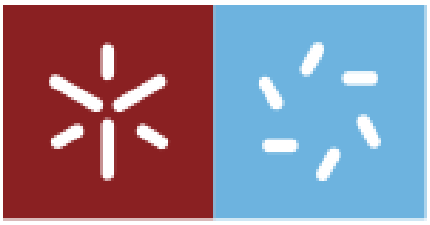

Universidade do Minho
Escola de Ciências

Relatório do Projeto

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

TP1

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
    }
```

```python
    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.

```python
@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}")
```

TP2

Introdução

Este trabalho prático tem como principais objetivos:

- Aplicar os conhecimentos adquiridos sobre Web Scraping para desenvolver, em Python, extratores de datasets a partir de sites recorrendo aos módulos beautifulsoup2 e requests;

- Reforçar a utilização do módulo re do Python para limpeza a pré-processamento dos dados.

O repositório utilizado foi o mesmo do TP1:
https://github.com/EmaVasconcelos/ALPCD_TP1_GRUPO3

Integração do itjobs.pt com o Teamlyzer

Na alínea a) foi desenvolvido o comando get, que recebe um jobID e devolve informação combinada do anúncio no itjobs.pt e da empresa correspondente no Teamlyzer. O objetivo é enriquecer os dados da API com rating, descrição, benefícios e salário médio da empresa.

A função começa por construir um pedido HTTP para a rota job/get.json da API do itjobs.pt, enviando a chave API_KEY e o jobID introduzido pelo utilizador. Se o código de resposta não for 200, o programa informa o erro e termina. Quando a resposta é válida, o JSON é convertido num dicionário Python e é extraído o nome da empresa através de job["company"]["name"]. Se o anúncio não tiver empresa, a função indica ao utilizador que não existe empresa associada a esse job.

```python
TEAMLYZER_HEADER = {"User-Agent": "Mozilla/5.0 (compatible; TeamlyzerScraper/1.0)"}

@app.command()
def get(jobid: str, export: bool = typer.Option(False, "--export", "-e")):
    """obter informações sobre um jobID e adicionar informações acerca da empresa."""

    params = {"api_key": API_KEY, "id": jobid}
    response = requests.get(URL, params=params, headers=header)

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

    job = response.json()
    company = job.get("company", {}).get("name", "").strip()
    if not company:
        typer.echo("Não existe nenhuma empresa relacionada a este anúncio.")
        raise typer.Exit(1)
```

Para aceder ao Teamlyzer, o nome da empresa é normalizado: é colocado em minúsculas e os espaços são trocados por hífen, formando um “slug” simples. Esse slug é usado para construir duas URLs: a página principal da empresa (https://pt.teamlyzer.com/companies/<empresa>) e a página de benefícios (https://pt.teamlyzer.com/companies/<empresa>/benefits-and-values). Em todos os pedidos é usado o cabeçalho User-Agent indicado no enunciado, para evitar bloqueios do site.

```python
nome_empresa = company.lower().replace(" ", "-")
company_url = f"https://pt.teamlyzer.com/companies/{nome_empresa}"

teamlyzer_rating = None
teamlyzer_description = None
teamlyzer_salary = None
teamlyzer_benefits = None
```

Com o módulo requests é feita uma chamada à página principal da empresa no Teamlyzer. O HTML é tratado com BeautifulSoup.

- O rating geral é obtido a partir de um elemento com classes associadas à classificação, sendo o texto guardado em teamlyzer_rating.
- A descrição da empresa é lida de um com classe ellipsis center_mobile, ficando em teamlyzer_description.
- Para o salário médio, o código procura a secção das reviews, encontra o ícone do euro e a partir daí localiza um bloco com o valor do salário. O texto é limpo e guardado em teamlyzer_salary.
  Se algum destes elementos não existir ou o pedido falhar, os respetivos campos são deixados a None e o utilizador é informado.

```
try:
    response_teamlyzer= requests.get(company_url, headers=TEAMLYZER_HEADER)
    response_teamlyzer.raise_for_status()

    soup = BeautifulSoup(response_teamlyzer.text, "html.parser")

    rating = ((soup.find("span", class_="text-center c_rating")) or
              (soup.find("span", class_="text-center aa_rating")))

    if rating:
        teamlyzer_rating = rating.get_text(strip=True)

    description = soup.find("div", class_="ellipsis center_mobile")
    if description:
        teamlyzer_description = description.get_text(strip=True)

    reviews = soup.find("div", class_="col-lg-12 box_background_style_overall voffset2")
    if reviews:
        salary_icone= reviews.find("i", class_="fa fa-eur")
        if salary_icone:
            box= salary_icone.find_parent("div", class_="panel mini-box")
            if box:
                salary_box = box.find("div", class_="box-info")
                if salary_box:
                    salary = salary_box.find("p", class_="size-h2")
                    if salary:
                        teamlyzer_salary = " ".join(salary.get_text(strip=True).split()[:3])

except requests.RequestException:
    typer.echo("Erro ao aceder ao Teamlyzer.", err=True)
    raise typer.Exit(1)
```

Numa segunda fase é feita uma requisição à página benefits-and-values da mesma empresa. O HTML é novamente analisado com BeautifulSoup.

- São percorridos blocos que agrupam benefícios por tema
- O título de cada tema é extraído de um cabeçalho <h3>.
- Dentro de cada bloco são recolhidos os textos dos benefícios individuais.
- Estes dados são guardados num dicionário onde a chave é o tema e o valor é a lista de benefícios. No fim, este dicionário é convertido numa string única, juntando cada tema e os respetivos benefícios, e essa string é guardada em teamlyzer_benefits.

```
benefits_url= f"https://pt.teamlyzer.com/companies/{nome_empresa}/benefits-and-values"
try:
    response_benefits= requests.get(benefits_url, headers=TEAMLYZER_HEADER)
    response_benefits.raise_for_status()

    soup_benefits = BeautifulSoup(response_benefits.text, "html.parser")

    teamlyzer_benefits = {}
    area=None

    benefits_blocks = soup_benefits.find_all("div", class_="col-lg-12 voffset3 divider_benefits")
    if not benefits_blocks:
        typer.echo("Empresa sem página de benefícios no Teamlyzer.")
        teamlyzer_benefits = None
    else:
        for block in benefits_blocks:
            block_area=block.find("h3")
            if block_area and block_area.find("b"):
                area = block_area.find("b").get_text(strip=True)
                teamlyzer_benefits[area] = []

            benefits = block.find_all("div", class_="flex_details")
            for benefit in benefits:
                benefits_text=benefit.get_text(strip=True)
                if benefits_text and area:
                    teamlyzer_benefits[area].append(benefits_text)
```

```
    benefits_lines = []
    for tema, beneficios in teamlyzer_benefits.items():
        if beneficios:
            beneficios_str = ", ".join(beneficios)
            benefits_lines.append(f"{tema}:{beneficios_str}")
    teamlyzer_benefits= " | ".join(benefits_lines)

except requests.RequestException:
    typer.echo(f"Erro ao aceder à página de benefícios do Teamlyzer.", err=True)
    teamlyzer_benefits = None
```

Por fim, a função get constrói um dicionário resultado com: o job_id, o título do anúncio, o nome da empresa e os quatro campos extraídos do Teamlyzer (teamlyzer_rating, teamlyzer_description, teamlyzer_salary, teamlyzer_benefits). Este dicionário é impresso no terminal em formato JSON. Depois o utilizador é questionado se pretende exportar os dados para CSV; em caso afirmativo é criado um ficheiro job_teamlyzer_<jobid>.csv com uma linha de cabeçalho e uma linha com todos estes valores.

```python
resultado = {
    "job_id": jobid,
    "job_title": job.get("title", ""),
    "company_name": company,
    "teamlyzer_rating": teamlyzer_rating,
    "teamlyzer_description": teamlyzer_description,
    "teamlyzer_benefits": teamlyzer_benefits,
    "teamlyzer_salary": teamlyzer_salary
}

print(json.dumps(resultado, indent=2, ensure_ascii=False))

exportar_csv = typer.confirm("Deseja exportar o resultado para CSV?")

if exportar_csv:
    filename = f"job_teamlyzer_{jobid}.csv"
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["job_id", "job_title", "company_name", "teamlyzer_rating",
                         "teamlyzer_description", "teamlyzer_salary", "teamlyzer_benefits"])
        writer.writerow([
            jobid, job["title"], company,
            teamlyzer_rating or "", teamlyzer_description or "",
            teamlyzer_salary or "", teamlyzer_benefits or ""
        ])
    typer.echo(f"CSV exportado com sucesso: {filename}")
else:
    typer.echo("Exportação não foi concluída")
```

Estatística de vagas por zona e tipo de trabalho

O comando statistics tem como objetivo gerar estatísticas conjuntas sobre as vagas disponíveis do site itjobs.pt, organizando a informação por zona geográfica e tipo da posição. Começamos por efetuar um pedido à API, onde são definidos parâmetros como o número máximo de resultados e a chave de autenticação da API. Caso a resposta ao servidor não seja bem sucedida, o programa apresenta uma mensagem de erro e termina a execução. Após a obtenção dos dados, os resultados são percorridos, e por cada oferta, são extraídos o título da posição e as respectivas localizações. Uma vez que uma oferta pode estar associada a múltiplas zonas, cada combinação de zona, título é considerada individualmente. Estas combinações são utilizadas como chaves de um dicionário, onde o valor corresponde ao número de vagas para esse trabalho nessa zona específica.

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

    parametros={
        "limit":200,
        "api_key":API_KEY
    }

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

    if response.status_code!=200:
        typer.echo("Erro ao obter os dados da API.")
        raise typer.Exit(1)

    jobs=response.json().get("results",[])
    estatisticas={}

    for job in jobs:
        titulo=job.get("title","")
        zonas=[loc.get("name","") for loc in job.get("locations",[])]

        for zona in zonas:
            chave=(zona,titulo)
            estatisticas[chave]=estatisticas.get(chave,0)+1

    typer.echo("Estatisticas de empregos por zona e tipo:")
    for (zona,titulo),contagem in estatisticas.items():
        typer.echo(f"{zona:<20} - {titulo:<40}: {contagem} empregos")
```

No final do processamento, as estatísticas são apresentadas no terminal, indicando o número de vagas para cada tipo de emprego e cada zona. Também é disponibilizada ao utilizador a opção de exportar os resultados obtidos para um csv. caso o utilizador escolha a exportação, é criado o ficheiro estatisticas_empregos.csv. Se a exportação não for selecionada, o programa informa que nenhum ficheiro foi gerado.

```python
csv_export=typer.confirm("Deseja exportar as estatísticas para CSV?")

if csv_export:
    filename="estatisticas_empregos.csv"
    with open(filename,mode='w',newline='',encoding='utf-8') as csvfile:
        writer=csv.writer(csvfile)
        writer.writerow(["zona","tipo de emprego","contagem"])
        for (zona,titulo),contagem in estatisticas.items():
            writer.writerow([zona,titulo,contagem])

    typer.echo(f"Estatísticas exportadas para {filename}")

else:
    typer.echo("Nao foi exportado nenhum csv.")
```

Listar as principais skills de um Website

O script implementa um comando de linha de comandos (CLI) utilizando a biblioteca Typer, que recolhe e analisa dados do website Teamlyzer para identificar as top 10 skills mais pedidas para um determinado trabalho. Começamos por definir o endereço base do site onde estão os empregos e criamos um comando CLI (list_skills) que recebe como

argumento o nome do trabalho. Definimos a url do site de forma a realizarmos o pedido HTTP GET à página. Convertemos para HTML a página através do BeautifulSoup de forma a analisar e a extrair a informação útil. A partir daí, fazemos a extração das skills, selecionando todos os os links e filtrando apenas os links que apontam para tags de skills. Se nenhuma skill for encontrada, o programa termina. Por fim, caso sejam encontradas skills, conta quantas vezes cada skills aparece e seleciona as 10 principais.

```python
BASE_URL_TEAMLYZER = "https://pt.teamlyzer.com/companies/jobs"

@app.command()
def list_skills(job: str):

    """Lista as top 10 skills pedidas no website Teamlyzer para um determinado trabalho."""

    url = f"https://pt.teamlyzer.com/companies/jobs?tags=python&order=most_relevant"

    try:
        response = requests.get(url, headers=header)
        response.raise_for_status()
    except requests.RequestException as e:
        typer.echo(f"Erro ao obter dados do Teamlyzer: {e}", err=True)
        raise typer.Exit()

    # Parse HTML
    soup = BeautifulSoup(response.text, "html.parser")

    all_skills = []

    tag_links = soup.select("div.tags a[href*='/companies/?tags=']")
    for link in tag_links:
        skill = link.get_text(strip=True).lower()
        if skill:
            all_skills.append(skill)

    if not all_skills:
        typer.echo("Nenhuma skill encontrada para este trabalho.")
        raise typer.Exit()

    # Contar ocorrências
    counter = Counter(all_skills)
    top10 = counter.most_common(10)

    resultado = [{"skill": skill, "count": count} for skill, count in top10]

    print(json.dumps(resultado, indent=2, ensure_ascii=False))
```

No final do processamento, é apresentado no terminal uma lista de dicionários que apresenta o nome da skill e o número de ocorrências, apresentando este resultado em formato json. Também é disponibilizada a opção de exportar os resultados obtidos para um ficheiro csv. Caso o utilizador não queira exportar os resultados para csv aparece uma mensagem a dizer que a exportação foi cancelada.

```python
    exportar_csv = typer.confirm("Deseja exportar o resultado para CSV?")

    if exportar_csv:
        filename = f"top10_skills_{job}.csv"
        with open(filename, mode="w", newline="", encoding="utf-8") as f:
            writer = csv.DictWriter(f, fieldnames=["skill", "count"])
            writer.writeheader()
            writer.writerows(resultado)

        typer.echo(f"CSV exportado com sucesso: {filename}")
    else:
        typer.echo("Exportação cancelada.")

if __name__ == "__main__":
    app()
```

Funcionalidade de exportar para csv

Esta funcionalidade encontra-se já implementada e descrita na secção do TP1, onde explicamos o processo da geração do ficheiro csv, bem como os campos exportados.

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.