Les expressions régulières

Définition d’un dictionnaire:

•Les **expressions régulières** sont des motifs utilisés pour rechercher ou manipuler des chaînes de caractères.

•Elles permettent de valider des formats, extraire des informations, ou même remplacer du texte. En Python, elles sont gérées par le module re.

```
import re
```

Création et Utilisation de Regex

Pour créer une expression régulière, nous utilisons la fonction **re.compile()** pour définir un motif.

```
import re

pattern = re.compile(r'motif')
```

Le préfixe r indique une **chaîne brute** où les caractères d'échappement (comme \n pour une nouvelle ligne) ne sont pas interprétés par Python mais sont traités directement dans le motif.

La fonction search()

- Recherche la première occurrence d'un motif dans une chaîne

```python
import re

result = re.search(r'\d+', 'Il y a 42 capteurs actifs parmi les 100 capteurs.')
if result:
    print('Motif trouvé :', result.group())  # Affiche : Motif trouvé : 42
```

- ou

```python
import re

pattern = re.compile(r'\d+')
result = pattern.search('Il y a 42 capteurs en défaut sur les 100 capteurs.')
print(result.group())  # Affiche : 42
```

La fonction match()

- Vérifie si la chaîne commence par le motif.

```python
import re

pattern = re.compile(r'\d+')

result = pattern.match('42 capteurs en défaut sur les 100 capteurs.')

if result:
    print('Motif au début :', result.group())  # Affiche : Motif au début : 42
```

- Retourne **None** si aucun motif trouvé

La fonction findall()

Retourne toutes les occurrences du motif dans une chaîne

```python
import re

pattern = re.compile(r'\d+')

matches = pattern.findall('42 capteurs, 12 caméras.')
print('Occurrences trouvées :', matches)  # Affiche : ['42', '12']
```

Retourne une **liste vide** si aucune occurrence n’est trouvé

La fonction finditer()

finditer() permet de trouver toutes les occurrences d'un motif regex dans une chaîne, mais contrairement à **findall(),** il renvoie un itérateur d'objets avec informations détaillées.

```python
import re
chaine = "20% de réduction sur €100"

motif = re.compile(r'\d+')
resulat = motif.finditer(chaine)

for element in resulat:
    print(element.group())
```

La fonction sub()

Remplace toutes les occurrences d'un motif par une autre chaîne

```python
import re

pattern = re.compile(r'\d+')

new_text = pattern.sub('XX', '42 capteurs, 12 caméras.')

print('Nouveau texte :', new_text)  # Affiche : XX capteurs, XX caméras.
```

Récapitulatif des principales méthodes

Méthode	Description	Exemple
re.search()	Recherche la première occurrence d'un motif	re.search(r'\d+', '42 capteurs')
re.match()	Vérifie si la chaîne commence par un motif	re.match(r'\d+', '123 capteurs’)
re.findall()	Retourne toutes les occurrences d'un motif	re.findall(r'\d+', '42 cap, 12 cam’)
re.sub()	Remplace toutes les occurrences d'un motif	re.sub(r'\d+', 'XX', '42 capteurs’)

Drapeaux (flags):

Recherche insensible à la casse avec le flag: **re.IGNORECASE ou (re.I)**

```python
import re

pattern = re.compile(r'capteurs', re.IGNORECASE)

result = pattern.search('42 CAPTEURS en défaut.')

if result:
    print(result.group())  # Affiche : CAPTEURS
else:
    print("Aucun motif trouvé")
```

Drapeaux (flags):

re.MULTILINE (re.M) : Permet à ^ et $ de correspondre au début et à la fin de chaque ligne.

```python
import re

pattern = re.compile(r'^capteurs', re.MULTILINE)

result = pattern.findall("capteurs\ncapteurs de réseau")

print(result)  # Affiche : ['capteurs', 'capteurs']
```

Classes et Intervalles de Classes

Classes de caractères :

\d : Chiffre.
\w : Caractère alphanumérique (lettres, chiffres, underscore).
\s : Espace blanc.

Intervalles :

[a-z] : Lettres minuscules.
[A-Z] : Lettres majuscules.
[0-9] : Chiffres.

Classes et Intervalles de Classes

Exemple: classes de caractères :

```python
import re

# Classe de caractères [a-z] pour les lettres minuscules
pattern = re.compile(r'[a-z]+')
result = pattern.findall('Python3 RegEx is awesome!')
print(result)  # Affiche : ['ython', 'eg', 'is', 'awesome']
```

Les Quantificateurs

Permettent de spécifier le nombre de répétitions d'un motif.

***** : 0 ou plus répétitions.

+ : 1 ou plus répétitions.

? : 0 ou une fois

{n,m} : de n à m répétitions.

Les Quantificateurs

Exemple de quantificateur

```python
import re

# Utilisation des quantificateurs pour un numéro de téléphone
pattern = re.compile(r'0\d{9}')

result = pattern.search('Numéro : 0623456789')
if result:
    print("Numéro valide :", result.group())
else:
    print("Numéro non valide")
```

Métacaractères Spéciaux

Caractères spéciaux dans les regex

. : N'importe quel caractère.

^ : Début de la chaîne.

$: Fin de la chaîne.

\b : Limite de mot.

Exemple d’utilisation

```python
import re

texte = """
Les dates importantes sont les suivantes : 25/12/2024, 01-01-2023, et 15/08/2025.
"""
pattern = r'\d{2}[-\/]\d{2}[-\/]\d{4}'

dates = re.findall(pattern, texte)

for date in dates:
    print(f"Date trouvée : {date}")
```

Lookahead

Vérifie que quelque chose suit sans le capturer

```python
import re

pattern = re.compile(r'\d+(?=%)')

result = pattern.search('20% de réduction')

print(result.group())  # Affiche : 20
```

Lookbehind

Vérifie que quelque chose précède sans le capturer

```python
import re

pattern = re.compile(r'(?<=\$)\d+')

result = pattern.search('Prix : $100')

print(result.group())  # Affiche : 100
```

Test avec try except

Le bloc **try except** permet de gérer ces erreurs et d'éviter que le programme ne se termine brutalement en cas erreur de syntaxe dans le motif.

```
import re
texte = 'Ce texte contient un motif.'
try:
    pattern = re.compile(r'motif(?m)')
    result = pattern.search(texte)
    print(result.group())
except re.error:
    print("Erreur dans l'expression régulière")
```

Cas du motif introuvable

```python
import re
texte = 'Ce texte contient un motif.'
try:
    pattern = re.compile(r'motig')
    result = pattern.search(texte)
except re.error:
    print("Erreur dans l'expression régulière")
else:
    if result:
        print(result.group())
    else:
        print("Motif non trouvé")
```

Exemple

```python
import re

# Regex pour valider une adresse e-mail
pattern = re.compile(r'^([a-z])+[a-z0-9_.-]+@([a-z])+[a-z]+\.[a-z]{2,4}$')

email = "contact@example.com"

# Utilisation de search() avec ancrage, classe et quantificateur
result = pattern.search(email)
if result:
    print("Adresse e-mail valide :", result.group())
else:
    print("Adresse e-mail non valide")
```