# Aula Teórica 3 (guião)

## Semana de 17 a 21 de Fevereiro 2025


## José Carlos Ramalho


## Sinopsis:

Expressões regulares: o módulo re em Python

# Aquecimento

Quase todas as linguagens de programação utilizam parentesis na sua sintaxe. Especifica uma expressão regular que aceita frases compostas apenas por parentesis em que estes estão aninhados/agrupados corretamente.

Exemplos de frases corretas:

```
()
(())()
((()())())(())
```

Exemplos de frases incorretas:

```
)
((()))
())()
(())()(()
```

# Expressões Regulares em Python

## Strings e Raw Strings

```
s1 = "\\"
s2 = r"\\"
print(s1, s2)
```

```
Running cells with 'Python 3.12.6' requires the ipykernel package.

Run the following command to install 'ipykernel' into the Python environment.

Command: '/usr/local/bin/python3 -m pip install ipykernel -U --user --force-reinstall'
```

## Greedy and non greedy quantifiers

```
import re
texto = "<a href='http://exemple.com'>This is a link!</a>"

er1 = re.compile(r'<.*>')
print(er1.match(texto))
er2 = re.compile(r'<.*?>')
print(er2.match(texto))

print(re.search(r"'.*'", texto))
```

## Grupos de captura

```
import re
# Reconhecer um inteiro
er3 = re.compile(r'(\+|-)?(\d+)')
print(er3.match("-123"))
print(er3.match("-123").groups())
print(er3.match("-123").group(0))
print(er3.match("-123").group(1))
print(er3.match("-123").group(2))

import re
texto = "<a href='http://exemple.com'>This is a link!</a>"
texto2 = "<address>Isto é um exemplo</address>"

er4 = re.compile(r'<(.*?)>')
print(er4.match(texto))
print(er4.match(texto).groups())

er9 = re.compile(r'<\s*(\w+)\b>')
print(er9.match(texto2))
print(er9.match(texto2).groups())
print(er9.match(texto2).group(1))
```

Exercício: apanhar o texto do link

```
import re
texto = "<a href='http://exemple.com'>This is a link!</a>"
er10 = re.compile(r'(<[^>]*>)([^<]*)(<[^>]*>)')
print(er10.match(texto))
_, tlink, _ = er10.match(texto).groups()
print(tlink)
```

## Grupos de captura com reutilização

```
# Apanhar o URL
import re
texto = "<a href='http://exemple.com\'>This is a link!</a>"
er5 = re.compile(r'<a\s*href=([\'"])(.*)(\1)')
```

```
print(er5.match(texto))
print(er5.match(texto).groups())
print(er5.match(texto).group(2))
```

Exercício: Todas as strings binárias que começam e terminam pelo mesmo dígito binário

```
import re
binarias = ["0101010101", "111", "10001", "110", "101010101", "00110",
"1"]
er6 = re.compile(r'^([01])[01]*(\1)$')

for b in binarias:
    if er6.match(b):
       print(er6.match(b).group(0), er6.match(b).group(1),
er6.match(b).group(2))
```

Desafio:

1. Tenta reescrever a express\ao regular sem recorrer a '\1';
2. Desenha o AFD.

## Named Groups: `(?P<name>...)`

```
import re
texto = "<p>Como dizia <nome>Afonso Henriques</nome>...</p>"
er7 = re.compile(r'<nome>(?P<nome>.*)<\/nome>')
print(er7.search(texto))
print(er7.search(texto).group('nome'))

# Com reutilização
import re
texto = "<p>Como dizia <nome>Afonso Henriques</nome>...</p>"
er7 = re.compile(r'<(?P<tag>\w+)>(?P<conteudo>.*)<\/(?P=tag)>')
print(er7.search(texto))
print(er7.search(texto).group('tag'))
print(er7.search(texto).group('conteudo'))
print(er7.search(texto).groups())
print(er7.search(texto).groupdict())

# Tentando processar as tags interiores ('nested tags')
import re
texto = "<p>Como dizia <nome>Afonso Henriques</nome>...</p>"
er7 = re.compile(r'<(?P<tag>\w+)>(?P<conteudo>.*)<\/(?P=tag)>')
print(er7.search(texto))
print(er7.search(texto).group('tag'))
print(er7.search(texto).group('conteudo'))
print(er7.search(texto).groups())
print(er7.search(texto).groupdict())
```

```
dictP = er7.search(texto).groupdict()
print(er7.search(dictP['conteudo']).groupdict())

# 5 - 15 + 23 ... - 7
import re
texto = "5 - 15 + 23 - 43 - 7"
er8 = re.compile(r'\s*(?P<val>\d+)\s*((?P<op>[+\-])\s*\d+)*')
print(er8.search(texto))
print(er8.search(texto).groups())
print(er8.search(texto).groupdict())
```

Exercício: Processar um ficheiro CSV

```
import re
# Processar uma linha de CSV
linha = "A22345;Ana Maria Domingues;LEI;12;16;14;13;12"
aluno = re.compile(r'(?P<id>A\d+);(?P<nome>(\w+|\s+)+);(?P<curso>\w+);
(?P<nota1>\d+);(?P<nota2>\d+);(?P<nota3>\d+);(?P<nota4>\d+);(?
P<nota5>\d+)')
print(aluno.search(linha))
print(aluno.search(linha).groups())

adic = aluno.search(linha).groupdict()
print(adic['id'], adic['nome'])
```

Exercício: Generalizar o exercício anterior e aplicar a um dataset

```
# alunos.csv
alunos = [
    "a1,Aysha Melanie Gilberto,LEI,12,8,19,8",
    "a2,Igor André Cantanhede,ENGFIS,12,16,18,20",
    "a3,Laurénio Narciso,ENGFIS,8,14,15,14"
]

aluno = re.compile(r'(?P<id>a\d+),(?P<nome>(\w+|\s+)+),(?P<curso>\w+),
(?P<nota1>\d+),(?P<nota2>\d+),(?P<nota3>\d+),(?P<nota4>\d+)')

turma = []
for a in alunos:
    turma.append(aluno.match(a).groupdict())

print(turma)
```

## Groups: start, end, span

```
import re

p = re.compile(r'\d+')
print(p.match("12"))
digitos = p.match("12").group()
```

```
print(int(digitos)*2)
print(p.match("12").group())
print(p.match("12").start())
print(p.match("12").end())
print(p.match("12").span())
```

## Funções do módulo re

- `re.search(pattern, string, flags=0)` - Percorre a string até à primeira localização onde ocorra um match com a expressão regular; Retorna `None` se nenhuma parte da string corresponder ao padrão;
- `re.match(pattern, string, flags=0)` - Se 0 ou mais carateres no início da string corresponderem ao padrão, retorna um objeto Match, caso contrario retorna `None`;
- `re.fullmatch(pattern, string, flags=0)` - Se o padrão corresponder à totalidade da string retorna um objeto Match, caso contrario retorna `None`;
- `re.split(pattern, string, maxsplit=0, flags=0)` - Parte a string de acordo com o padrão devolvendo uma lista das partes. O parâmetro `maxsplit` pode ser usado para definir um limite de divisões (por omissão é 0, o que corresponde a divisões infinitas).;
- `re.findall(pattern, string, flags=0)` - Retorna todas as partes que não se sobrepoêm e que fazem match com o padrão;
- `re.finditer(pattern, string, flags=0)` - Retorna um iterador sobre as partes que não se sobrepoêm e que fazem match com o padrão;
- `re.sub(pattern, repl, string, count=0, flags=0)` - Substitui todas as correspondências da expressão regular `pattern` na `string` por `repl`. `repl` pode ser uma string, uma expressão regular ou uma função que recebe um objeto Match e devolve uma string. O parâmetro `count` determina o limite de substituições (por omissão é 0, ou seja, não há limite);
- `re.subn(pattern, repl, string, count=0, flags=0)` - Realiza a mesma operação que a anterior, `sub()`, mas retorna um tuplo `(nova_string, num_subs)`.

`fullmatch()`

```
import re

credit_card_pattern = r'\d{4}-\d{4}-\d{4}-\d{4}'
credit_card_number = '1234-5678-9012-3456'

if re.fullmatch(credit_card_pattern, credit_card_number):
    print('Cartão válido!')
```

```
else:
    print('Cartão inválido!')
```

`findall()`

```
import re

text = "Contact us at support@domain.com or sales@domain.com"
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

email_list = re.findall(email_pattern, text)
print(email_list)
```

`finditer()`

```
import re

text = "Contact us at support@domain.com or sales@domain.com"
email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

email_list = []

for match in re.finditer(email_pattern, text):
    email_list.append(match)
    start_pos = match.start()
    end_pos = match.end()
    email = match.group()
    print(f"Found email '{email}' at positions {start_pos}-{end_pos-
1}.")

print(email_list)
```

```
import re
# Apanhar as palavras terminadas em mente: Qual é o problema?
p = re.compile(r'\w*mente\b')
texto = """
Fernando Pessoa
A população de Portugal encontra-se dividida, psicologicamente,
Interregno

A população de Portugal encontra-se hoje dividida, psicologicamente,
em cinco grupos distintos.
O primeiro grupo, e o maior, que consiste na quase totalidade da nossa
larga percentagem de analfabetos e em uma razoável parte do povo, e
baixa burguesia,
que não o é, apresenta os característicos distintivos do indivíduo
português — a sua ausência de personalidade, a sua afectividade
impulsiva e incoerente,
a sua descontinuidade de vontade e de pensamento, e o seu amor pátrio
animal e firme, pelo qual se congrega com facilidade em tomo do que
constitua, ou julgue
```

```
que constitui, a Nação.

O segundo grupo, mais restrito, mas suficientemente largo para ser
importante na vida nacional, é o que forma a massa dos partidos
políticos; formam-no grande parte
da baixa burguesia, grande parte da média burguesia, e uma parte
incerta da alta burguesia.
Este português, tendo a mesma descontinuidade que o do primeiro grupo,
já não tem as boas qualidades fundamentais, que aquele distinguem.
O seu patriotismo, às vezes real, é todavia desfigurado por
partidarismos vários, que por vezes se sobrepõem a ele.
Ignorante, e, por isso, admirador de um estrangeiro que desconhece,
esta gente é a que crê nos sagrados princípios da revolução,
ou nos princípios igualmente sagrados da Monarquia Integral.
Indolente, vivendo de empenhos e de cargos públicos que não exerce,
é esta camada o principal obstáculo à reformação da Nação Portuguesa.
O português simples é um simples animal afectivo e perturbado:
uma forte superior direcção orienta-o e leva-o para onde quer. Estes
outros, supondo que têm opiniões, têm-nas todavia o bastante para
constituir um estorvo.
Duplo estorvo — porque contagiam os estúpidos ingénuos que lhes estão
abaixo e porque resistem aos cultos ou mais inteligentes, que lhes
estão acima.

O terceiro grupo, psicologicamente parecido com o segundo, pois o
caracterizam a mesma incapacidade de acção útil, é formado de grande
parte da alta burguesia e de grande parte da burguesia média. São
inertes, conservadores e desnacionalizados. São os do "lá fora é outra
coisa", "isto é um país único", "isto é pior que Marrocos", frases
que, em justiça se diga, ou não aparecem na boca dos outros grupos, ou
só episodicamente e por imitação aparecem.

Um quarto grupo, formado de elementos causais de todas as classes,
grupo restrito e (...).

"""

lista = p.findall(texto)
print(lista)

# Encontrar números no texto
import re

def recon(texto):
    m = re.findall(r'\d+', texto)
    if m:
        print("Encontrado: ", m)
    else:
        print("Não encontrado...")
```

```
recon("123 + 456 = 999")
```

```
import re

def recon(texto):
    m = re.finditer(r'\d+', texto)
    if m:
        for obj in m:
            print("Encontrado: ", obj.group())
    else:
        print("Não encontrado...")

recon("123 + 456 = 999")
```

# TPC3: Conversor de MarkDown para HTML

Criar em Python um pequeno conversor de MarkDown para HTML para os elementos descritos na "Basic Syntax" da Cheat Sheet:

## Cabeçalhos: linhas iniciadas por "# texto", ou "## texto" ou "### texto"

In: `# Exemplo`

Out: `<h1>Exemplo</h1>`

## Bold: pedaços de texto entre "**":

In: `Este é um **exemplo** ...`

Out: `Este é um <b>exemplo</b> ...`

## Itálico: pedaços de texto entre "*":

In: `Este é um *exemplo* ...`

Out: `Este é um <i>exemplo</i> ...`

## Lista numerada:

In:

```
1. Primeiro item
2. Segundo item
3. Terceiro item
```

Out:

```
<ol>
<li>Primeiro item</li>
<li>Segundo item</li>
<li>Terceiro item</li>
</ol>
```

## Link: texto

In: `Como pode ser consultado em [página da UC](http://www.uc.pt)`

Out: `Como pode ser consultado em <a href="http://www.uc.pt">página da UC</a>`

## Imagem: texto alternativo

In: Como se vê na imagem seguinte: `![imagem dum coelho](http://www.coellho.com) ...`

Out: `Como se vê na imagem seguinte: <img src="http://www.coellho.com" alt="imagem dum coelho"/> ...`