## Was sind reguläre Ausdrücke?

Ein regulärer Ausdruck ist eine Zeichenkette, die ein Suchmuster beschreibt. Dieses Muster kann verwendet werden, um Text zu suchen, zu ersetzen oder zu überprüfen, ob ein bestimmtes Muster in einem String vorkommt.

**Gründe für den Einsatz von RegEx:**

- Suchen und Ersetzen von Mustern
- Überprüfen der Formatierung (z.B. Email-Adressen, Telefonnummern)
- Extrahieren von Informationen aus unstrukturierten Daten

## Wichtige RegEx-Symbole

Hier sind einige grundlegende RegEx-Symbole, die häufig verwendet werden:

- `.` : Ein beliebiges Zeichen (außer Zeilenumbruch).
- `^` : Beginn eines Strings.
- `$` : Ende eines Strings.
- `[]` : Ein Bereich von Zeichen (z.B. `[a-z]` für Kleinbuchstaben).
- `|` : Logisches "oder" (alternativ).
- `\d` : Eine Ziffer (entspricht `[0-9]` ).
- `\w` : Ein Wortzeichen (entspricht `[a-zA-Z0-9_]` ).
- `\s` : Ein Leerzeichen (Whitespaces).
- `*` : Null oder mehr Vorkommen des vorherigen Zeichens.
- `+` : Ein oder mehr Vorkommen des vorherigen Zeichens.
- `?` : Null oder ein Vorkommen des vorherigen Zeichens.
- `{n,m}` : Zwischen n und m Vorkommen des vorherigen Zeichens.

## RegEx in Pandas verwenden

Pandas bietet eine eingebaute Unterstützung für reguläre Ausdrücke, die du mit der `str` -Methode auf Strings im DataFrame anwenden kannst.

**Beispiel:**

```
import pandas as pd

# Beispiel-DataFrame
```

```
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Email': ['alice@example.com', 'bob123@example.net', 'charlie@domai
df = pd.DataFrame(data)

# Überprüfen, ob die E-Mail-Adressen ein gültiges Muster haben
pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'  # RegEx für
df['Email Valid'] = df['Email'].str.match(pattern)

print(df)
```

**Erklärung des RegEx-Musters:**

- `^[a-zA-Z0-9._%+-]+` : Der Beginn der E-Mail (Ziffern, Buchstaben, Punkt, Unterstrich oder andere Symbole).
- `@` : Das `@` -Symbol.
- `[a-zA-Z0-9.-]+` : Der Domain-Name (Buchstaben, Zahlen, Punkt, Bindestrich).
- `\.[a-zA-Z]{2,}$` : Der TLD (z.B. `.com`, `.org` ), wobei es sich um mindestens 2 Buchstaben handelt.

## Weitere nützliche RegEx-Funktionen in Pandas

1. **`str.contains()`**
   Sucht nach einem Muster innerhalb eines Strings (gibt `True` oder `False` zurück).

   ```
   df['Contains com'] = df['Email'].str.contains('com')
   ```

2. **`str.replace()`**
   Ersetzt ein Muster durch einen neuen String.

   ```
   df['Replaced Email'] = df['Email'].str.replace('@', ' at ')
   ```

3. **`str.extract()`**
   Extrahiert Teile eines Strings, die einem Muster entsprechen.

   ```
   df['Domain'] = df['Email'].str.extract(r'@([a-zA-Z0-9.-]+)')
   ```

4. **`str.findall()`**
   Findet alle Vorkommen eines Musters in einem String.

```
df['Digits'] = df['Email'].str.findall(r'\d+')  # Extrahiert alle Zahl
```

## Beispiel: Erkennen von Telefonnummern

Angenommen, wir haben einen DataFrame mit Telefonnummern und möchten überprüfen, ob die Telefonnummern im Format `XXX-XXX-XXXX` vorliegen:

```
# Beispiel-DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Phone': ['123-456-7890', '987-654-3210', '123-4567', '555-123-4567
df = pd.DataFrame(data)

# RegEx für die Telefonnummer
phone_pattern = r'^\d{3}-\d{3}-\d{4}$'  # Format: XXX-XXX-XXXX

df['Phone Valid'] = df['Phone'].str.match(phone_pattern)

print(df)
```

### Erklärung des RegEx-Musters:

- `^\d{3}` : Drei Ziffern am Anfang.
- `-` : Bindestrich.
- `\d{3}` : Drei Ziffern.
- `-` : Bindestrich.
- `\d{4}$` : Vier Ziffern am Ende.

## RegEx mit verschiedenen Optionen

1. **`flags` -Parameter**
   Du kannst den `flags` -Parameter verwenden, um die Suche zu modifizieren (z.B. Groß-/Kleinschreibung ignorieren).

```
df['Email Valid'] = df['Email'].str.match(pattern, flags=re.IGNORECASE
```

2. **`na` -Parameter**

   Der `na` -Parameter erlaubt es, einen benutzerdefinierten Wert für `NaN` zu setzen, wenn der Ausdruck keine Übereinstimmung findet.

```
df['Phone Valid'] = df['Phone'].str.match(phone_pattern, na=False)
```

## Ausblick:

Mit Funktionen wie `str.contains()`, `str.replace()`, `str.extract()` und `str.findall()` kannst du Muster suchen, ersetzen und extrahieren.