Strings

Strings são **sequências de caracteres** usadas para **representar texto**, delimitadas por aspas simples, duplas ou triplas.

```
mensagem = 'Olá, mundo!'
mensagem = "Python é incrível!"


texto = """Essa é uma string
que pode ter múltiplas
linhas."""
```

Aspas simples ou duplas podem ser usadas, conforme a preferência ou contexto.

Aspas triplas permitem criar strings de múltiplas linhas com quebras de linha.

Métodos de manipulação

Os métodos de manipulação de strings são usados para transformar, modificar ou analisar textos.

Método	Conceito	Exemplo	Saída
strip()	Remove espaços em branco (ou caracteres especificados) do início e fim da string.	" exemplo ".strip()	"exemplo"
lower()	Converte todos os caracteres da string para letras minúsculas.	"EXEMPLO".lower()	"exemplo"
upper()	Converte todos os caracteres da string para letras maiúsculas.	"exemplo".upper()	"EXEMPLO"
replace()	Substitui uma substring específica por outra na string.	"Olá Mundo".replace("Mundo", "Python")	"Olá Python"

f-strings

As f-strings permitem a formatação de strings de forma simples e legível, incorporando expressões e variáveis diretamente no texto.

Elas usam a sintaxe: f"{variável}"

```
estudante = "Pedro"
nota = 10
mensagem = f"{estudante} tirou a nota {nota}!"

print(mensagem)
```

A mensagem impressa será a string “Pedro tirou a nota 10!”.

Indexação de strings

A indexação permite acessar caracteres individuais de uma string através de seu índice, começando de 0 para o primeiro caractere.

Para acessar caracteres a partir do final, usa-se índices negativos, onde -1 é o último caractere.

```
texto = "Python"

print(texto[5])
print(texto[-1])
```

Imprime o último caractere da string que é “n”.

Slicing

O slicing (em português fatiamento) permite extrair uma parte da string.

A sintaxe é `string[início:fim:passo]`, onde:

- ✔ **início:** índice inicial. O caractere dessa posição é incluído na string final.
- ✔ **fim:** índice final. O caractere dessa posição é excluído na string final.
- ✔ **passo:** intervalo entre os índices (opcional)

Exemplo

```
texto = "Python"

print(texto[1:4])
print(texto[:3])
print(texto[::2])
```

Extrai os caracteres da posição 1 até a posição 3 (não incluindo o índice 4), resultando em 'yth'.

Extrai os primeiros 3 caracteres, resultando em 'Pyt'.

Extrai os caracteres de forma alternada, pegando um a cada dois, resultando em 'Pto'.

Operador in

O operador in verifica se uma substring está presente em uma string.

Ele retorna True se a substring estiver presente na string e False caso contrário.

```
texto = "Python"

print("Py" in texto)

print("Java" in texto)
```

Verifica se a substring "Py" está presente na string "Python", retornando True porque ela é encontrada no início da string.

Verifica se a substring "Java" está em "Python", retornando False porque não está presente.

Método startswith()

O método startswith() verifica se a string começa com uma substring específica.

Ele retorna True se a string iniciar com a substring especificada e False caso contrário.

```
texto = "Python"

print(texto.startswith("Py"))

print(texto.startswith("py"))
```

Verifica se a string "Python" começa com a substring "Py", retornando True no primeiro print. No segundo print verifica se começa com "py", retornando False devido à diferença de maiúsculas e minúsculas.

Método endswith()

O método endswith() verifica se a string termina com uma substring específica.

Ele retorna True se a string finalizar com a substring especificada e False caso contrário.

```
texto = "Python"

print(texto.endswith("on"))

print(texto.endswith("ton"))
```

Verifica se a string "Python" termina com a substring "on", retornando True no primeiro print. No segundo print verifica se termina com "ton", retornando False.

Regex

Regex (expressões regulares) é uma ferramenta poderosa para buscar, manipular e validar padrões em strings usando uma linguagem de padrões.

É essencial para lidar com padrões de texto mais complexos, como e-mails, números de telefone, validação de entradas, entre outros.

Com Regex é possível criar padrões dinâmicos para validar, buscar ou substituir dados em textos, de forma compacta e poderosa para lidar com manipulação de strings.

Caracteres literais

Caracteres literais em regex correspondem exatamente aos caracteres na string de entrada, como letras, números e símbolos, sem nenhum comportamento especial, a menos que sejam caracteres especiais.

Por exemplo, se você usar o caractere a em uma regex, ele corresponderá ao caractere "a" em uma string, sem nenhuma modificação ou interpretação especial.

Exemplo: Se você quiser encontrar a palavra "Python" em um texto, pode usar a regex Python.

Caracteres especiais têm um significado específico dentro das expressões regulares. Para serem usados como caracteres literais devem ser "escapados" com uma barra invertida (\).

Caracteres especiais

Caracteres especiais em expressões regulares são símbolos com significados específicos que permitem definir padrões complexos.

Símbolo	Descrição
.	Corresponde a qualquer caractere, exceto nova linha
\d	Corresponde a qualquer dígito (0-9)
\D	Corresponde a qualquer caractere que não seja um dígito
\w	Corresponde a qualquer caractere alfanumérico (letras, números e underline)
\W	Corresponde a qualquer caractere que não seja alfanumérico
\s	Corresponde a qualquer espaço em branco (espaço, tabulação, etc.)
\S	Corresponde a qualquer caractere que não seja espaço em branco

Classe de caracteres

Classes de caracteres em expressões regulares são grupos de caracteres definidos entre colchetes, que pode corresponder a diferentes padrões.

Símbolo	Descrição
[abc]	Corresponde a qualquer caractere 'a', 'b' ou 'c'
[^abc]	Corresponde a qualquer caractere que não seja 'a', 'b' ou 'c'
[a-z]	Corresponde a qualquer caractere de 'a' a 'z' (minúsculas)
[A-Z]	Corresponde a qualquer caractere de 'A' a 'Z' (maiúsculas)
[0-9]	Corresponde a qualquer dígito (0-9)
[a-zA-Z]	Corresponde a qualquer letra, maiúscula ou minúscula

Quantificadores

Quantificadores em expressões regulares são usados para especificar o número de ocorrências de um padrão.

Símbolo	Descrição
*	Corresponde a 0 ou mais ocorrências do padrão anterior
+	Corresponde a 1 ou mais ocorrências do padrão anterior
?	Corresponde a 0 ou 1 ocorrência do padrão anterior
{n}	Corresponde exatamente a n ocorrências do padrão anterior
{n,}	Corresponde a n ou mais ocorrências do padrão anterior
{n,m}	Corresponde entre n e m ocorrências do padrão anterior

Exemplo

Regex que corresponde aos números de telefone no formato (XX) XXXX-XXXX ou (XX) XXXXX-XXXX, como os telefones "(12) 3456-7890" e "(21) 98765-4321".

Regex: `\(\d{2}\)\s\d{4,5}-\d{4}`

Explicação:

- ✔ **\(\d{2}\):** Dois dígitos dentro de parênteses, como o código de área.
- ✔ **\s:** Um espaço.
- ✔ **\d{4,5}:** Quatro ou cinco dígitos para o número do telefone.
- ✔ **-:** O hífen literal.
- ✔ **\d{4}:** Quatro dígitos no final.

Exemplo

Regex que corresponde a datas no formato "DD/MM/AAAA", como as datas "12/05/2023" e "01/01/2021"

Regex: `\b\d{2}/\d{2}/\d{4}\b`

Explicação:

- ✔ **\b:** Limite de palavra para garantir que a correspondência seja uma data completa.
- ✔ **\d{2}:** Dois dígitos para o dia e o mês.
- ✔ **/:** O caractere de barra literal.
- ✔ **\d{4}:** Quatro dígitos para o ano.
- ✔ **\b:** Limite de palavra no final para garantir que não haja caracteres extras após a data.

Módulo re

O módulo re oferece suporte para trabalhar com expressões regulares em Python.

Para utilizá-lo, basta importar em seu código com:

```
import re
```

Métodos do re

O módulo re oferece métodos para facilitar a interação com textos de maneira flexível e eficiente.

Método	Descrição	Exemplo de uso	Saída
search	Procura por um padrão em qualquer parte da string. Retorna o primeiro resultado encontrado.	re.search(r"\d+", "Há 1234 alunos")	Retorna '1234'
match	Verifica se o padrão corresponde ao início da string.	re.match(r"abc", "abcdef")	Retorna um match com 'abc'
findall	Retorna todas as ocorrências do padrão em uma lista.	re.findall(r"\d+", "Eu tenho 3 gatos e 2 cachorros")	Retorna a lista ['3', '2']
sub	Substitui ocorrências do padrão por uma string.	re.sub(r"\d", "#", "Meu número é 1234")	Retorna 'Meu número é ####'

Método group

O método group() é utilizado para acessar a correspondência encontrada por uma expressão regular, retornando o texto correspondente ao padrão.

Ele é comum quando se usa métodos como re.search() ou re.match() que retornam um objeto de correspondência.

Se uma correspondência for encontrada, group() retorna o valor da string que corresponde ao padrão.

Exemplo

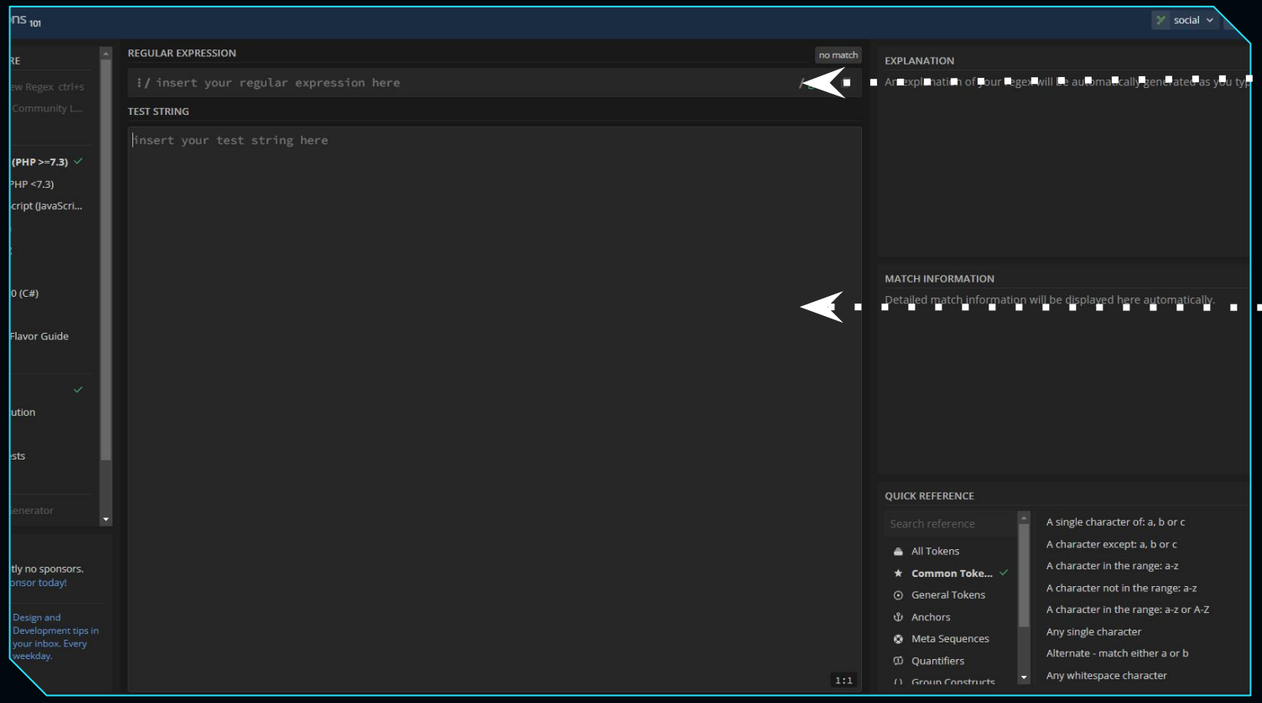
O código a seguir utiliza o módulo re para extrair um endereço de email de uma string:

```python
import re

texto = "Entre em contato pelo email support@example.com"

padrao_email = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

resultado = re.search(padrao_email, texto)

if resultado:
    print("Email encontrado:", resultado.group())
else:
    print("Nenhum email encontrado.")
```

Quando re.search() encontra uma correspondência, resultado.group() retorna o texto correspondente ao padrão. Se não houver correspondência, resultado será None, e o código deve evitar chamar group() para evitar erros, por isso usamos um else.

Ferramentas

Testar expressões regulares em ferramentas como o regex101 é importante para validar e depurar padrões antes de usá-los no código.



Campo para inserir o Regex

Campo para testar com strings

Para se aprofundar em regex

Programação > Cursos de Computação

Curso de

Expressões Regulares: faça buscas, validações e substituições de textos

Acessar!

alura

Compartilhe um resumo de seus novos conhecimentos em suas redes sociais.

#aprendizadoalura