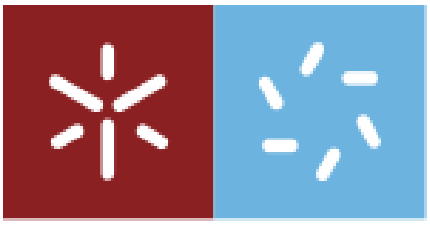

Universidade do Minho
Escola de Ciências

Relatório do Projeto 01

Aplicação de conhecimentos - Módulos requests/re e formatos de dados

UC: Ambientes e Linguagens de Programação para Ciência de Dados

Realizado por:

Ema Vasconcelos (a111111)
Mafalda Marialva (a110033)
Maria Rodrigues(a112026)

2025/2026

Índice

Introdução:

Com este projeto pretende-se realizar um Command Line Interface (CLI) que permita a um utilizador interagir e obter dados da REST API. Este trabalho prático (TP) tem como principais objetivos:

- aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases dentro de textos;

- desenvolver sistematicamente, a partir de ER, Processadores de Linguagens Regulares, ou Filtros de Texto (FT), que filtrem ou transformem textos com base no conceito de regras de produção Condição Ação;

- utilizar o módulo re do Python – com as suas funções de search(), split(), sub()– para implementar os FT pedidos;

- desenvolver conhecimentos sobre REST API e o módulo requests do Python.

O repositório utilizado foi:
https://github.com/EmaVasconcelos/ALPCD_TP1_GRUPO3

Configuração inicial

Começamos por importar todas as bibliotecas necessárias. Algumas delas são: requests (para fazer chamadas HTTP à API externa), typer (utilizado para criar interfaces de linhas de comando intuitivas) e re(para processar e limpar texto com a utilização de expressões regulares).
Estabelecemos também a BASE_URL que define onde na API vamos buscar os dados, a API_KEY única para os pedidos ao servidor e o header que evita que hajam erros ou bloqueios por parte do servidor.

```
import requests
import json
import re
import typer
import csv
from typing import Optional,List

app=typer.Typer()

BASE_URL="https://api.itjobs.pt/job/list.json"
API_KEY="0e1d382aea19663d13db0633833b9b40"

header= {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
}
```

Callback principal

Criamos um callback principal que permite que o script rode mesmo sem os subcomandos.

```
@app.callback(invoke_without_command=True)
def main():
        #Permite subcomandos explícitos (ex.: python jobsit.py top 5).
    pass
```

Listar os N trabalhos mais recentes do site

O comando top lista os N empregos mais recentes através de uma chamada à API.
Primeiro, colocamos os parâmetros de requisição, que incluem o limite de resultados e chave de autenticação. De seguida, enviamos a requisição GET para o servidor e caso esta seja bem sucedida (código 200) extraímos a lista de resultados e exibimos os títulos. Por fim, já incluindo a alínea e) que será apresentada no final, o utilizador é questionado se pretende guardar os resultados para csv.

```python
@app.command()
def top(n:int, export:Optional[str]=typer.Option(None,"--export","-e")):

    parametros={
        "limit":n,
        "api_key":API_KEY
    }

    response=requests.get(BASE_URL,params=parametros,headers=header)

    if response.status_code==200:


        jobs=response.json().get("results",[])
        print(f"\nTop {n} empregos mais recentes:")
        for job in jobs:
            print(f"{job.get('title','N/A')}")


        csv_export=typer.confirm("Deseja exportar para CSV?")
        if csv_export:
            export_to_csv(jobs,"recent_jobs.csv")

        else:
            typer.echo("Exportação cancelada.")

    else:
        typer.echo(f"Erro ao obter dados da API: {response.status_code}", err=True)
        raise typer.Exit(1)
```

Listar todos os trabalhos part-time com certas especificações

O comando search executa pesquisas filtradas, tendo em conta que o trabalho é part-time, a localização e a empresa.
O processo é muito parecido com o comando anterior, apenas tem mais parâmetros de forma a filtrar tudo. Após obter os resultados, valida-se se a localidade coincide com os locais de oferta. Limpamos também o resultado, removendo tags HTML e trocamos o texto para 400 caracteres de legibilidade utilizando expressões regulares.
Estruturamos os dados numa lista de dicionários e exibimos em formato JSON formatado.

```python
@app.command()
def search(localidade:str,empresa:str,n:int,export:Optional[str]=typer.Option(None,"--export","-e")):

    parametros={
        "limit":n,
        "api_key":API_KEY,
        "tipo":"part-time",
        "localizacao":localidade,
        "empresa":empresa
    }
```

```
    response=requests.get(BASE_URL,params=parametros,headers=header)
    if response.status_code==200:
        jobs=response.json().get("results",[])

        simple_jobs=[]
        for job in jobs:
            locs=[loc.get("name","") for loc in job.get("locations",[])]
            if not any(localidade.lower() in l.lower() for l in locs):
                continue


            desc=job.get("description","") or job.get("body","") or ""
            desc_clean=re.sub(r'<[^<]+?>','',desc).strip()
            simple_job={
                "titulo":job.get("title",""),
                "empresa":job.get("company",{}).get("name",""),
                "descricao":desc_clean[:400],
                "data_publicacao":job.get("publishedAt",""),
                "salario":job.get("wage",""),
                "localizacao":", ".join([loc.get("name","") for loc in job.get("locations",[])]),
            }
            simple_jobs.append(simple_job)

        typer.echo(json.dumps(simple_jobs,indent=2,ensure_ascii=False))
```

Mais uma vez damos a opção ao utilizador de exportar os dados para CSV. Ao testar, podemos colocar até Porto EmpresaY 3 , mas caso apenas haja só um trabalho part-time no Porto só vai aparecer esse.

Extrair o regime de trabalho

O comando `type`, responsável por identificar o regime de trabalho associado a um anúncio de emprego da API do itjobs.pt, a partir do seu identificador interno (job ID). O comando consulta a API, valida os dados recebidos e, através de expressões regulares aplicadas ao título, descrição e corpo do anúncio, classifica o regime como remoto, presencial, híbrido ou outro, apresentando o resultado diretamente no terminal.

```
@app.command()
def type(jobid: str):
    """Extrai o regime de trabalho (remoto/híbrido/presencial/outro) de um anúncio específico a partir job ID."""

    URL = "https://api.itjobs.pt/job/get.json"
    parametros = {"api_key":API_KEY,"id":jobid}

    response = requests.get(URL, params=parametros, headers=header)
```

Na primeira parte do código, o comando type recebe o job id como argumento e constrói o pedido à API. Para obter os dados do anúncio, é utilizada a função requests.get(), que passa como parâmetros a chave de autenticação e o id do anúncio. Ainda é responsável por enviar o pedido e guardar a resposta.

```
    if response.status_code != 200:
        typer.echo(f"Erro ao obter dados do job ID {jobid}: {response.status_code}", err=True)
        raise typer.Exit(1)

    job = response.json()

    if not job or ("title" not in job and "description" not in job and "body" not in job):
        typer.echo(f"Erro: job ID {jobid} inválido ou não encontrado.", err=True)
        raise typer.Exit(1)
```

De seguida, o programa valida se o pedido foi bem sucedido. Caso o status_code não for 200, o programa termina imediatamente e informa o utilizador do erro. É feita uma verificação ao conteúdo devolvido pela API, garantindo que o objeto contém pelo menos os campos title, description ou body. Se nenhum destes existir, o programa considera o job inválido e termina. Esta validação assegura que o comando só continua quando existem dados para analisar.

```
descricao = job.get("description","") or ""
corpo = job.get("body","") or ""
titulo=job.get("title","") or ""
texto = f"{titulo} {descricao} {corpo}"
texto_final = re.sub(r'<[^<]+?>', '', texto).lower().replace('\n', ' ').strip()

remoto = re.search(r"(remoto|remote|teletrabalho|home office)", texto_final)
presencial = re.search(r"(presencial|escritório|office)", texto_final)
hibrido = re.search(r"(híbrido|hybrid)", texto_final)
```

Nesta parte, o programa prepara o texto do anúncio e aplica expressões regulares para identificar o regime de trabalho. Primeiro, concatena o título, a descrição e o corpo do anúncio. Em seguida, remove eventuais tags HTML e normaliza o texto para minúsculas, o que torna a análise mais consistente. Depois da limpeza, são aplicadas expressões regulares para procurar palavras-chave que indicam o regime laboral.

```
if (remoto and presencial) or hibrido:
    regime = "híbrido"
elif remoto:
    regime = "remoto"
elif presencial:
    regime = "presencial"
else:
    regime = "outro"

typer.echo(f"Regime de trabalho: {regime}")
```

Por último, o programa decide o regime com base nas expressões encontradas: se aparecerem termos de remoto e presencial em simultâneo, ou se houver referência explícita a híbrido, o resultado é classificado como híbrido, caso contrário, é avaliado se é remoto ou presencial. Se não houver correspondência, devolve “outro”.

Contar ocorrências de skills

O comando skills faz uma requisição HTTP a uma API para que listar trabalhos que exigem uma determinada lista de skills dentro de um intervalo de datas (data_inicial e data_final). Caso a requisição seja bem sucedida, mostra o JSON no terminal e pergunta ao usuário se pretende exportar os resultados para CSV (export_to_csv).

O código começa por registar a função como um comando typer, que transforma a função skills em um comando de aplicação CLI. A função lista as strings e o intervalo de datas.

```
@app.command()
def skills(skills: List[str], data_inicial: str, data_final: str):
```

De seguida foi criado um dicionário com os parâmetros da requisição HTTP que inclui a chave da API (API_KEY), separa strings por vírgulas e contém a data inicial e a data final.

```
parametros = {
    "api_key": API_KEY,
    "skills": ",".join(skills),
    "data_inicial": data_inicial,
    "data_final": data_final
}
```

Faz uma requisição HTTP para a BASE_URL, usa o dicionário header, que foi definido no início do projeto e os parâmetros definidos acima.

```
response = requests.get(BASE_URL, headers=header, params=parametros)
```

Verifica se a resposta do status foi bem sucedida, decodifica a resposta em JSON, extrai os resultados, e imprime no terminal de forma legível.

```
if response.status_code == 200:
    jobs = response.json().get("results", [])
    print(json.dumps(jobs, indent=2))
```

Por fim, coloca a opção ao usuário se deseja exportar os resultados para CSV, caso a requisição falhar imprime uma mensagem de erro.

```
    csv_export=typer.confirm("Deseja exportar para CSV?")
    if csv_export:
        export_to_csv(jobs,"skill_jobs.csv")
else:
    print(f"Erro: {response.status_code}")
```

Com este código é possível contar a ocorrência de skills no formato JSON num determinado período de tempo e caso o utilizador pretenda guardar em formato CSV.

Possibilidade de exportar para csv

A função export_to_csv() converte os dados de empregos em ficheiros bem estruturados. Primeiro, define se os nomes das colunas que aparecerão no ficheiro final. Abre-se um ficheiro em modo de escrita com o encoding UTF-8 para conseguir "apanhar" os caracteres especiais. Itera-se sobre cada emprego e extrai-se os campos relevantes, aplicando limpeza de dados(por exemplo a remoção de HTML utilizando expressões regulares). No final confirma-se ao utilizador que a função foi bem sucedida e informa qual o nome do ficheiro.

```
def export_to_csv(jobs:list, filename:str):

    fieldnames=["titulo","empresa","descricao","data_publicacao","salario","localizacao"]

    with open(filename, mode='w', newline='', encoding='utf-8') as csvfile:
        writer=csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

        for job in jobs:
            row={
                "titulo": job.get("title", ""),
                "empresa": job.get("company", {}).get("name", ""),
                "descricao": re.sub('<[^<]+?>', '', job.get("description", "")),
                "data_publicacao": job.get("publishedAt", ""),
                "salario": job.get("wage", ""),
                "localizacao": ", ".join([loc.get("name", "") for loc in job.get("locations", [])])
            }
            writer.writerow(row)


    typer.echo(f"Empregos exportados para {filename}")
```

Conclusão:

Este projeto permitiu ao grupo compreender melhor o funcionamento de REST API's, aumentou a nossa capacidade de escrever expressões regulares e desenvolver sistematicamente os processadores de linguagens regulares, e perceber melhor como utilizar o módulo re do Python.