

TRYBE

Modulo IV – Ciência da Computação

Bloco 35: Introdução à Python e Raspagem de Dados da Web

Python para: ciência de dados, criação de aplicações web, automação de tarefas repetitivas, aplicativos desktop e até aplicações para dispositivos móveis.

1) Aprendendo Python

Introdução (O que é Python?)

Vantagens: simplicidade da sintaxe e do servidor web, libs inclusas, comunidade.

Terminal Interativo (REPL)

`$ python3` para ver o python instalado por default
Abre o terminal interativo funcionando com *Read-Eval-Print Loop*.

Comando básicos:

`>>> import antigravity`

(tipo de Hello world)

`>>> import this`

(manifesto Python – guia [PEP 8](#)).

Operações básicas

`# comentando`

`2 * 3 # saída: 6`

`2 + 3`

`3 / 2`

`square_root = 25 ** (1/2) # sem let/var/const`

`print(square_root + 1)`

`counter += 1`

`3 // 2 # saída: 1, arredonda resultado para baixo`

`3 / 2 # saída: 1.5`

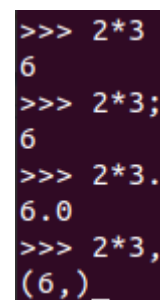
`'1' == 1 # saída: false, compara tipos`

`a == b # saída: true`

`and`

`5 <= idade or idade >= 65`

`18 < temperatura < 25`



```
>>> 2*3
6
>>> 2*3;
6
>>> 2*3.
6.0
>>> 2*3,
(6,)
```

Tipos de dados embutidos

Testar com `type(variável)`.

→ Números

inteiros **int**, fracionários **float**, **complex** (4j no lugar de i)

→ **str, bool**

→ **Listas**

list (listas, inclusive heterogêneas)

```
fruits = ["orange", "apple", "grape", "pineapple"]  
fruits[0] # o acesso é feito por índices iniciados em 0  
fruits[-1] # o acesso também pode ser negativo  
fruits.append("banana") # adicionando uma nova fruta  
fruits.remove("pineapple") # removendo uma fruta  
fruits.extend(["pear", "melon", "kiwi"]) # acrescenta uma lista de frutas a lista original  
fruits.index("apple") # retorna o índice onde a fruta está localizada, neste caso 1  
fruits.sort() # ordena a lista de frutas
```

tuple (list que não podem ser modificados durante a execução do programa)

```
user = ("Cássio", "Botaro", 42) # elementos são definidos separados por vírgula, envolvidos por  
parenteses  
user[0] # acesso também por índices
```

→ **Conjuntos**

set (conjuntos de elementos únicos e não ordenados)

```
permissions = {"member", "group"} # elementos separados por vírgula, envolvidos por chaves  
permissions.add("root") # adiciona um novo elemento ao conjunto  
permissions.add("member") # como o elemento já existe, nenhum novo item é adicionado ao  
conjunto  
permissions.union({"user"}) # retorna um conjunto resultado da união  
permissions.intersection({"user", "member"}) # retorna um conjunto resultante da intersecção dos  
conjuntos  
permissions.difference({"user"}) # retorna a diferença entre os dois conjuntos
```

frozenset (set que não podem ser modificados durante a execução do programa)

```
permissions = frozenset(["member", "group"]) # assim como o set, qualquer estrutura iterável pode  
ser utilizada para criar um frozenset  
permissions.union({"user"}) # novos conjuntos imutáveis podem ser criados à partir do original,  
mas o mesmo não pode ser modificado  
permissions.intersection({"user", "member"}) # retorna um conjunto resultante da intersecção dos  
conjuntos  
permissions.difference({"user"}) # retorna a diferença entre os dois conjuntos
```

→ **Objeto**

dict (associa chave e valor)

```
people_by_id = {1: "Cássio", 2: "João", 3: "Felipe"} # elementos no formato "chave: valor"  
separados por vírgula, envolvidos por chaves  
people_by_name = {"Cássio": 1, "João": 2, "Felipe": 3} # outro exemplo, dessa vez usando  
strings como chaves (ao contrário de JS, as aspas duplas são obrigatórias)
```

```
people_by_id[1] # saída: Cássio
del people_by_id[1] # elementos podem ser removidos com a palavra chave del
people_by_id.items() # dict_items([(1, "Cássio"), (2, "João"), (3, "Felipe")])
# um conjunto é retornado com tuplas contendo chaves e valores
```

→ Sequência

range

```
list(range(5)) # saída: [0, 1, 2, 3, 4], pois definimos apenas nº de parada
list(range(1, 6)) # saída: [1, 2, 3, 4, 5], começo e parada
list(range(1, 11, 2)) # saída: [1, 3, 5, 7, 9], começo, parada e passo
```

Estruturas condicionais

if, elif, else

```
position = ""
if salary <= 2000:
    position = "estagiário"
elif 2000 < salary <= 5800:
    position = "júnior"
elif 5800 < salary <= 7500:
    position = "pleno"
elif 7500 < salary <= 10500:
    position = "senior"
else:
    position = "líder"
```

: e **identação** (4 espaços) para substituir {} ou seja começo e final de bloco
; e **switch** não são mais úteis pois já tem legibilidade
dica de usar dict para deixar aninhamento de condicionais mais legível

Estruturas de repetição

for

for in - for each

Elementos iteráveis: list, str, tuple, set, dict

compreensão de listas

Para quando uma nova lista é criada como resultado de uma iteração, sem precisar de .append.

```
min_rating = 3.0
filtered_restaurants = [restaurant
                        for restaurant in restaurants
                        if restaurant["nota"] > min_rating]
print(filtered_restaurants)
# como hof filter
filtered_restaurants = [restaurant["name"]
                        for restaurant in restaurants
                        if restaurant["nota"] > min_rating]
# como hof map
```

while

Acontecerá enquanto a condição for satisfeita.

Funções

```
def nameFunction(x, y):  
    return
```

```
>>> def imc (peso, altura):  
...     return peso / (altura / 100) ** 2  
...  
>>> imc (100, 185)  
29.218407596785973  
>>> imc (peso=100, altura=185)  
29.218407596785973  
>>> █
```

Duas formas de chamar:

- **posicional**
- **nomeada**

Parâmetros também podem ser **variádicos** (variar quantidade). Como acessar:

- posicionais variádicos: como tuplas no interior de uma função
- nomeados variádicos: como dicionário.

→ **Escopo:**

Variáveis definidas dentro das funções tem escopo local, porém uma função quando não encontra um nome no escopo local irá procurar no espaço de nomes global.

→ *Nomear ou não dentro de parâmetros* pode mudar saída com algumas functions embutidas do Python:

```
len([1, 2, 3, 4]) # função len não aceita argumentos nomeados  
  
len(obj=[1, 2, 3, 4]) # este código irá falhar  
  
print("Botaro", "Cássio", ", ") # imprime Botaro Cássio ,  
  
print("Botaro", "Cássio", sep=", ") # nomeando o terceiro parâmetro, agora temos a saída: Botaro, Cássio
```

Configurar ambiente Python

([link referência](#))

- **pyenv para gerenciar versões;**

```
curl https://pyenv.run | bash
```

```
exec $SHELL
```

```
/ export PATH="$HOME/.pyenv/bin:$PATH"
```

```
eval "$(pyenv init -)"
```

```
eval "$(pyenv virtualenv-init -)" /
```

```
pyenv install -l
```

```
pyenv install 3.9.1
```

```
pyenv global 3.9.1
```

```
pyenv global
```

```
pyenv versions
```

- pyenv para gerenciar versões;

→ *name_snake_case.py* : convenção de nome para arquivo com python, considerado módulo.

- **pip** para gerenciar pacotes;

```
sudo apt install python3-pip
```

```
python3 -m pip --version
```

// resposta deve ser pip 19.2.3 from /usr/lib/python3.8/site-packages (python 3.8)

- **venv** para criar ambientes virtuais e isolar pacotes instalados e suas respectivas versões;

```
sudo apt install python3-venv
```

```
python3 -m venv -h
```

// resposta deve ser com infos de usage: arguments: etc

- **flake8** para formatar code segundo PEP8 e verificar sua complexidade ciclomática;

```
sudo python3 -m pip install flake8
```

```
python3 -m flake8 --version
```

- **black** como formatador automático intransigente de código;

```
sudo python3 -m pip install black
```

```
python3 -m black --version
```

Disparar com atalho *shift + ctrl + i*.

- **plugin Python no VSCode**;

instalar: *ctrl+P* - ext install ms-python.python

configurar: *ctrl+shift+p* - Preferences: Open Settings (JSON)

```
// ...
```

```
"python.linting.enabled": true,
```

```
"python.linting.flake8Enabled": true,
```

```
"python.formatting.blackArgs": [
```

```
"-l 79"
```

```
],
```

```
"python.formatting.provider": "black",
```

```
// ...
```

- **plugin CodeRunner no VSCode para rodar códigos em mais de 30 linguagens**;

instalar e logo configurar no mesmo Open Settings (JSON):

```
// ...
```

```
"code-runner.executorMap": {
```

```
"python": "python3 -u"
```

```
},
```

```
"code-runner.runInTerminal": true,
```

```
// ...
```

Disparar com *ctrl+alt+N*

Escrevendo os primeiros arquivos

→ **Convenções**

- passar duas linhas

- variáveis em maiúscula

→ **3 jeitos de importar** (sem precisar export)

import http # importa o pacote http como um módulo

from http import client # importa o módulo client do pacote http

from http.client import HTTP_POST # importa a constante HTTP_POST do módulo client do pacote http

Precaução para não printar todos prints do arquivo importado: **variável `__name__`** (interpretador Python identifica o arquivo executado, valor "`__main__`" quando invocamos um módulo como script).

→ **Rodar no terminal do VSCode**

`python3 name_module`

Para não ter que lembrar de tudo

`help("what-you-research")`

q para sair

exemplo: `abs`, `len`, `math`.

Dicas diversas

→ **jeitos de fazer template literals em Python**

- '{0}', seilá, {1}'.format('oi', 'quanto')

// vai imprimir oi, seilá, quanto

- f-string

f'{trecho de código aqui}, olaaa {outro trecho de código aqui}'

→ `==` versus `is` (`==` compara valores, `is` o que é)

→ Terminal python

ctrl + D para sair dele

```
In [125]: lista_a = [1, 2, 3]
In [126]: lista_b = [1, 2] + [3]
In [127]: lista_a
Out[127]: [1, 2, 3]
In [128]: lista_b
Out[128]: [1, 2, 3]
In [129]: lista_a = lista_b
Out[129]: True
In [130]: lista_a is lista_b
Out[130]: False
```

diferença `==` e `is`

2) Entrada e Saída de Dados

Input e output com Python.

Estruturando uma aplicação

Pacotes: módulos Python que contêm outros módulos e/ou pacote.

Ambiente Virtual

Gerir com venv:

`python3 -m venv .venv` (criar ambiente isolado com nome venv, na raiz do projeto);

`source .venv/bin/activate` (ativar ambiente isolado para usar ele);

`which python3` (verificar em qual ambiente estamos).

Entrada

Para saber interagir com user.

2 jeitos de pegar info externa:

→ **input**

input embutido, param opcional prompt, mensagem tipo texto.

```
input("Digite uma mensagem:")
```

→ **módulo sys**

sys.argv pega automaticamente os parâmetros passados.

```
import sys
```

```
if __name__ == "__main__":  
    for argument in sys.argv:  
        print("Received -> ", argument)
```

Comando

```
python3 arquivo.py 2 4 "teste"
```

Retorno

```
Received -> arquivo.py
```

```
Received -> 2
```

```
Received -> 4
```

```
Received -> teste
```

Saída

Para imprimir valor.

→ **print**

```
print("Na mesma", end="")  
print("linha.")
```

→ **sys**

sys.stderr com info do erro.

```
import sys
```

```
err = "Arquivo não encontrado"  
print(f"Erro aconteceu: {err}", file=sys.stderr)
```

Manipulação de arquivos

→ Write

```
personagens_legais.py > ...
1 characters_file = open("meus-personagens-favoritos.txt", mode="w")
2
3 characters_file.write("Tio Patinhas\n")
4 characters_file.write("Neo\n")
5 characters_file.write("Homem Aranha\n")
6
7 print("Batman", file=characters_file)
8
9 more_characters = ["Sonho\n", "Ash Ketchum\n"]
10
11 characters_file.writelines(more_characters)
12
13 characters_file.close()
```

gerir e ver:

```
python3 personagens_legais.py
cat meus-personagens-favoritos.txt
```

`file = open("name.format", mode="w")`

`file.write("nome idade\n")`

escrever no file via print:

`print("Something", file=file)`

`LINES = ["Alberto 35\n", "Betina 22\n", "João 42\n", "Victor 19\n"]`

`file.writelines(LINES)`

`.close()` - importante por limite de SO

→ Read

`file = open("arquivo.txt", mode="r")`

`content = file.read()`

`print(content)`

`file.close()`

Caso de arquivos binários: `mode="wb"` e `mode="rb"`.

Lidando com exceções

→ **Sintaxe**, como qualquer linguagem

→ **Exceptions** como `ZeroDivisionError`, `NameError` e `TypeError`.

Tratamento com try e except:

`while True:`

`try:`

`x = int(input("Please enter a number: "))`

`break`

`except ValueError:`

`print("Oops! That was no valid number. Try again...")`

Tratamento para arquivos, com else e finally:

`try:`

`arquivo = open("arquivo.txt", "r")`

`except OSError:`

`# será executado caso haja uma exceção`

`print("arquivo inexistente")`

`else:`

`# será executado se tudo ocorrer bem no try`

`print("arquivo manipulado e fechado com sucesso")`

`arquivo.close()`

`finally:`

`# será sempre executado, independentemente de erro`

`print("Tentativa de abrir arquivo")`

with (palavra reservada para gerenciamento de contexto) *garante que certas ações sejam tomadas independentemente se ocorreu ou não um erro naquele contexto*, como fechar arquivo:

```
# Criamos um contexto, limitando o escopo onde o arquivo está aberto.
# O uso do "as" aqui é somente para atribuir o valor utilizado no contexto à variável file
with open("arquivo.txt", "w") as file:
    file.write("Michelle 27\n")
# como estamos fora do contexto, o arquivo foi fechado
print(file.closed)
```

Manipulando arquivos JSON

Por suas vantagens (legibilidade e tamanho), JSON bem usado para troca de infos entre sistemas.

→ **json**

import json (suficiente pois é módulo embutido)

Principais métodos para manipulação: *load* , *loads* , *dump* , *dumps*.

Na **desserialização**, que faz a transformação de texto em formato JSON para Python:

- loads carrega o JSON a partir de um **texto**;
- load carrega o JSON a partir de um **arquivo**.

Na **serialização**, que é o caminho inverso, dumps para texto e dump para arquivo.

Outros módulos para serializar: [marshal](#) e [pickle](#).

```
import json # json é um modulo que vem embutido, porém precisamos importá-lo

with open("pokemons.json") as file:
    content = file.read() # leitura do arquivo
    pokemons = json.loads(content)["results"] # o conteúdo é transformado em estrutura python equivalente, dicionário neste caso.
    # acessamos a chave results que é onde contém nossa lista de pokemons

print(pokemons[0]) # imprime o primeiro pokemon da lista
```

```
import json

# leitura de todos os pokemons
with open("pokemons.json") as file:
    pokemons = json.load(file)["results"]

# separamos somente os do tipo grama
grass_type_pokemons = [
    pokemon for pokemon in pokemons if "Grass" in pokemon["type"]
]

# abre o arquivo para escrita
with open("grass_pokemons.json", "w") as file:
    # escreve no arquivo já transformando em formato json a estrutura
    json.dump(grass_type_pokemons, file)
```

Manipulando arquivos CSV

Formato CSV (*Comma Separated Values* – variações existem com ; e \t) comum em exportações de planilhas de dados e Bds.

→ [CSV](#)

Módulo com funções [reader](#) e [writer](#).

```
import csv
with open("balneabilidade.csv") as file:
    beach_status_reader = csv.reader(file, delimiter=";", quotechar='"') # dialeto padrão excel
    header, *data = beach_status_reader #trunque separando cabeçalho do restante
print(data)
```

→ **desempacotamento: head & tail**

```
a, b = "cd"
print(a) # saída: c
print(b) # saída: d

head, *tail = [1,2,3] # Quando um * está presente no desempacotamento, os valores são desempacotados em formato de lista.
print(head) # saída: 1
print(tail) # saída: [2, 3]
```

→ leitor e escritor baseado em dicionários: [.DictReader](#) & [.DictWriter](#), sem precisar manipular i

→ libs para análise da dados como [Pandas](#).

Dicas diversas

→ **HOFs no Python:**

- hof() que tem que envelopar dentro de list() para retornar bem o objeto

`nameMap = list(map(function, objectToIterate))`

`nameFilter = list(filter((lambda x: x < 0), range(-10,5)))`

- OU [compreensão de listas](#)

`nomes = [fruta['name'] for fruta in frutas]`

`caras = [fruta for fruta in frutas if fruta['preco'] > 4]`

3) Raspagem de Dados

Para extrair da web dados estruturados que possa analisar.

Introdução: definição

Técnica computacional para extrair dados provenientes de um serviço ou aplicação, e estruturar esses dados em BDs planilhas, ou outros formatos.

Passos: *realização de requisições, análise das resposta, extração dos dados, navegação do conteúdo, limpeza e armazenamento dos dados.*

1. Requisições web

Pode ser feito via pacote Python [urllib](#) lidando com HTTP, ou lib [requests](#), melhor leitura humana.

- sempre criar ambiente virtual antes de instalar lib
- instalar via `python3 -m pip install requests`
- fazer req de tipo get, post... e imprimir resultado.

!Alguns problemas possíveis na requisição

→ Rate Limit

Possibilidade de receber erro `status 429`, "*Too Many Requests*".

Boa prática: impor pausas entre reqs para nunca ficar fora do ar.

```
import requests
import time # agora parei de passar as duas linhas depois dos imports, apenas para o resumo
for _ in range(15):
    response = requests.get("https://www.cloudflare.com/rate-limit-test/") # Site limita 10 reqs/min
    print(response)
    time.sleep(6) # Coloca uma pausa de 6 segundos a cada requisição
```

→ Time Out

Resolver com tratamento de erro e def de tempo.

```
import requests
try:
    # recurso demora muito a responder
    response = requests.get("http://httpbin.org/delay/10", timeout=2)
except requests.ReadTimeout:
    # vamos fazer uma nova requisição
    response = requests.get("http://httpbin.org/delay/1", timeout=2)
finally:
    print(response.status_code)
```

2. Analisando respostas

Lib [parsel](#):

- instalar `python3 -m pip install parsel`
- usar

```
from parsel import Selector
import requests
```

```
response = requests.get("http://books.toscrape.com/")
selector = Selector(text=response.text)
print(selector)
```

```
# valores de css achados inspecionando a página
titles = selector.css(".product_pod h3 a::attr(title)").getall()
prices = selector.css(".product_price .price_color::text").getall()
```

```
for product in selector.css(".product_pod"):
```

```
title = product.css("h3 a::attr(title)").get()
price = product.css(".price_color::text").get()
print(title, price)
```

saber que também existe selector: [xpath](#)

! Casos que podem ser encontrados:

→ Recursos paginados

Precisamos poder buscar dados enquanto uma nova página for encontrada.

```
# Existe uma classe next, que podemos recuperar a url através do seu elemento âncora
next_page_url = selector.css(".next a::attr(href)").get()
print(next_page_url)
```

```
from parsel import Selector
import requests

# Define a primeira página como próxima a ter seu conteúdo recuperado
URL_BASE = "http://books.toscrape.com/catalogue/"
next_page_url = 'page-1.html'
while next_page_url:
    # Busca o conteúdo da próxima página
    response = requests.get(URL_BASE + next_page_url)
    selector = Selector(text=response.text)
    # Imprime os produtos de uma determinada página
    for product in selector.css(".product_pod"):
        title = product.css("h3 a::attr(title)").get()
        price = product.css(".price_color::text").get()
        print(title, price)
    # Descobre qual é a próxima página
    next_page_url = selector.css(".next a::attr(href)").get()
```

// dar page 1 como começo

// condicionar que enquanto tiver next, get.

→ Recursos obtidos à partir de outro recurso

Exemplo: raspar elemento que faça parte de tela de detalhes.

- investigar url dessa tela, capturar numa var href, `print(URL_BASE + href)`;
- investigar como pegar aquele elemento;
- no code, adicionar req e lógica de pegar elemento.

3. Limpeza de dados

Sujeiras para consertar antes de armazenar.

→ **regex** e substituir getall por **re**, get por **re_first**.

```
prices = selector.css(".product_price .price_color::text").re(r"£\d+\.\d{2}")
```

→ **Tirar sufixo** more sem perda acidental

"Fatiamos" a descrição removendo o sufixo

```
suffix = "...more"
```

```
if description.endswith(suffix):
```

```
    description = description[:-len(suffix)] #na v3.9, removesuffix , equivale palavra[:-len(suffix)]
```

```
print(description)
```

→ **Fatiamento**

Podemos acessar elementos de listas, tuplas e strings através de um **índice**.
Fatiamento serve para pegar subsequência.

```
[1, 2, 3][0] # Saída: 1
(1, 2, 3, 4)[:2] # Saída: (1, 2)
(1, 2, 3, 4)[1:] # Saída: (2, 3, 4)
"palavra"[3:7] # Saída: "avra"
[1, 2, 3, 4][1::2] # Saída: [2, 4]
```

4. Banco de Dados

→ **instalar** lib [pymongo](#).

```
$ python3 -m venv .venv && source .venv/bin/activate
```

```
$ python3 -m pip install pymongo
```

→ **conectar** com BD

```
from pymongo import MongoClient
```

```
# Por padrão o host é localhost e porta 27017
# Estes valores podem ser modificados passando uma URI
# client = MongoClient("mongodb://localhost:27017/")
client = MongoClient()
```

→ **acessar BD** e collections

```
from pymongo import MongoClient
```

```
client = MongoClient()
# o banco de dados catalogue será criado se não existir
db = client.catalogue
# a coleção books será criada se não existir
students = db.books
client.close() # fecha a conexão com o banco de dados
```

→ **adicionar** com [insert_one](#) ou [insert_many](#)

```
# book representa um dado obtido na raspagem
book = {
    "title": "A Light in the Attic",
}
document_id = db.books.insert_one(book).inserted_id #_id único é gerado e retornado.
print(document_id)
```

→ **buscar** com [find](#) ou [find_one](#)

```
for book in db.books.find({"title": {"$regex": "t"}}):
    print(book["title"])
```

→ **lembrar do with** para não precisar repetir client.close().

Dicas diversas

- [Site feito para treinar](#) raspagem de dados.
 - [Scrapy](#), ferramenta poderosa para raspagem de dados.
 - `_` é o último valor retornado, ni IPython (bom REPL)
 - [.strip](#) para arrancar espaço em branco
 - equivalente de npm install, instalar todas dependências de vez: `pip install -r requirements.txt`
-

4) Testes

Testes automatizados

Framework [pytest](#) :

- `python3 -m pip install pytest`
- `python3 -m pytest --version`

codigo.py

```
def is_odd(number):  
    'Retorna True se um número é verdadeiro, senão False.'  
    return number % 2 != 0
```

test_codigo.py

// arquivo com prefixo test_

```
from codigo import is_odd
```

// importando o testado

```
def test_is_odd_when_number_is_odd_returns_true():  
    'Para um número ímpar a função deve retornar o valor True'  
    assert is_odd(3) is True
```

// definição das funções de teste

```
def test_is_odd_when_number_is_even_returns_false():  
    'Para um número par a função deve retornar o valor True'  
    assert is_odd(2) is False
```

// assert: se a comparação entre resultado e expectativa for true, nada acontece, se for false, manda exceção do tipo AssertionError

- rodar com `python3 -m pytest`

Testando falhas

codigo.py

← Verificar que um **erro deve ocorrer**.

```
# ...  
  
def divide(a_number, other_number):  
    "Retorna a divisão de a_number por other_number"  
    return a_number / other_number
```

Params raises e match

test_codigo.py

// raises da pytest verifica se a exceção ocorreu

```
import pytest  
from codigo import is_odd, divide  
  
# ...  
  
def test_divide_when_other_number_is_zero_raises_an_exception():  
    with pytest.raises(ZeroDivisionError, match="division by zero"):  
        divide(2, 0)
```

// match pode receber regex e verifica se o texto retornado na exceção é o esperado

Um pouco de contexto

pytest.fixture para passar precondições ou estados necessários para a execução de um teste.

@pytest.fixture

```
def orders():
    """Nosso cenário (contexto) temos os seguintes pedidos"""
    return [
        {"customer": "maria", "order": "pizza", "day": "terça-feira"},
        {"customer": "joao", "order": "hamburger", "day": "terça-feira"},
        {"customer": "maria", "order": "pizza", "day": "quarta-feira"},
        {"customer": "maria", "order": "hamburger", "day": "quinta-feira"},
    ]

# estamos adicionando a fixture orders ao teste
# ou seja temos um contexto onde os pedidos são os definidos anteriormente
def test_get_most_ordered_dish_per_costumer_when_customer_is_maria(orders):
    assert get_most_ordered_dish_per_costumer(orders, "maria") == "pizza"

# novamente adicionamos um cenário (contexto) ao teste onde os pedidos realizados são os
# definidos na fixture
def test_get_order_frequency_per_costumer_when_customer_is_joao_and_order_is_pizza(orders):
    assert get_order_frequency_per_costumer(orders, "joao", "pizza") == 0

def test_get_order_frequency_per_costumer_when_customer_is_maria_and_order_is_hamburger(orders):
    assert get_order_frequency_per_costumer(orders, "maria", "hamburger") == 1
```

Dublês de teste

Para **simular recursos externos**.

Tipos:

- **fakes**: objetos que possuem implementações funcionais, porém normalmente simplificadas;
- **mocks**: são pré programados para verificarem as chamadas das funções que receberem;
- **stubs**: proveem respostas predefinidas;
- **spies**: são como stubs, mas também armazenam informações de como foram chamados.

Casos:

1/ substituir abertura de um arquivo por objeto que possui as suas implementações funcionais, read

```
from pokemon import retrieve_pokemons_by_type
from io import StringIO
```

```
def test_retrieve_pokemons_by_type():
    # definimos um pokemon do tipo grama
    grass_type_pokemon = {
        "national_number": "001",
        "evolution": None,
        "name": "Bulbasaur",
        "type": ["Grass", "Poison"],
        "total": 318,
        "hp": 45,
        "attack": 49,
        "defense": 49,
```



```

    "sp_atk": 65,
    "sp_def": 65,
    "speed": 45,
}
# definimos também um pokemon do tipo água
water_type_pokemon = {
    "national_number": "007",
    "evolution": None,
    "name": "Squirtle",
    "type": ["Water"],
    "total": 314,
    "hp": 44,
    "attack": 48,
    "defense": 65,
    "sp_atk": 50,
    "sp_def": 64,
    "speed": 43,
}
# criamos um arquivo em memória que o seu conteúdo são os dois pokemons
fake_file = StringIO(
    json.dumps({"results": [grass_type_pokemon, water_type_pokemon]})
)
# quando pesquisamos por pokemons do tipo grama,
# o pokemon do tipo água não deve ser retornado
assert retrieve_pokemons_by_type("Grass", fake_file) == [
    grass_type_pokemon
]

```

2/ Função esperando nome de arquivo e abertura do mesmo acontece dentro da função: substituir método open em tempo de execução por um objeto **mock_open**

```

import json
from unittest.mock import mock_open, patch
from pokemon import retrieve_pokemons_by_type

#mesma criação de 2 pokemons que acima

pokemon_json_content = json.dumps({"results": [grass_type_pokemon, water_type_pokemon]})
# substituímos a função padrão do python open por mock_open
# uma versão que se comporta de forma parecida, porém simulada
with patch("builtins.open", mock_open(read_data=pokemon_json_content)):
    # repare que o nome do arquivo não é importante aqui
    # a esses parâmetros não utilizados damos o nome de dummies
    # como neste contexto alteramos o open pelo mock_open,
    # o argumento "dummy" poderia ser substituído por qualqr coisa, já que não será utilizado pela função
    assert retrieve_pokemons_by_type("Grass", "dummy") == [
        grass_type_pokemon
    ]

```


Dicas diversas

→ **.fixture**: tudo o que vou precisar durante o teste. Aqui associado com function que retorna dados.

```
@pytest.fixture
def stock_data():
    with open('./tests/testdata/stock.json') as data_file:
        data = json.load(data_file)
    return data
```

// pytest permite integrar function nele, como stock_data que vai ser executada depois no teste

→ **decorador** para associar parâmetro ao testar a function

```
@pytest.mark.parametrize('stock_data', ['./tests/testdata/stock.json'])
def test_generate_report_actual_data(stock_data):
    # arrange
    # act
    report = generate_simple_report(stock_data)
    # assert
    assert report.splitlines()[0] == 'Data de fabricação mais antiga: 2019-09-13'
```

→ **Aspas triplas**: elas aceitam mais coisas dentro delas. Por exemplo, se você tiver um texto que contenha aspas simples e aspas duplas, vc não conseguiria escrever sem colocar um \ na frente das aspas usando strings normais.

→ **Tipos de testes**: lembrar de testar como um todo integração de funções e funcionalidade além de testes de unidade.

KAHOOT DE FIM DE BLOCO

Python é linguagem interpretada

if(num == 1) return true; como fica isso em Python?

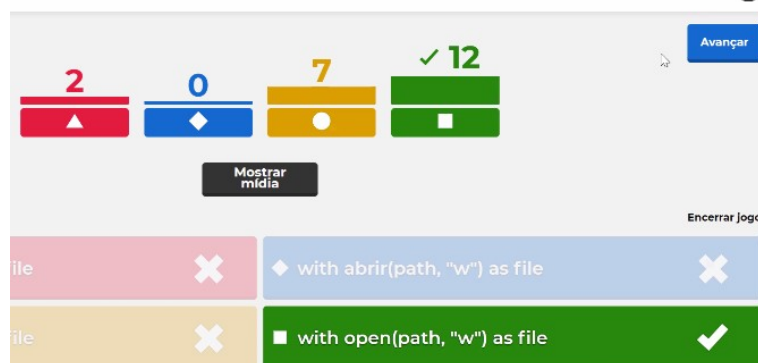


▲ if num == 1: return True



◆ if num == 1: return true

file = open(path, mode="w"), também pode ser:



file



◆ with abrir(path, "w") as file



file



■ with open(path, "w") as file



5) Projeto - Tech news

Dicas diversas do projeto

→ Ambiente

- Para deletar: ir no explorer, ctrl+H para ver arquivos escondidos, .venv aparece, pode tirar.
- Verificar, ao rodar testes, por exemplo em novo terminal, que ainda tem ambiente ativado.

```
juliette@juliette-HP-Laptop-15-dw0xxx:~/Documents/Curso Trybe/IV - CC/sd-05-tech-news$ python3 -m pytest
/usr/bin/python3: No module named pytest
juliette@juliette-HP-Laptop-15-dw0xxx:~/Documents/Curso Trybe/IV - CC/sd-05-tech-news$ ls
correct.csv          file_csv_update.csv  requirements.txt     tech_news
dev-requirements.txt pyproject.toml        setup.cfg            tech_news.egg-info
file_csv.csv         README.md             setup.py             tests
juliette@juliette-HP-Laptop-15-dw0xxx:~/Documents/Curso Trybe/IV - CC/sd-05-tech-news$ source .venv/bin/activate
(.venv) juliette@juliette-HP-Laptop-15-dw0xxx:~/Documents/Curso Trybe/IV - CC/sd-05-tech-news$ python3 -m pytest tests/test_collector/test_scrapper.py
===== test session starts =====
platform linux -- Python 3.8.5, pytest-6.1.2, py-1.10.0, pluggy-0.13.1 -- /home/
```

→ Testes

python3 -m pytest (todos testes)

python3 -m pytest -s -vv (ver mais precisamente)

python3 -m pytest tests/nomedoarquivo.py (testes de um arquivo)

python -m pytest tests/test_arquinox.py::nome_do_teste_x

(para testar especificamente o que estiver dentro do arquivo)

→ Rodar resultados de functions

python3 -i your-path.py

(para abrir um terminal Python do arquivo onde quer testar functions)

```
juliette@juliette-HP-Laptop-15-dw0xxx:~/Documents/Curso Trybe/IV - CC/sd-05-tech-news$ python3 -i tech_news/collector/scraper.py
>>> fetch_content("https://app.betrybe.com/")
'<!doctype html><html lang="pt-BR"><head><meta charset="utf-8"/><link rel="style
sheet" href="/assets/css/iso_bootstrap4.1.0.min.css"><link rel="stylesheet" href
="/assets/css/bootstrap-toc.min.css"><link rel="icon" href="/favicon.png"/><meta
name="viewport" content="width=device-width,initial-scale=1"/><meta name="theme
-color" content="#000000"/><meta name="description" content="Aprenda a programar
com uma formação de alta qualidade e só comece a pagar quando conseguir um bom
```

→ Rodar mongo junto (contra erros de connection)