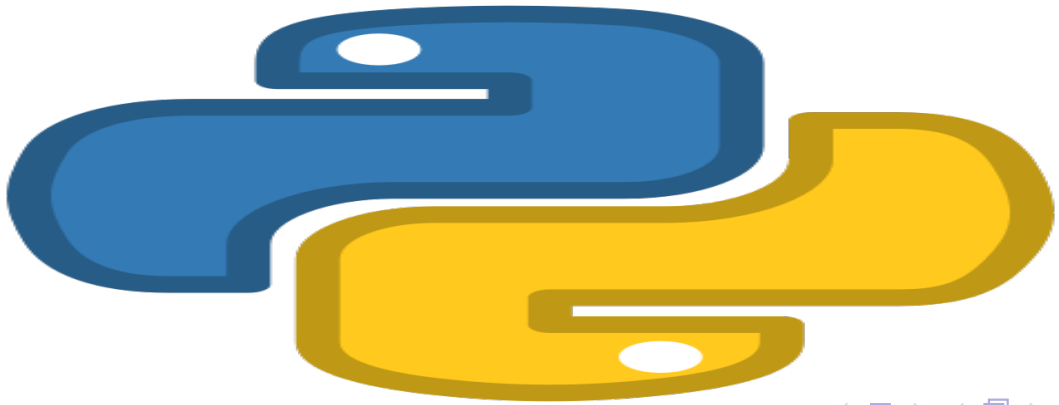
# **Python** : expressions régulières

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



# Plan

# Python

## Expressions régulières

- outil de recherche puissant adopté par la plupart des langages de programmation
- facilitant la recherche dans les chaînes de caractères (et donc le remplacement, le calcul de nombre d'occurrences...)
- permettent de vérifier si des chaînes de caractères respectent certains formats (email, numéro de téléphone...)
- définies dans le module `re` de la **STDLIB**
- syntaxe quasi-similaire quel que soit le langage de programmation

# Python

## Quatre méthodes de recherche disponibles

- `search(motif, chaine)` : permet de chercher `motif` dans `chaine` et retourner soit un objet de correspondance (`Match`) soit `None`
- `match(motif, chaine)` : fonctionne comme `search` mais cherche seulement au début de la chaîne
- `fullmatch(motif, chaine)` : vérifie si la totalité de la chaîne correspond exactement au motif.
- `findall(motif, chaine)` : retourne une liste contenant toutes les occurrences de motif dans chaine
- `split(motif, chaine)` : permet de découper `chaine` selon le motif défini dans `motif`
- `sub(motif, new, chaine)` : permet de remplacer toutes les occurrences de `motif` dans `chaine` par `new`
- `finditer(motif, chaine)` : permet de retourner un itérateur contenant des objets `Match` : un pour chaque correspondance

# Python

**Recherche d'une sous-chaîne dans une chaîne de caractère**

```
import re

string = "Bonjour tout le monde"
resultat = re.search("Tout", string)

if resultat:
    print(resultat)
else:
    print("Non trouvé")

# affiche Non trouvé
```

# Python

**Recherche avec une expression régulière insensible à la casse**

```
import re

string = "Bonjour tout le monde"
resultat = re.search("Tout", string, re.IGNORECASE)

if resultat:
    print(resultat)
else:
    print("Non trouvé")

# affiche <re.Match object; span=(8, 12), match='tout'>
```

# Python

## Valeur de retour de `search` (et `match`) : objet `Match` avec les méthodes suivantes

- `group()` : la correspondance exacte
- `start()` : l'indice de début de la correspondance
- `end()` : l'indice de fin de la correspondance
- `end()` : un tuple contenant l'indice de début et l'indice de fin de la correspondance
- `groups()` : un tuple contenant les correspondances

# Python

**Cependant, la méthode `match` vérifie la présence d'un motif seulement au début de la chaîne**

```
import re

string = "Bonjour tout le monde"
resultat = re.match("Tout", string, re.IGNORECASE)

if resultat:
  print("Trouvé")
else:
  print("Non trouvé")

# affiche Non trouvé
```

# Python

**La méthode `match` trouve bien le motif `Bon`**

```
import re

string = "Bonjour tout le monde"
resultat = re.match("Bon", string, re.IGNORECASE)

if resultat:
  print("Trouvé")
else:
  print("Non trouvé")

# affiche Trouvé
```

# Python

**Pour chercher toutes les occurrences d'un motif**

```
import re

string = "Bonjour tout le monde"
resultat = re.findall("on", string, re.IGNORECASE)

print(resultat)

# affiche ['on', 'on']
```

# Python

**Pour remplacer toutes les occurrences d'un motif**

```
import re

string = "Bonjour tout le monde"
resultat = re.sub("on", "an", string)

print(resultat)

# affiche Banjour tout le mande
```

# Python

**Pour indiquer le nombre d'occurrences à remplacer**

```
import re

string = "Bonjour tout le monde"
resultat = re.sub("on", "an", string, 1)

print(resultat)

# affiche Banjour tout le monde
```

# Python

**Pour découper une chaîne selon un motif (ici un espace)**

```
import re

string = "Bonjour tout le monde"
resultat = re.split(" ", string)

print(resultat)

# affiche ['Bonjour', 'tout', 'le', 'monde']
```

# Python

**On peut aussi indiquer le nombre d'occurrence à considérer pour le motif de découpage**

```
import re

string = "Bonjour tout le monde"
resultat = re.split(" ", string, 2)

print(resultat)

# affiche ['Bonjour', 'tout', 'le monde']
```

## Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$ : minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$ : exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$ : minimum `n` occurrences de `a` consécutives

## Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$ : minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$ : exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$ : minimum `n` occurrences de `a` consécutives



```
print(re.findall("ba?c", "bacbaacbc"))
# affiche Banjour tout le monde
```

## Contraintes exprimées avec les expressions régulières

- `a+` : 1 ou plusieurs `a`
- `a*` : 0 ou plusieurs `a`
- `a?` : 0 ou 1 `a`
- `a`$\{n, m\}$ : minimum `n` occurrences de `a` consécutives, maximum `m` occurrences de `a` consécutives
- `a`$\{n\}$ : exactement `n` occurrences de `a` consécutives
- `a`$\{n, \}$ : minimum `n` occurrences de `a` consécutives



```
print(re.findall("ba?c", "bacbaacbc"))
# affiche Banjour tout le monde
```

```
print(re.findall("ba{0,1}c", "bacbaacbc"))
# affiche Banjour tout le monde
```

# Python

## Contraintes exprimées avec les expressions régulières

- $a \mid b$ : `a` ou `b`
- `^` : commence par
- `$` : se termine par
- `()` : le groupe
- `a(?!b)` : `a` non suivi de `b`
- `a(?=b)` : `a` suivi de `b`
- `(?<!a)b` : `b` non précédé par `a`

# Python

## Contraintes exprimées avec les expressions régulières

- `.` : n'importe quel caractère
- `\d` : un chiffre
- `\D` : tout sauf un chiffre
- `\w` : un caractère alphanumérique ou _
- `\W` : tout sauf un caractère alphanumérique
- `\t` : un caractère de tabulation
- `\n` : un caractère de retour à la ligne
- `\s` : un espace
- ...

# Python

## Contraintes exprimées avec les expressions régulières

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule ou en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- . . .

# Python

## Contraintes exprimées avec les expressions régulières

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule ou en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- ...

**Pour utiliser un caractère réservé (^, $...) dans une expression régulière, il faut le précéder par **

# Python

## Exercice 1

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère contient 3 occurrences (pas forcement consécutives) de la sous-chaîne `ab`.

- `true` pour `abacccababcc`
- `true` pour `abababccccab`
- `false` pour `baaaaabaccccba`

# Python

## Exercice 2

Trouver une expression régulière qui permet de déterminer si une chaîne commence par la lettre `a`, se termine par la lettre `b` et pour chaque `x` suivi de `y` (pas forcement consécutives), il existe un `z` situé entre `x` et `y`.

- `true` pour `ab`
- `true` pour `abxyb`
- `true` pour `abxazyb`
- `false` pour `abxuyb`
- `false` pour `abxazyxyb`

# Python

## Exercice 3

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à une adresse e-mail.

# Python

## Exercice 3

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à une adresse e-mail.

## Exercice 4

Trouver une expression régulière qui permet de déterminer si une chaîne de caractère correspond à un numéro de téléphone français.

# Python

**Pour vérifier si une chaîne est un numéro de téléphone français valide**

```
import re

motif = '\d{2} \d{2} \d{2} \d{2} \d{2}'

string = "01 23 45 67 89"

if re.fullmatch(motif, string):
    print(f"'{string}' est un numéro valide.")
else:
    print(f"'{string}' n'est pas valide.")

# affiche '01 23 45 67 89' est un numéro valide.
```

# Python

**La chaîne suivante est validée par `match` mais pas par `fullmatch`**

```
import re

motif = '\d{2} \d{2} \d{2} \d{2} \d{2}'

string = "01 23 45 67 89 suite"

if re.fullmatch(motif, string):
    print(f"'{string}' est un numéro valide.")
else:
    print(f"'{string}' n'est pas valide.")

# affiche '01 23 45 67 89 suite' n'est pas valide.

if re.match(motif, string):
    print(f"'{string}' est un numéro valide.")
else:
    print(f"'{string}' n'est pas valide.")

# affiche '01 23 45 67 89 suite' est un numéro valide.
```

# Python

**Pour récupérer un itérateur d'objet `Match` de toutes les correspondances**

```
import re

string = "Bonjour tout le monde"
motif = 'o[nu]'

resultat = list(re.finditer(motif, string, re.IGNORECASE))

if resultat:
    for match in resultat:
        print(f"{match.group()} trouvé à la position {match.
            start()}-{match.end()}")
else:
    print("Non trouvé")

# on trouvé à la position 1-3
# ou trouvé à la position 4-6
# ou trouvé à la position 9-11
# on trouvé à la position 17-19
```