🧩 Python Regular Expressions (Expresiones Regulares)

Las **expresiones regulares (RegEx)** son patrones que permiten buscar, validar, reemplazar o dividir texto de forma precisa y flexible. Python incluye el módulo `re` para trabajar con expresiones regulares.

🔷 Introducción a las Expresiones Regulares

- Una **expresión regular** es una **cadena de texto** que describe un patrón de búsqueda.
- Se usa comúnmente para validar datos (emails, contraseñas, fechas), extraer información o modificar texto.
- En Python, el módulo `re` debe importarse explícitamente:

```
import re
```

🔷 Conceptos Básicos

Función	Descripción	Ejemplo
`re.search(pattern, string)`	Busca la primera coincidencia en toda la cadena	`re.search("dog", "my dog")` ✅
`re.match(pattern, string)`	Solo busca al **inicio** de la cadena	`re.match("dog", "dog runs")` ✅
`re.findall(pattern, string)`	Devuelve una **lista** con todas las coincidencias	`re.findall("cat", "cat cat dog")` → `['cat', 'cat']`
`re.finditer(pattern, string)`	Devuelve un **iterador** con objetos `Match`	útil para obtener posiciones
`re.sub(pattern, repl, string)`	Reemplaza coincidencias por `repl`	`re.sub("cat", "dog", "cat run")` → `"dog run"`
`re.split(pattern, string)`	Divide la cadena por el patrón	`re.split(r"\s+", "a b c")` → `['a','b','c']`

🔷 Objetos `Match`

Cuando se usa `re.search()` o `re.match()`, se obtiene un objeto con información útil:

```
match = re.search(r"(\w+)@(\w+)", "email test@mail.com")
print(match.group())     # test@mail.com
print(match.group(1))    # test
```

```
print(match.group(2))    # mail
print(match.start(), match.end())  # posiciones del match
```

◈ Patrones Básicos de Expresiones Regulares

Patrón	Significado	Ejemplo
`.`	Cualquier carácter excepto salto de línea	`r"a.b"` → “acb”, “arb”
`\d`	Cualquier dígito `[0-9]`	`\d\d` → dos dígitos seguidos
`\w`	Cualquier carácter alfanumérico `[a-zA-Z0-9_]`	`\w+` → una o más letras
`\s`	Espacio en blanco	`\s+` → uno o más espacios
`\D`	No dígito	
`\W`	No alfanumérico	
`\S`	No espacio	
`^`	Inicio de línea	`^Hello` busca “Hello” al inicio
`$`	Fin de línea	`world$` busca “world” al final
`[]`	Conjunto de caracteres	`[aeiou]` busca vocales
`[^ ]`	Negación dentro del conjunto	`[^0-9]` → todo excepto números
`|`	Operador OR	`cat | dog` → “cat” o “dog”
`()`	Grupo de captura	`(abc)+` → uno o más “abc”

◈ Cuantificadores

Cuantificador	Significado	Ejemplo
`*`	0 o más repeticiones	`a*b` → “b”, “ab”, “aab”
`+`	1 o más repeticiones	`a+b` → “ab”, “aab”
`?`	0 o 1 repetición	`colou?r` → “color”, “colour”
`{n}`	Exactamente n repeticiones	`\d{4}` → “2025”
`{n,}`	n o más repeticiones	`\d{2,}`
`{n,m}`	Entre n y m repeticiones	`a{1,3}` → “a”, “aa”, “aaa”

◈ Flags (modificadores)

Los **flags** cambian el comportamiento de las búsquedas:

Flag	Descripción	Uso
`re.IGNORECASE` o `re.I`	Ignora mayúsculas/minúsculas	`re.search(r"python", text, re.I)`
`re.MULTILINE` o `re.M`	Hace que `^` y `$` coincidan en cada línea	
`re.DOTALL` o `re.S`	El punto `.` incluye saltos de línea	
`re.VERBOSE` o `re.X`	Permite escribir expresiones con comentarios y espacios	

◈ Escapes útiles

Algunos caracteres tienen significado especial y deben escaparse con `\`:

Carácter	Escapado	Ejemplo
`.`	`\.`	Coincide con un punto literal
`\`	`\\`	Coincide con una barra invertida
`[ ]`	`\[ \]`	Coincide con corchetes
`( )`	`\( \)`	Coincide con paréntesis

◈ Ejemplos Prácticos

📓 Validar una dirección de correo electrónico

```
import re
pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"

email = "user@mail.com"
if re.match(pattern, email):
    print("Email válido")
else:
    print("Email inválido")
```

📓 Buscar todos los números de un texto

```
text = "Hay 2 gatos, 5 perros y 12 peces"
numbers = re.findall(r"\d+", text)
print(numbers)  # ['2', '5', '12']
```

Reemplazar texto con `re.sub()`

```
text = "Python 2 is old, Python 3 is modern"
new_text = re.sub(r"Python 2", "Python 3", text)
print(new_text)  # Python 3 is old, Python 3 is modern
```

Dividir texto por separadores múltiples

```
line = "apple, banana; orange|grape"
fruits = re.split(r"[;,\|]", line)
print(fruits)  # ['apple', ' banana', ' orange', 'grape']
```

Buenas Prácticas

- Siempre **usa cadenas raw (`r"..."`)** para evitar conflictos con escapes.

  ```
  re.search(r"\d+", text)
  ```

- **Comienza simple**, luego añade grupos o cuantificadores.
- Usa `re.VERBOSE` para **comentar expresiones complejas**.
- Valida tus expresiones con herramientas online como: regex101.com

---

Referencias oficiales

- Python `re` module documentation
- W3Schools – Python RegEx
- Regex101 – Live tester & debugger

¿Querés que sigamos con el siguiente módulo del curso (el de **Managing Data and Processes with Python**, donde se ve subprocess, os, sys, etc.)?