

```
numbers = list(range(2,11,2))
print(numbers)
```

```
[2, 4, 6, 8, 10]
```

4.3.3 Estatística simples com lista de números

As principais funções estatísticas estão contidas na Table 9.

Table 9: Estatística simples

Funções	Descrição
`min()`	Retorna o valor mínimo.
`max()`	Retorna o valor máximo.
`sum()`	Somatório.

As principais bibliotecas auxiliares de funções estatísticas são:

1. `math`
2. `numpy` as np
3. `statistics`
4. `Pandas` as pd

Medidas de posição utilizando bibliotecas python, Table 10.

Table 10: Medidas de posição, bibliotecas python

Funções	Descrição
`np.mean()`	Média aritmética
`statistics.median()`	Mediana
`statistics.mode()`	Moda
`np.quantiles(array, 0.5)`	Quartil
`np.percentile(array, 50)`	Percentil

Medidas de dispersão utilizando bibliotecas python, Table 11.

Table 11: Medidas de dispersão, bibliotecas python

Funções	Descrição
`pd.var()`	Variância
`pd.std()`	Desvio-padrão
`pd.mad()`	Desvio absoluto
`pd.cov()`	Covariância
`pd.corr()`	Correlação

4.4 list comprehensions

List comprehensions é uma forma de criar listas já acoplando o laço for nelas, deixando o código mais enxuto.

Sintaxe:

```
nome_lista = [expressão_calculada_do_for for variável in range()]
```

Exemplo:

```
squares = [value ** 2 for value in range(1,11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

4.5 Trabalhando com parte de uma lista

Neste tópico vamos trabalhar com um grupo de itens de uma lista, no Python é chamado de *fatia* (de uma lista).

4.5.1 Fatiando uma lista

1. Fatia simples

Para criar uma fatia, especifique o índice do primeiro e o último elemento com os quais você deseja trabalhar.

O Python para em um item antes do segundo índice (índice final) especificado.

Exemplo:

```
#Exibindo os 3 primeiros elementos de uma lista.
players = ["charles","martina","michael","florence","eli"]
print(players[0:3])
#Serão exibidos os itens na posição 0, 1 e 2.
```

```
['charles', 'martina', 'michael']
```

2. Delimitando ínicio e fim da fatia.

Podemos começar de qualquer índice.

Exemplo:

```
#Exibindo do segundo ao quarto item.
players = ["charles","martina","michael","florence","eli"]
print(players[1:4])
```

```
['martina', 'michael', 'florence']
```

3. Omitindo índices

Se omitirmos o primeiro índice, o Python começará do índice 0 (ínicio). De maneira analóga, se omitirmos o segundo índice (índice final), o Python terminará no último item.

Exemplo:

```
#Exibindo os 2 primeiros elementos de uma lista.
players = ["charles","martina","michael","florence","eli"]
print(players[:2])
```

```
['charles', 'martina']
```

4. índice negativo

O índice negativo devolve um elemento a determina distância do final da lista. Assim podemos exibir qualquer fatia a partir do final da lista.

Exemplo:

```
#Exibindo os 3 últimos elementos de uma lista.
players = ["charles","martina","michael","florence","eli"]
print(players[-3:])
```

```
['michael', 'florence', 'eli']
```

4.5.2 Percorrendo uma fatia com um laço - `for`

Podemos usar uma fatia em um laço `for` se quisermos percorrer um subconjunto de elementos de uma lista.

Exemplo:

```
players = ["charles","martina","michael","florence","eli"]
print("Here are the first three players on my team:")
for player in players[:3]:
  print(player.title())
```

```
Here are the first three players on my team:
Charles
Martina
Michael
```

4.5.3 Copiando uma lista

Vamos explorar o modo de copiar uma lista e analisar uma situação em que copiar uma lista é útil.

1. Copiando uma lista inteira, usando *fatia*.

Podemos criar uma fatia que inclua a lista inteira, omitindo o primeiro e segundo índices.

Exemplo:

```
#Usamos o metódo de fatia para copiar listas.
my_foods = ["pizza","falafel","carrot cake"]
friend_foods = my_foods[:]

print("My favorite food are:")
print(my_foods)

print("\nMy friend's favorite food are:")
print(friend_foods)
```

```
My favorite food are:
['pizza', 'falafel', 'carrot cake']

My friend's favorite food are:
['pizza', 'falafel', 'carrot cake']
```

Ambas as listas `my_foods` e `friend_fooods`, contém os mesmos elementos, porém são listas diferentes. Ao modificarmos uma delas a outra não é modificada automáticamente, por serem listas diferentes.

2. Variáveis que apontam para mesma lista.

Se ao invés de copiarmos uma fatia de uma lista para a outra, mesmo que seja a lista inteira, definirmos que uma variável é igual a outra, nesse caso criamos duas variáveis que apontam para a mesma lista. Ou seja, se modificarmos qualquer uma das listas, a outra é automaticamente modificada, pois ambas são a mesma lista.

Exemplo:

```
#Ambas variáveis apontam para a mesma lista.
my_foods = ["pizza","falafel","carrot cake"]
friend_foods = my_foods

friend_foods.append("ice cream")

print("My favorite food are:")
print(my_foods)

print("\nMy friend's favorite food are:")
print(friend_foods)
```

```
My favorite food are:
['pizza', 'falafel', 'carrot cake', 'ice cream']

My friend's favorite food are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

4.6 Tuplas

Tuplas são listas em que os itens não são criadas para mudar (listas imutáveis).

4.6.1 Definindo uma tupla

Uma tupla se parece com uma lista, exceto por usar parênteses no lugar de colchetes.

Sintaxe:

```
tuplas = (valor_1,valor_2,valor_3,...)
```

Exibimos cada elemento de uma tupla com a mesma sintaxe que usamos para acessar elementos de uma lista.

Exemplo:

```
dimensions = (200,50)
print(dimensions[0])
```

```
200
```

Se tentarmos alterar algum elemento de uma tupla, será retornado um erro de tipo.

4.6.2 Percorrendo todos os valores de uma tupla com um laço

Podemos percorrer uma tupla usando um laço `for`, da mesma forma que uma lista.

Exemplo:

```
dimensions = (200,50)
for dimension in dimensions:
  print(dimension)
```

```
200
50
```

4.6.3 Sobrescrevendo uma tupla

Não é possível modificar os elementos de uma tupla. Retornaria um erro de tipo.

Esse tipo de operação não funcionaria:

`tupla[0] = valor_novo`

Porém é possível subrescrever a tupla, imputando novos valores a variável.

Exemplo:

```
#Sobrescrevendo uma tupla
dimensions = (200,50)
print("Original dimensions:")
for dimension in dimensions:
  print(dimension)

dimensions = (400,100)
print("\nModified dimensions:")
for dimension in dimensions:
  print(dimension)
```

```
Original dimensions:
200
50

Modified dimensions:
400
100
```

5 Estatística básica

5.1 Teoria

- Definição de Estatística:

  A Estatística de uma maneira geral compreende aos métodos científicos para COLETA, ORGANIZAÇÃO, RESUMO, APRESENTAÇÃO e ANÁLISE de Dados de Observação (Estudos ou Experimentos), obtidos em qualquer área de conhecimento. A finalidade é a de obter conclusões válidas para tomada de decisões.

  – Estatística Descritiva Parte responsável basicamente pela COLETA e SÍNTESE (Descrição) dos Dados em questão.

    Disponibiliza de técnicas para o alcance desses objetivos. Tais Dados podem ser provenientes de uma AMOSTRA ou POPULAÇÃO.

  – Estatística Inferencial

    É utilizada para tomada de decisões a respeito de uma população, em geral fazendo uso de dados de amostrais.

    Essas decisões são tomadas sob condições de INCERTEZA, por isso faz-se necessário o uso da TEORIA DA PROBABILIDADE.

- O fluxograma da estatística descritiva pode ser espresso da seguinte forma:

Figure 1: Estatística descritiva.

- A representação tabular (Tabelas de Distribuição de Frequências) deve conter:

  – Cabeçalho

    Deve conter o suficiente para que as seguintes perguntas sejam respondidas "**o que?**" (Relativo ao fato), "**onde?**" (Relativo ao lugar) e "**quando?**" (Correspondente à época).

– Corpo

É o lugar da Tabela onde os dados serão registrados. Apresenta colunas e sub colunas.

– Rodapé

Local destinado à outras informações pertinentes, por exemplo a Fonte dos Dados.

- População e Amostras

  – População

  É o conjunto de todos os itens, objetos ou pessoas sob consideração, os quais possuem pelo menos uma característica (Variável) em comum. Os elementos pertencentes à uma População são denominados "Unidades Amostrais".

  – Amostras

  É qualquer subconjunto (não vazio) da População. É extraída conforme regras pré-estabelecidas, com a finalidade de obter "estimativa" de alguma característica da População.

- Tipos de variáveis

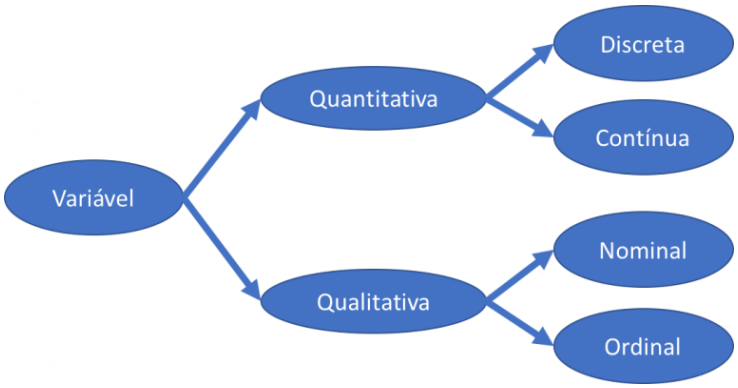


Figure 2: Tipos de variáveis.

– *Qualitativo nominal*

Não possuem uma ordem natural de ocorrência.

– *Qualitativo ordinal*

Possuem uma ordem natural de ocorrência.

– *Quantitativo descreta*

Só podem assumir valores inteiros, pertencentes a um conjunto finito ou enumerável.

– *Quantitativo continua*

  Podem assumir qualquer valor em um determinado intervalo da reta dos números reais.

5.2 Preparação dos dados (sumariazar dados coletados)

- Frequência (conceito)

  É a quantidade de vezes que um valor é observado dentro de um conjunto de dado.

- Distribuição em frequências

  - A distribuição tabular é denominada: “Tabela de Distribuição de Frequências”.
  - Podemos separar em 3 modelos de distribuição tabular:
    * Variável Quantitativa Discreta.
    * Variável Quantitativa Contínua.
    * Variáveis Qualitativas.

5.2.1 Variável Quantitativa Discreta

- Passos da preparação dos dados:
  - 1º Passo - **DADOS BRUTOS**:
    Obter os dados da maneira que foram coletados.
  - 2º Passo - **ROL**:
    Organizar os DADOS BRUTOS em uma determinada ordem (crescente ou decrescente).
  - 3º Passo - **CONSTRUÇÃO TABELA**:
    Na primeira coluna são colocados os valores da variável, e nas demais as respectivas frequências.
  - Frequência absoluta simples.
    Nº de vezes que cada valor da variável se repete.
- Principais campos da **distribuição tabular de variaveis quantitativas discreta**:
  - n é o número total de elementos da amostra.
  - x_i é o número de valores distintos que a variavel assume.
  - F_i é a Frequência Absoluta Simples.
  - f_i é a Frequência Relativa Simples.
  - $f_i\%$ é a Frequência Relativa Simples Percentual. $f_i\% = f_i \cdot 100\%$.
  - F_a é a Frequência Absoluta Acumulada.

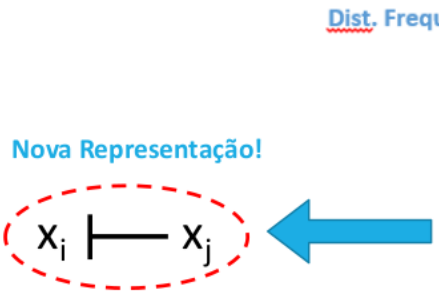
xi	Fi	fi	fi%	Fa↓	Fa↑	fa↓	fa↑
0	6	0,2	20	6	30	0,2	1
1	11	0,37	37	17	24	0,57	0,8
2	8	0,27	27	25	13	0,84	0,43
3	2	0,07	7	27	5	0,91	0,16
4	2	0,06	6	29	3	0,97	0,09
6	1	0,03	3	30	1	1	0,03
Total	30	1	100	-	-	-	-

Figure 3: Distribuição tabular quantitativa discreta.

Observação:
As setas simbolizam ordem crescente ou decrescente.

5.2.2 Variável Quantitativa Contínua

- Teoria:
  - A construção da representação tabular é realizada de maneira análoga ao caso das variáveis discretas.
  - As frequências são agrupadas em classes, denominadas de "Classes de Frequência".
  - Denominada "Distribuição de Frequências em Classes" ou "Distribuição em Frequências Agrupadas".



Dist. Frequências "X ~ Nº de Acidentes por dia, na BR 101, Setembro de 2015

xi	Fi	fi	fi%	Fa↓	Fa↑	fa↓	fa↑
0	6	0,2	20	6	30	0,2	1
1	11	0,37	37	17	24	0,57	0,8
2	8	0,27	27	25	13	0,84	0,43
3	2	0,07	7	27	5	0,91	0,16
4	2	0,06	6	29	3	0,97	0,09
6	1	0,03	3	30	1	1	0,03
Total	30	1	100	-	-	-	-

Fonte: Governo Federal

Figure 4: Distribuição de frequências em classes.

- Convencionar o tipo de intervalo para as classes de frequência:

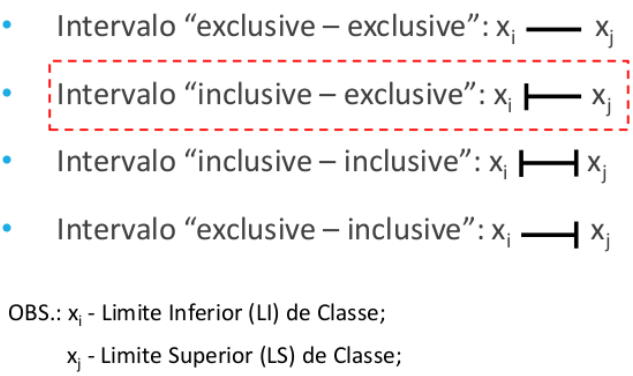


Figure 5: Intervalo de classes, distribuição de frequências quantitativa continua.

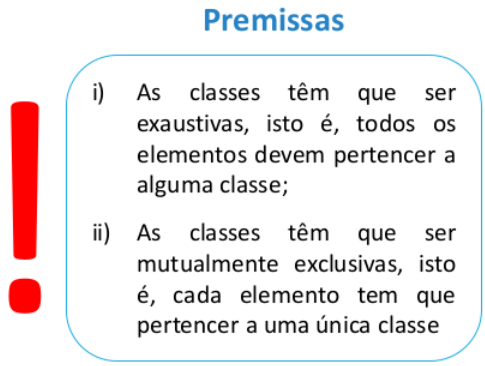


Figure 6: Distribuição frequências quantitativa continua, premissas.

Passos para contruir a **Tabela Distribuição de Frequências Contínua**:

1. Como estabelecer o **número de classes** (k):

- Normalmente varia de 5 a 20 classes.

- Critério fórmula de Sturges:

$$k \cong 1 + 3,3 \cdot \log(n)$$

- Critério da Raiz quadrada:

$$k \cong \sqrt{n}$$

  Onde n é o número de elementos amostrais.

2. Como calcular a **Amplitude Total** (AT_x):

- Diferença entre o maior e o menor valor observado.
- Intervalo de variação dos valores observados.
- Aproximar valor calculado para múltiplo do nº classes (k).
- Garantir inclusão dos valores mínimo e máximo.
- Cálculo:

$$AT_x = Mx(X_i) - Mn(X_i)$$

Onde,

AT_x é a Amplitude Total.

$Mx(X_i)$ é o *valor máximo das amostras.*

$Min(X_i)$ é o *valor mínimo das amostras.*

- Exemplo:

Se $k = 5$,

$AT_x = 28$

Logo, arredondando $AT_x = 30$, para aproximar o valor AT_x de um múltiplo de k.

3. Como cálcular a **Amplitude das classes da frequência** (h):

- As classes terão amplitudes iguais.
- Cálculo:

$$h = h_i = \frac{AT_x}{k}$$

Onde, k é o **número de classes** e AT_x é a **Amplitude Total**.

4. Como determinar o ponto médio das classes, representatividade da classe (p_i):

$$p_i = \frac{(LS_i - LI_i)}{2}$$

Onde,
LS_i é o limite superior da classe.
LI_i é o limite inferior da classe.

5. Passos da preparação dos dados:

- 1º Passo - **DADOS BRUTOS**: Obter os dados da maneira que foram coletados.
- 2º Passo - **ROL**: Organizar os DADOS BRUTOS em uma determinada ordem (crescente ou decrescente).
- 3º Passo - **CONSTRUÇÃO TABELA**: Na primeira coluna são colocados as classes, e nas demais as respectivas frequências.
- Exemplo:

Nº Classe	Classes (xi)	Fi	fi	fi%	Fa↓	Fa↑	fa↓	fa↑	fa↓%	pi
1	45 |--- 52	3	0,08	8	3	40	0,08	1	100	48,5
2	52 |--- 59	7	0,18	18	10	37	0,26	0,92	92	55,5
3	59 |--- 66	11	0,28	28	21	30	0,53	0,75	75	62,5
4	66 |--- 73	10	0,25	25	31	19	0,78	0,47	47	69,5
5	73 |--- 80	4	0,10	10	35	9	0,88	0,22	22	76,5
6	80 |--- 87	4	0,10	10	39	5	0,98	0,12	12	83,5
7	87 |--- 94	1	0,02	2	40	1	1,00	0,02	2	90,5
Total		40	1,00	100	-	-	-	-		-

Fonte: Dados Fictícios

Figure 7: Tabela de_distribuição de frequência quantitativa continua.

X_i são as classes.
F_i é a Frequência Absoluta Simples.
f_i é a Frequência Relativa Simples.
$f_i\%$ é a Fequência Relativa Simples Percentual.
F_a é a Frequência Absoluta Acumulada.
f_a é a Fequência Absoluta Acumulada Simples.
$f_a\%$ é a Fequência Absoluta Acumulada Simples Percentual.
p_i é a Representatividade da classe (ponto médio das classes).

5.2.3 Variáveis Qualitativas

- Passos da preparação dos dados:
  - Análogo ao procedimento para dados discretos.
  - 1º Passo - **DADOS BRUTOS**: Obter os dados da maneira que foram coletados.
  - 2º Passo - **ROL**: Nesse caso é feita organização dos DADOS BRUTOS em ordem (Crescente ou Decrescente) de importância.
  - 3º Passo - **CONSTRUÇÃO TABELA** (Com duas ou mais colunas).
- Distribuição de Frequencia:
  - x_i é o número de valores distintos que a variavel assume.
  - F_i é a Frequência Absoluta Simples.
  - f_i é a Frequência Relativa Simples.
  - $f_i\%$ é a Fequência Relativa Simples Percentual.
  - Inserir comentário sobre os dados.

5.3 Medidas de posição

6 Instruções IF

Bibliografia

[1] Eric Matthes. *Curso Intensivo de Python - 3ª Edição: Uma Introdução Prática e Baseada em Projetos à Programação*. Novatec Editora, 2023. ISBN: 9788575228432. URL: https://books.google.com.br/books?id=mkW7EAAAQBAJ.