💜

Basic Python for Data Science

Características importantes da Linguagem:

→ **Sintaxe Simples e Legível** – Código limpo e fácil de entender.

→ **Multiparadigmas de Programação** – Suporta programação procedural, orientada a objetos e funcional.

- **Programação Procedural** – Baseada em funções e procedimentos que executam tarefas sequencialmente. Exemplo: `def calcular_soma(a, b): return a + b`
- **Programação Orientada a Objetos (POO)** – Organiza o código em classes e objetos, permitindo reutilização e modularidade. Exemplo: `class Carro: def __init__(self, modelo): self.modelo = modelo`
- **Programação Funcional** – Baseada em funções que podem ser passadas como argumentos, retornadas por outras funções e armazenadas em variáveis. Exemplo: `map(lambda x: x * 2, [1, 2, 3])`

Reading and Writing CSV files

→ Dataframe X Dataset

- Um **DataFrame** é uma estrutura de dados bidimensional usada em bibliotecas como Pandas (Python) e R, onde os dados são organizados em linhas e colunas, semelhante a uma tabela. Ele permite manipulações como filtragem, agregação e junção de dados.
- Já um **Dataset** é um conceito mais amplo, referindo-se a qualquer conjunto de dados coletados para análise. Um dataset pode estar em diversos formatos, como tabelas, bancos de dados, arquivos CSV, JSON, entre outros.

→ Exemplos de principais funções:

```
#Função para ver quais chaves
dataframe[0].keys()

#Função para calcular a soma
sum(float(d["cty"]) for d in
```

→ **Dinamicamente Tipada** – Não exige declaração explícita de tipos de variáveis.

- Em compensação, ocupa mais espaço de memória.

→ **Interpretada** – Executa código linha por linha, sem necessidade de compilação.

→ **Extensível e Embutível** – Pode ser integrada com outras linguagens como C/C++.

→ **Bibliotecas Poderosas** – Grande ecossistema para data science, web, automação, etc.

- Apesar disso, possui muitas **built-in function** (função embutida) já.

→ **Portabilidade** – Código pode rodar em diferentes sistemas operacionais sem modificações.

Vocabulário interessante:

- **Paradigmas de Programação**
- **Built-in function** (função embutida)

Funções:

→ Declaração de uma função (`def`) e definição de parâmetros de diferentes tipos.

```
#Função para verificar todos
cilindros = set(d['cyl'] for
```

Advanced Python Objects, map():

→ **Principais conceitos da POO**:

1. **Classe**: Um "molde" para criar objetos. Define quais características (atributos) e ações (métodos) um objeto terá.
2. **Objeto**: Uma instância (cópia) de uma classe. Cada objeto pode ter valores diferentes para seus atributos.
3. **Atributos**: São as características dos objetos, como nome, idade ou localização.
4. **Métodos**: São funções dentro da classe que definem comportamentos dos objetos.
5. **Encapsulamento**: A ideia de esconder certos detalhes internos do objeto e permitir acesso apenas por métodos específicos.
6. **Herança**: Permite criar uma nova classe baseada em outra, herdando seus atributos e métodos.
7. **Polimorfismo**: Permite que diferentes classes tenham

```
def add_numbers(x, y, z=None,
    if (flag):
        print('Flag is true!'
    if (z==None):
        return x + y
    else:
        return x + y + z

print(add_numbers(1, 2, flag=

#Output: Flag is true! 3
```

→ Podemos declarar um variável que carregue a função

- **Facilitar o uso de funções** – Permite renomear funções longas para nomes curtos e mais fáceis de lembrar.
- **Criar funções dinâmicas** – Permite passar funções como argumentos para outras funções, facilitando a aplicação de transformações em dados.
- **Usar funções como objetos** – Python trata funções como objetos de primeira classe, então elas podem ser armazenadas em listas, dicionários e variáveis.

```
import numpy as np

# Atribuindo função da biblio
media = np.mean
```

métodos com o mesmo nome, mas com comportamentos distintos.

→ Definindo uma classe que possui variaveis ligadas a si e métodos (funções)

```
class Person:  # Criando um
a classe chamada "Person"
    department = 'School of
Information'  # Variável da
classe (todos os objetos te
rão isso)

    def set_name(self, new_
name):  # Método para defin
ir o nome da pessoa
        self.name = new_nam
e  # Atributo "name" recebe
um valor

    def set_location(self,
new_location):  # Método pa
ra definir a localização
        self.location = new
_location  # Atributo "loca
tion" recebe um valor
```

→ Definindo um objeto e preenchendo todos os campos dentro indicados pela sua classe

```
person = Person()  # Criamo
s um objeto da classe Perso
n
person.set_name('Christophe
```

```
dados = [10, 20, 30, 40]
print(media(dados))

#Output: 25
```

Sequências e Tipos:

→ Checagem de tipos a partir da função `type( )`

Tuple

→ Sequência de variáveis não mutável

```
x = (1, 'a', 2, 'b')
type(x)
```

List

→ Sequência de variáveis mutável

```
x = [1, 'a', 2, 'b']
type(x)
```

→ Funções importantes:

```
#Função que adiciona elemento
x.append(elemento que se dese

#Funções para ler a lista
for item in x:
    print(item)

i = 0
while(i != len(x)):
    print(x[i])
```

```
r Brooks')  # Definimos o n
ome da pessoa
person.set_location('Ann Ar
bor, MI, USA')  # Definimos
a localização da pessoa

print('{} live in {} and wo
rks in the department {}'.f
ormat(person.name, person.l
ocation, person.departmen
t))

print(f'{person.name} live
in {person.location} and wo
rks in the department {pers
on.department}')
```

→ A função `map()` em Python é usada para aplicar uma função a cada item de um ou mais iteráveis (como listas ou tuplas) sem a necessidade de escrever loops explícitos. Ela retorna um **objeto iterável**, que pode ser convertido em uma lista, tupla ou iterado diretamente.

```
#Como funciona:
map(função, iterável1, iterá

#Exemplo: Acha o valor minimo
store1 = [10.00, 11.00, 12.34
store2 = [9.00, 11.10, 12.34,
cheapest = map(min, store1, s

#Exemplo:
people = ['Dr. Christopher Br
```

```
    i++

#Operações matemáticas com li
[1,2] + [3,4] #[1, 2, 3, 4]
[1] * 3 #[1, 1, 1]
1 in [1, 2, 3] #True

#Slicing
x = "This is a String"
print(x[0:2]) #Th
```

Dictionary

→ Associa chaves (keys) com valores (values)

→ Funções importantes:

```
#Criação de um Dictionary
meu_dict = {"nome": "Rebecka"
outro_dict = dict(cor="azul",

#Funções para ler listas
for name in x:
    print(x[name])

for email in x.values():
    print(email)

for name, email in x.items():
    print(name)
    print(email)
```

String:

```
def split_title_and_name(pers
    title = person.split( )[0
    lastname = person.split(
    return '{} {}'.format(tit

list(map(split_title_and_name

#Output: ['Dr. Brooks', 'Dr.
```

→ Função map( ) é essencial em Data Science, ajudando principalmente na limpeza de dados:

```
data = [' Python ', 'Data Sci
cleaned = list(map(str.strip,
print(cleaned)  # ['Python',
```

Advanced Python Lambda and List Comprehensions

→ Lambda é uma função anônima, ou seja, que não possui nome como a maioria das outras funções

```
#Declaração da função
my_function = lambda a, b, c

#Exemplo:
my_function(1, 2, 3) #Output:
```

→ Compressão de listas auxiliam para diminuir a quantidade de linhas de

→ List formada exclusivamente de caracteres

```
sales_record = {
'price': 3.24,
'num_items': 4,
'person': 'Chris'}

sales_statement = '{} bought

print(sales_statement.format(
```

código, deixar mais limpo e evitar a declaração de muitos loops

```
#Opção 1: Sem compressão de l
my_list = []
for number in range(0, 1000):
    if number % 2 == 0:
        my_list.append(number
my_list

#Opção 2: Com compressão de l
[expressão for variável in it
my_list = [number for number
my_list
```

- A primeira aparição (`number` antes do `for`) determina o valor que será colocado na lista.
- A segunda (`number` dentro do `for`) é a variável iteradora que percorre os valores.

→ Outro exemplo:

```
def times_tables():
    lst = []
    for i in range(10):
        for j in range (10):
            lst.append(i*j)
    return lst

times_tables() == [i*j for i
```