# Les expressions régulières

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
Si le motif n’est pas trouvé elle renvoie **None**

```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
```

```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 liste

```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 cpt, 12 cam’) |
| **re.finditer()** | Renvoie un itérateur d'objets avec informations détaillées. | re.finditer(r'\d+', ‘4 cpt, 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']
```

# Drapeaux (flags):

Par défaut, le point (.) correspond à n'importe quel caractère **sauf** les sauts de ligne (\n).

**Avec re.DOTALL** Le (.) correspond à **tout** caractère, y compris les sauts de ligne.

```python
import re

texte = "Hello\nWorld"
pattern = r"Hello.*World"

match = re.search(pattern, texte, re.DOTALL)
print(match.group())  # Affiche "Hello\nWorld"
```

# 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']
```

# \s et la fonction split()

Ce code permet de diviser la chaîne **"le langagae python3 est excellent!"** en une liste de mots en se basant sur un ou plusieurs espaces (\s+)

```
import re

chaine = "le langagae python3   est excellent!"

maListe = re .split(r'\s+',chaine)

print(maListe) #affiche  ['le', 'langagae', 'python3', 'est', 'excellent!']
```

# 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}** : exactement n 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.

**|** : "OU" logique

# Le métacaractère \b

Le métacaractère \b s'assure que seul le mot **python** est capturé.

```
import re

chaine = "le langagae python3 est excellent!"

# regex = re.compile(r'python') # trouve le motif python

regex = re.compile(r'\bpython\b') # Ne touve pas le motif
```

# Le métacaractère |

L'opérateur | (pipe) est utilisé pour indiquer une alternative entre plusieurs motifs.

Il correspond à un "OU" logique entre plusieurs expressions régulières..

```python
import re
logs_reseau = """
2024-10-12 10:15:32 Protocole: TCP Connexion établie
2024-10-12 10:16:45 Protocole: UDP Connexion établie
2024-10-12 10:17:58 Protocole: TCP Connexion terminée
"""
pattern = r"TCP|UDP"

result = re.findall(pattern, logs_reseau)
print(result)  # ['TCP', 'UDP', 'TCP']
```

# Le métacaractère ^ et $

Pour vérifier si la chaine commence par un caractère **^** et se termine par un caractère $ Il faut les échapper avec un antislash : **\^** et **\$**

```
import re

chaine = '^ :au début et fini par: $'

regex = re.compile(r'^\^.*\$$')

resulat = regex.search(chaine)
```

# Différence entre ". " et "\. "

Il faut **échapper le point** avec un antislash (\.) dans une expression régulière si tu veux le chercher littéralement.

En regex, le point (.) est un métacaractère qui correspond à **n'importe quel caractère** (sauf le retour à la ligne)

```python
import re

chaine = 'Une phrase commence par une majuscule et se termine par un point.'

regex = re.compile(r'^[A-Z].*\.$')

resulat = regex.search(chaine)
```

# Le caractère "/" n’est pas un métacaractère

En Python, le caractère **/** n'est **pas un métacaractère**.

```python
import re

texte = "https://www.example.com/page"
pattern = r"https://www\.example\.com/page"

match = re.findall(pattern, texte)
print(match)  # Affiche ['https://www.example.com/page']
```

# Le métacaractère "\"

Pour vérifier si la chaîne **"\Le back slash"** contient un backslash (\)

il est plus simple d'utiliser une chaîne "raw" (avec r'\\') pour représenter le backslash dans une regex

```python
import re

chaine = r'\Le back slash'

regex = re.compile('\\\\')
# regex = re.compile(r'\\') # plus simple avec les raw string

resulat = regex.search(chaine)
```

# 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
```

# Lookahead négatif

S'assure qu'un motif **n'est pas suivi** d'un autre motif.

```python
import re

texte = "Il y a 100€ et 200$ ici."

pattern = r'\b\d+(?!€)\b'

match = re.findall(pattern, texte)
print(match)  # Affiche ['200']
```

# 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
```

# Lookbehind négatif

Vérifie que quelque chose précède sans le capturer

```python
import re

texte = "Il y a 100€ et 200$ ici."

# Regex avec lookahead négatif
pattern = r'\b\d+\b(?!€)'

match = re.findall(pattern, texte)
print(match)  # Affiche ['200']
```

# Vérifier la présence d’une majuscule

Vérifie si une chaîne contient **au moins une lettre majuscule** quelque part

```python
import re

chaine = "koRri"

regex = re.compile(r'^(?=.*[A-Z]).*')

resultat = regex.search(chaine)
if resultat:
    print(resultat.group())
else:
    print("Motif non trouvé")
```

# 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.

```python
import re
texte = 'Ce texte contient un motif.'

try:
    # (?m) est une erreur dans la regex
    pattern = re.compile(r'motif(?m)')
    result = pattern.search(texte)
    print(result.group())
except:
    print("Erreur dans l'expression régulière")
```

# Cas où le motif est 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")
```

# Groupe de capture

Les **groupes capturants** ( ) permettent de capturer des sous-chaînes spécifiques dans la regex.

Le premier groupe capture le **pourcentage**, et le second capture le **prix**.

```python
import re
texte = "20% de réduction sur un produit à 100€, et 30% sur un autre à 250€"
# Regex pour capturer les pourcentages suivis d'un prix
pattern = r'(\d+)%.*?(\d+)€'

matches = re.finditer(pattern, texte)

for match in matches:
    pourcentage = match.group(1)  # Correspond au nombre avant %
    prix = match.group(2)         # Correspond au nombre après €
    print(f"Pourcentage : {pourcentage}%, Prix : {prix}€")
```

# Groupe non capturant

```python
import re
logs = """
Device IoT_123 | Status: ERROR 500
Device IoT_456 | Status: OK
Device IoT_789 | Status: ERROR 404
"""
pattern = r"Device (IoT_\d+) \| (?:Status: ERROR) (\d+)"
matches = re.finditer(pattern, logs)
for match in matches:
    device_id = match.group(1)  # Capture l'ID de l'appareil
    error_code = match.group(2)  # Capture le code d'erreur
    print(f"Appareil : {device_id}, Code d'erreur : {error_code}")
```