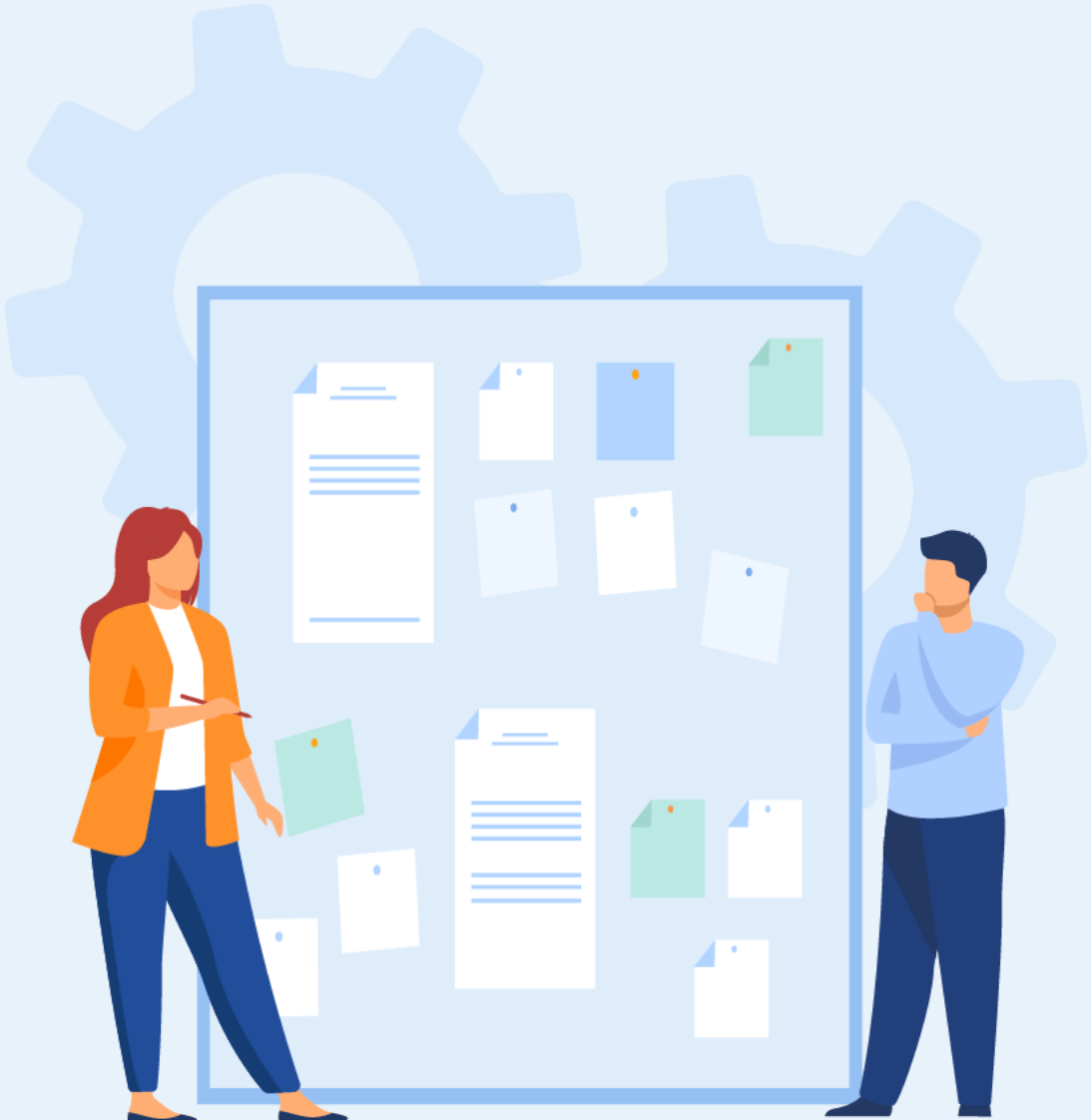


CHAPITRE 3

Utiliser les expressions régulières

Expressions régulières de base

- Les expressions régulières fournissent une notation générale permettant de décrire abstraitement des éléments textuels
- Une expression régulière se lit de gauche à droite.
- Elle constitue ce qu'on appelle traditionnellement un motif de recherche.
- Elles utilisent **six** symboles qui, dans le contexte des expressions régulières, acquièrent les significations suivantes :

1. **le point .** représente une seule instance de n'importe quel caractère sauf le caractère de fin de ligne.

Exemple : l'expression t.c représente toutes les combinaisons de trois lettres commençant par « t » et finissant par « c », comme tic, tac, tqc ou t9c

2. **la paire de crochets [ ]** représente une occurrence quelconque des caractères qu'elle contient.

Exemple : [aeiouy] représente une voyelle, et Duran[dt] désigne Durand ou Durant.

- Entre les crochets, on peut noter un intervalle en utilisant le tiret.

Exemple : [0-9] représente les chiffres de 0 à 9, et [a-zA-Z] représente une lettre minuscule ou majuscule.

- On peut de plus utiliser l'accent circonflexe en première position dans les crochets pour indiquer le contraire de

Exemple : [^a-z] représente autre chose qu'une lettre minuscule, et [^'"] n'est ni une apostrophe ni un guillemet.

3. **l'astérisque *** est un quantificateur, il signifie aucune ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

Exemple : L'expression ab* signifie la lettre a suivie de zéro ou plusieurs lettres b, par exemple ab, a ou abbb et [A-Z]* correspond à zéro ou plusieurs lettres majuscules.

Expressions régulières de base

4. **l'accent circonflexe ^** est une ancre. Il indique que l'expression qui le suit se trouve en début de ligne.

Exemple : l'expression ^Depuis indique que l'on recherche les lignes commençant par le mot Depuis.

5. **le symbole dollar $** est aussi une ancre. Il indique que l'expression qui le précède se trouve en fin de ligne.

Exemple : L'expression suivante :$ indique que l'on recherche les lignes se terminant par « suivante : ».

6. **la contre-oblique ** permet d'échapper à la signification des métacaractères. Ainsi \. désigne un véritable point, * un astérisque, \^ un accent circonflexe, $ un dollar et \\ une contre-oblique

Expressions régulières étendues

- Elles ajoutent cinq symboles qui ont les significations suivantes :

1. **la paire de parenthèses ( ):** est utilisée à la fois pour former des sous-motifs et pour délimiter des sous-expressions, ce qui permettra d'extraire des parties d'une chaîne de caractères.

Exemple : L'expression (to)* désignera to, tototo, etc.

2. **le signe plus +:** est un quantificateur comme *, mais il signifie une ou plusieurs occurrences du caractère ou de l'élément qui le précède immédiatement.

Exemple : L'expression ab+ signifie la lettre a suivie d'une ou plus

3. **le point d'interrogation ?:** il signifie zéro ou une instance de l'expression qui le précède.

Exemple : écran(s)? désigne écran ou écrans ;

Expressions régulières étendues

4. **la paire d'accolades { } :** précise le nombre d'occurrences permises pour le motif qui le précède.

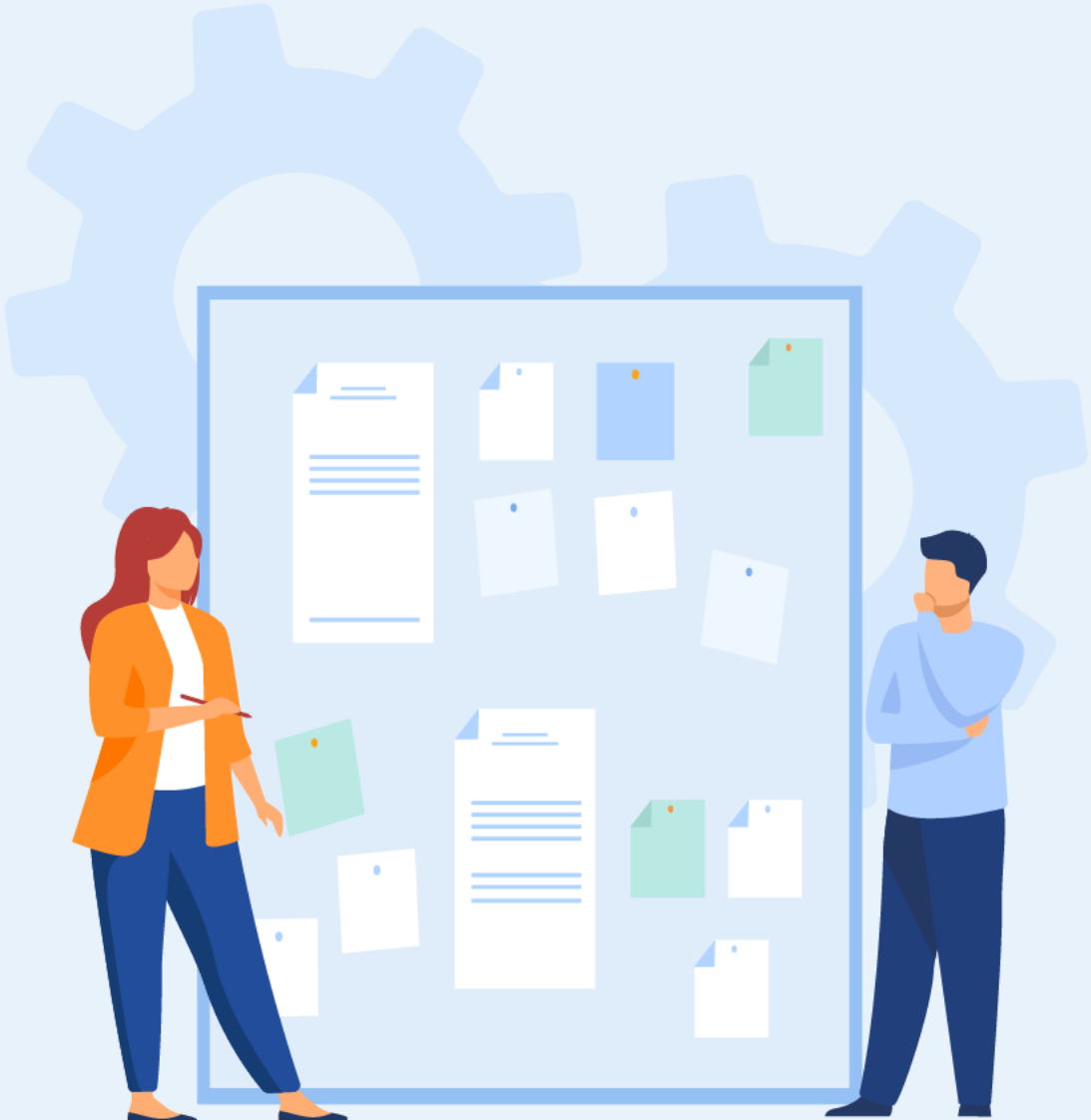
Exemple : [0-9]{2,5} attend entre deux et cinq nombres décimaux.

- Les variantes suivantes sont disponibles : [0-9]{2,} signifie au minimum deux occurrences d'entiers décimaux et [0-9]{2} deux occurrences exactement

5. **la barre verticale | :** représente des choix multiples dans un sous-motif.

Exemple : L'expression Duran[d|t] peut aussi s'écrire (Durand|Durant). On pourrait utiliser l'expression (lu|ma|me|je|ve|sa|di) dans l'écriture d'une date.

- **la syntaxe étendue comprend aussi une série de séquences d'échappement :**
  - \ : symbole d'échappement ;
  - \e : séquence de contrôle escape ;
  - \f : saut de page ;
  - \n : fin de ligne ;
  - \r : retour-chariot ;
  - \t : tabulation horizontale ;
  - \v : tabulation verticale ;
  - \d : classe des nombres entiers ;
  - \s : classe des caractères d'espacement ;
  - \w : classe des caractères alphanumériques ;
  - \b : délimiteurs de début ou de fin de mot ;
  - \D : négation de la classe \d ;
  - \S : négation de la classe \s ;
  - \W : négation de la classe \w ;
  - \B : négation de la classe \b.



CHAPITRE 3

Utiliser les expressions régulières

1. Création des expressions régulières
2. **Manipulation des expressions régulières**

- Le module re permet d'utiliser les expressions régulières dans les scripts Python.
- Les scripts devront donc comporter la ligne : **import re**
- **Fonction de compilation d'une regex en Python:**
  - Pour initialiser une expression régulière avec Python, il est possible de la compiler, surtout si vous serez amené à l'utiliser plusieurs fois tout au long du programme. **Pour ce faire, il faut utiliser la fonction compile()**

```
import re
expression =re.compile(r'\d{1,3}')
```

Les options de compilation :

- Grâce à un jeu d'options de compilation, il est possible de piloter le comportement des expressions régulières. On utilise pour cela la syntaxe (?…) avec les drapeaux suivants :
  - **a :** correspondance ASCII (Unicode par défaut) ;
  - **i :** correspondance non sensible à la casse ;
  - **L :** les correspondances utilisent la locale, c'est-à-dire les particularités du pays ;
  - **m :** correspondance dans des chaînes multilignes ;
  - **s :** modifie le comportement du métacaractère point qui représentera alors aussi le saut de ligne ;
  - **u :** correspondance Unicode (par défaut) ;
  - **x :** mode verbeux.

```
import re
expression =re.compile(r"[a-z]+")  #pilotage du comportement de l'expression régulière avec une sensibilité à la case
expression =re.compile(r" (?i) [a-z]+") #pilotage du comportement de l'expression régulière sans une sensibilité à la case
```

PARTIE 3

- Le module ‘re’ propose un ensemble de fonctions qui nous permet de rechercher une chaîne pour une correspondance :

Fonction	Description
Findall	Renvoie une liste contenant toutes les correspondances
Search	Envoie un objet Match s’il existe une correspondance n’importe où dans la chaine
split	Renvoie une liste où la chaine a été divisée à chaque correspondance
sub	Remplace une ou plusieurs correspondances par une chaine ordonnée

- **La fonction findall()** renvoie une liste contenant toutes les correspondances. La liste contient les correspondances dans l’ordre où elles sont trouvées. Si aucune correspondance n’est trouvée, une liste vide est renvoyée.

Exemple :

```
import re
Nameage = '''Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21'''
ages = re.findall(r'\d{1,3}', Nameage)  #chercher de un, deux ou trois entiers décimaux
print(ages)                             #affiche ['22', '33', '44', '21']
names = re.findall(r'[A-Z][a-z]*',Nameage)  #chaines contenant des miniscules et des majuscules
Print(names)                            #affiche ['Janice', 'Theon', 'Gabriel', 'Joey']
```

PARTIE 3

- La fonction search() recherche une correspondance dans la chaîne et renvoie un objet Match s'il existe une correspondance. S'il y a plus d'une correspondance, seule la première occurrence de la correspondance sera renvoyée:

Exemple : Extraction simple

- La variable expression reçoit la forme compilée de l'expression régulière, Puis on applique à ce motif compilé la méthode search() qui retourne la première position du motif dans la chaîne Nameage et l'affecte à la variable ages. Enfin on affiche la correspondance complète (en ne donnant pas d'argument à group())

```
import re
Nameage = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
expression = re.compile(r'\d{1,3}')  #chercher de un, deux ou trois entiers décimaux
ages=expression.search(Nameage) #retourne la première position du motif
print(ages) #affiche: <re.Match object; span=(11, 13), match='22'>
```

Position du résultat

Résultat =22

Exemple : Extraction des sous-groupes

- Il est possible d'affiner l'affichage du résultat en modifiant l'expression régulière de recherche de façon à pouvoir capturer les éléments du motif

```
import re
motif_date = re.compile(r"(\d\d ?) (\w+) (\d{4})")
corresp = motif_date.search("Bastille le 14 juillet 1789")

print("corresp.group() :", corresp.group())          #corresp.group() : 14 juillet 1789
print("corresp.group(1) :", corresp.group(1))        #corresp.group(1) : 14
print("corresp.group(2) :", corresp.group(2))        #corresp.group(2) : juillet
print("corresp.group(3) :", corresp.group(3))        #corresp.group(3) : 1789
print("corresp.group(1,3) :", corresp.group(1,3))    #corresp.group(1,3) : ('14', '1789')
print("corresp.groups() :", corresp.groups())        #corresp.groups() : ('14', 'juillet', '1789')
```

- Python possède une syntaxe qui permet de nommer des parties de motif délimitées par des parenthèses, ce qu'on appelle un motif nominatif :
  - syntaxe de création d'un motif nominatif : (?P<nom_du_motif>) ;
  - syntaxe permettant de s'y référer : (?P=nom_du_motif) ;

Exemple : Extraction des sous-groupes nommés

- la méthode **groupdict()** renvoie une liste comportant le nom et la valeur des sous-groupes trouvés (ce qui nécessite de nommer les sous-groupes).

```
import re
motif_date = re.compile(r"(?P<jour>\d\d ?) (?P<mois>\w+) (\d{4})")
corresp = motif_date.search("Bastille le 14 juillet 1789")

print(corresp.groupdict())   #{'jour': '14', 'mois': 'juillet'}
print(corresp.group('jour')) #14
print(corresp.group('mois')) #juillet
```

- **La fonction split()** renvoie une liste où la chaîne a été divisée à chaque correspondance
- L'exemple suivant divise la chaîne à chaque espace trouvé. '\s' est utilisé pour faire correspondre les espaces.

```
import re
adresse="Rue 41 de la République"
liste=re.split("\s",adresse)
print(liste)  #affiche: ['Rue', '41', 'de', 'la', 'République']
```

- Il est possible de contrôler le nombre d'occurrences en spécifiant le paramètre **maxsplit**:

Exemple : maxsplit=1

```
import re
adresse="Rue 41 de la République"
liste=re.split("\s",adresse,1)
print(liste)   #affiche ['Rue', '41 de la République']
```

- **La fonction sub()** remplace les correspondances par le texte de votre choix

Exemple : Remplacer chaque espace par un tiret '-':

```
import re
adresse="Rue 41 de la République"
liste=re.sub("\s","-",adresse)  #affcihe: Rue-41-de-la-République
print(liste)
```

- Il est possible de contrôler le nombre de remplacements en spécifiant le paramètre **count**:

Exemple : count=2

```
import re
adresse="Rue 41 de la République"
liste=re.sub("\s","-",adresse,2) #affiche Rue-41-de la République
print(liste)
```