# Regular expressions

## 1-explique moi ce code ligne par ligne comme si je ne savais rien en programmation: "

**import re**

email = input("What's your email?: ").strip()

if re.search(r".+@.+\.edu", email):

  print("Valid")

else:

  print("Invalid")

 ", apres explique moi ca " 1). 2)\* 3)+ 4)? 5){m}  6){m,n} "dans les regular expressions en commenacant par l'exemple la plus facile.

**Explication ligne par ligne du code**

**1. \*\*`import re`\*\***

   On importe le module `re` (Regular Expressions). C'est une boîte à outils pour analyser et manipuler du texte en cherchant des motifs spécifiques.

**2. \*\*`email = input("What's your email?: ").strip()`\*\***

   - `input(...)`: Demande à l'utilisateur de saisir son email.

   - `.strip()`: Supprime les espaces au début et à la fin de la saisie (ex: `"  john@example.com  "` devient `"john@example.com"`).

   - `email`: Variable qui stocke l'email nettoyé.

**3. `if re.search(r".+@.+\.edu", email):`\*\***

   - `re.search(...)`: Cherche si le motif décrit est présent dans l'email.

   - `r".+@.+\.edu"`: Le motif (regex) à chercher :

  - `.` : Correspond à n'importe quel caractère (sauf un saut de ligne).

  - `+` : Le caractère précédent doit apparaître \*\*1 fois ou plus\*\*.

  - `\.edu` : Le texte doit se terminer par `.edu` (le `\` est nécessaire pour "échapper" le point, sinon `.` signifie "n'importe quel caractère").

   Exemple de correspondance : `a@b.edu`, `john@univ.edu`.

**4. `print("Valid")`**

   Si le motif est trouvé, l'email est déclaré "Valid".

**5. `else:`**

   Si le motif n'est pas trouvé, exécute le bloc suivant.

**6. `print("Invalid")`**

   Affiche "Invalid".

---

## 

## **Explication des symboles des expressions régulières (du plus simple au plus complexe)**

### `\*` (Astérisque)

   - \*\*Signification\*\* : "Zéro ou plusieurs occurrences" du caractère précédent.

   - \*\*Exemple\*\* :

  - `a\*` correspond à `""` (rien), `"a"`, `"aa"`, `"aaa"`, etc.

  - `bo\*b` correspond à `"bb"`, `"bob"`, `"booob"`, etc.

### `+` (Plus)

   - \*\*Signification\*\* : "Une ou plusieurs occurrences" du caractère précédent.

   - \*\*Exemple\*\* :

  - `a+` correspond à `"a"`, `"aa"`, etc., mais \*\*pas\*\* à `""`.

  - `bo+b` correspond à `"bob"`, `"boob"`, mais \*\*pas\*\* à `"bb"`.

---

### 

### **`?` (Point d'interrogation)**

   - \*\*Signification\*\* : "Zéro ou une occurrence" du caractère précédent (optionnel).

   - \*\*Exemple\*\* :

  - `colou?r` correspond à `"color"` (le `u` est optionnel) et `"colour"`.

### 

### **`{m}` (Accolade avec un nombre)**

   - \*\*Signification\*\* : "Exactement \*\*m\*\* occurrences" du caractère précédent.

   - \*\*Exemple\*\* :

  - `a{3}` correspond uniquement à `"aaa"`.

### `{m,n}` (Accolade avec deux nombres)

   - \*\*Signification\*\* : "Entre \*\*m\*\* et \*\*n\*\* occurrences" du caractère précédent.

   - \*\*Exemple\*\* :

  - `a{2,4}` correspond à `"aa"`, `"aaa"`, ou `"aaaa"`.

### 6. \*\*`.` (Point)

   - \*\*Signification\*\* : "N'importe quel caractère unique" (sauf un saut de ligne).

   - \*\*Exemple\*\* :

  - `a.c` correspond à `"abc"`, `"aXc"`, `"a3c"`, etc.

### Résumé des Symboles :

| Symbole | Exemple    | Correspond à                          |

|---------|------------|----------------------------------------|

| `\*` | `a\*`   | `""`, `"a"`, `"aa"`, ...          |

| `+` | `a+`   | `"a"`, `"aa"`, ... (pas `""`)     |

| `?` | `a?`   | `""` ou `"a"`                     |

| `{m}`   | `a{3}` | Exactement `"aaa"`                |

| `{m,n}` | `a{2,4}`   | `"aa"`, `"aaa"`, `"aaaa"`         |

| `.` | `a.c`  | Un caractère quelconque entre `a` et `c` |

### 

### **Explication des symboles `^` et `$` dans les expressions régulières :**

Les symboles `^` et `$` sont utilisés pour ancrer le motif au \*\*début\*\* ou à la \*\*fin\*\* d'une chaîne de caractères. Ils sont essentiels pour valider des formats précis.

---

### 

### 1. \*\*`^` (Caret)\*\*

- \*\*Signification\*\* : Le motif doit commencer \*\*exactement ici\*\*, au début de la chaîne.

- \*\*Exemple\*\* :

  - `^Bonjour` correspond à `"Bonjour le monde"` mais \*\*pas\*\* à `"Dis Bonjour"`.

  - `^a` : La chaîne doit commencer par `a` (ex: `"apple"` ✅, `"banana"` ❌).

### \*\*`$` (Dollar)

- \*\*Signification\*\* : Le motif doit se terminer \*\*exactement ici\*\*, à la fin de la chaîne.

- \*\*Exemple\*\* :

  - `monde$` correspond à `"Bonjour le monde"` mais \*\*pas\*\* à `"monde entier"`.

  - `a$` : La chaîne doit finir par `a` (ex: `"pizza"` ✅, `"banane"` ❌).

---

#### Pourquoi les utiliser ensemble ?

Quand on combine `^` et `$`, on s'assure que \*\*toute la chaîne\*\* correspond au motif, pas juste une partie.

C'est crucial pour des validations strictes (ex: emails, mots de passe).

---

#### 

#### Exemple avec l'email :

Dans le code initial, le motif était `.+@.+\.edu`, qui pourrait correspondre à des chaînes comme :

- `"user@domain.edu"` ✅

- `"xxxuser@domain.edu.xxx"` ✅ (car `.edu` est présent \*quelque part\*).

Pour éviter ce problème, on utilise `^` et `$` pour forcer la \*\*correspondance totale\*\* :

```python

if re.search(r"^.+@.+\.edu$", email):

```

- `^` : Début de la chaîne.

- `.+@.+\.edu` : Le motif classique.

- `$` : Fin de la chaîne.

Maintenant, \*\*seuls\*\* les emails qui \*\*finissent exactement par `.edu`\*\* seront acceptés :

- ✅ `"user@harvard.edu"`

- ❌ `"user@harvard.edu.com"` (car `.edu` n'est pas à la fin).

---

### Résumé des symboles d'ancrage :

| Symbole | Exemple      | Correspond à                          |

|---------|--------------|----------------------------------------|

| `^` | `^abc`   | La chaîne \*\*commence\*\* par `abc`   |

| `$` | `xyz$`   | La chaîne \*\*finit\*\* par `xyz`      |

| `^...$` | `^a...z$`    | La chaîne entière suit le motif `a...z` |

---

#### Application à ton code :

Pour améliorer la validation de l'email, utilise :

```python

if re.search(r"^.+@.+\.edu$", email):

```

Cela garantit que l'email est \*\*exactement\*\* de la forme `xxx@xxx.edu`, sans rien avant ou après.

#### 

#### **Explication des symboles `[]` et `[^]` dans les expressions régulières :**

Les crochets `[]` permettent de définir un \*\*ensemble de caractères autorisés\*\* à un endroit précis du motif. C'est utile pour accepter des choix spécifiques ou des plages de caractères.

---

#### `[]` (Crochets simples)

- \*\*Signification\*\* : "Un \*\*seul\*\* caractère parmi ceux listés dans les crochets".

- \*\*Exemple\*\* :

  - `[aeiou]` : Correspond à \*\*une seule voyelle\*\* (a, e, i, o, u).

- ✅ `"a"`, `"e"`

- ❌ `"b"`, `"zz"`

  - `[A-Za-z]` : Correspond à \*\*une seule lettre\*\* (majuscule ou minuscule).

  - `[0-9]` : Correspond à \*\*un seul chiffre\*\*.

---

#### `[^]` (Crochets avec caret `^`)

- \*\*Signification\*\* : "Un \*\*seul\*\* caractère qui \*\*n'est pas\*\* dans la liste".

- \*\*Attention\*\* : Le `^` doit être le \*\*premier caractère\*\* dans les crochets pour indiquer une négation.

- \*\*Exemple\*\* :

  - `[^aeiou]` : Correspond à \*\*une consonne\*\* ou tout caractère non-voyelle.

- ✅ `"b"`, `"3"`, `"#"`

- ❌ `"a"`, `"e"`

  - `[^0-9]` : Tout caractère \*\*non-chiffre\*\*.

---

#### Cas pratiques :

#### Exemple 1 : Valider un code à 3 chiffres

```python

re.search(r"^[0-9]{3}$", code)  # Ex: "123" ✅, "12A" ❌

```

- `[0-9]` : Un chiffre entre 0 et 9.

- `{3}` : Exactement 3 occurrences.

#### ---

#### 

#### **Exemple 2 : Trouver des voyelles dans un mo**t

```python

re.findall(r"[aeiou]", "Bonjour")  # Retourne ["o", "u"]

```

---

#### Exemple 3 : Exclure des caractères

```python

re.search(r"^[^@]+$", texte)  # ✅ Si le texte ne contient PAS de "@"

```

- `[^@]` : N'importe quel caractère sauf `@`.

- `+` : Un ou plusieurs caractères.

---

#### 

#### Cas spéciaux dans `[]` :

- \*\*Plages\*\* :

  - `[a-z]` : Toutes les lettres minuscules.

  - `[A-Za-z]` : Lettres sans chiffres.

**- \*\*Caractères spéciaux\*\* :**

  - Pour inclure un `-` littéral, placez-le en \*\*premier\*\* ou \*\*dernier\*\* : `[-abc]` ou `[abc-]`.

  - Pour inclure un `^` littéral, ne le mettez pas en premier : `[a^bc]`.

  - Pour inclure un `]`, échappez-le avec `\` : `[a-z\]]`.

**---**

#### Comparaison avec d'autres symboles :

| Symbole | Exemple       | Correspond à                          |

|---------|---------------|----------------------------------------|

| `[]` | `[aeiou]` | Un seul caractère parmi a, e, i, o, u |

| `[^]`   | `[^0-9]`  | Un seul caractère qui n'est pas un chiffre |

| `.` | `a.c`     | Un caractère quelconque entre `a` et `c` |

---

#### 

#### **Application à la validation d'ema**il :

Pour améliorer le code précédent, on pourrait utiliser `[]` pour restreindre les caractères autorisés avant le `@` :

```python

if re.search(r"^[A-Za-z0-9\_.+-]+@[A-Za-z0-9-]+\.[A-Za-z]{2,}$", email):

```

- `[A-Za-z0-9\_.+-]` : Caractères alphanumériques, underscores, points, `+`, `-`.

- `[A-Za-z]{2,}` : Extension de domaine (comme `.com`, `.edu`) avec au moins 2 lettres.

---

#### 

#### **Résumé :**

- `[]` : \*\*Inclusion\*\* de caractères.

- `[^]` : \*\*Exclusion\*\* de caractères.

- Les deux permettent de contrôler finement les caractères autorisés à une position donnée.

## 

## Explication des symboles `\d`, `\D`, `\s`, `\S`, `\w`, `\W` dans les expressions régulières :

Ces symboles sont des \*\*raccourcis\*\* pour représenter des catégories de caractères. Ils simplifient l’écriture des motifs.

---

#### 

### `\d` (decimal digit)

- \*\*Signification\*\* : Un \*\*chiffre\*\* (équivalent à `[0-9]`).

- \*\*Exemple\*\* :

  - `\d\d` correspond à `"42"`, `"99"`, etc.

  - `ID-\d{3}` correspond à `"ID-007"`, `"ID-123"`.

---

### 

### `\D` (non-digit)

- \*\*Signification\*\* : Tout caractère \*\*qui n’est pas un chiffre\*\* (équivalent à `[^0-9]`).

- \*\*Exemple\*\* :

  - `\D\D` correspond à `"ab"`, `"!!"`, mais pas à `"a1"` ou `"12"`.

---

### 

### `\s` (whitespace)\*\*

- \*\*Signification\*\* : Un \*\*espace blanc\*\* (espace, tabulation `\t`, saut de ligne `\n`, retour chariot `\r`).

- \*\*Exemple\*\* :

  - `Hello\sWorld` correspond à `"Hello World"`, `"Hello\tWorld"`.

**---**

### 

### `\S` (non-whitespace)

- \*\*Signification\*\* : Tout caractère \*\*qui n’est pas un espace blanc\*\*.

- \*\*Exemple\*\* :

  - `\S\S\S` correspond à `"ABC"`, `"123"`, mais pas à `"A B"` ou `"   "`.

**---**

### `\w` (word character)

- \*\*Signification\*\* : Un caractère "alphanumérique" ou `\_` (équivalent à `[A-Za-z0-9\_]`).

- \*\*Exemple\*\* :

  - `\w+` correspond à `"Hello123"`, `"user\_name"`.

  - Utile pour les noms d’utilisateur : `^\w+$` valide `"john\_doe2"` mais pas `"john@doe"`.

---

### 

### `\W` (non-word character)

- \*\*Signification\*\* : Tout caractère \*\*qui n’est pas un caractère alphanumérique\*\* ou `\_` (symboles, espaces, etc.).

- \*\*Exemple\*\* :

  - `\W` correspond à `"@"`, `" "`, `"#"`, mais pas à `"a"` ou `"3"`.

---

### Tableau récapitulatif :

| Symbole | Signification              | Équivalent      | Exemple de correspondance   |

|---------|--------------------------------|---------------------|----------------------------------|

| `\d` | \*\*Chiffre\*\*                | `[0-9]`         | `"0"`, `"9"`                 |

| `\D` | \*\*Non-chiffre\*\*            | `[^0-9]`        | `"a"`, `"@"`, `" "`              |

| `\s` | \*\*Espace blanc\*\*           | `[ \t\n\r]`     | `" "`, `"\n"`, `"\t"`        |

| `\S` | \*\*Non-espace blanc\*\*       | `[^ \t\n\r]`    | `"A"`, `"3"`, `"#"`              |

| `\w` | \*\*Caractère alphanumérique\*\*   | `[A-Za-z0-9\_]`  | `"a"`, `"3"`, `"\_"`              |

| `\W` | \*\*Non-alphanumérique\*\*     | `[^A-Za-z0-9\_]` | `"@"`, `" "`, `"€"`              |

---

#### 

#### **Cas pratiques avec votre code d’email** :

Pour améliorer la validation de l’email, on pourrait utiliser `\w` pour le nom d’utilisateur et `\S` pour éviter les espaces :

```python

if re.search(r"^\w+@\w+\.\w{2,}$", email):

```

- `\w+` : Nom d’utilisateur avec lettres/chiffres/`\_`.

- `\w{2,}` : Extension de domaine (`.com`, `.edu`, etc.).

- \*\*Problème\*\* : N’accepte pas les `.` ou `-` dans le domaine (ex: `john@mon-domaine.edu` ❌).

---

#### 

#### Exemples concrets :

**1. \*\*Valider un numéro de téléphone\*\* :**

   ```python

   re.search(r"^\d{10}$", phone)  # "0612345678" ✅, "06-12-34-56" ❌

**```**

#### 

#### \*Trouver des hashtags\*\* :

   ```python

   re.findall(r"#\w+", "Hello #Python #2024!")  # Retourne ["#Python", "#2024"]

   `**``**

#### 

#### Détecter des espaces superflus\*\* :

   ```python

   if re.search(r"\s{2,}", texte):  # Avertit si 2 espaces ou plus.

   ```

---

#### 

#### **Astuce :**

- Utilisez `\d` au lieu de `[0-9]` pour plus de lisibilité.

- Utilisez `\W` pour détecter les caractères spéciaux non autorisés (ex: dans un mot de passe).

## 

## **Explication des symboles `A|B`, `(...)` et `(?:...)` dans les expressions régulières :**

Ces symboles servent à gérer des \*\*alternatives\*\* ou à \*\*grouper des parties du motif\*\*, avec ou sans mémorisation du résultat.

---

### 

### 1. \*\*`A|B` (Alternance)\*\*

- \*\*Signification\*\* : "Correspond à \*\*A\*\* \*\*OU\*\* à \*\*B\*\*".

- \*\*Exemple\*\* :

  - `chat|chien` correspond à `"chat"` \*\*ou\*\* `"chien"`.

  - `(je|tu) mange` correspond à `"je mange"` \*\*ou\*\* `"tu mange"`.

⚠️ \*\*Attention\*\* : Sans parenthèses, `|` agit sur tout ce qui est à gauche ou à droite.

Exemple : `a|b+c` signifie `"a"` \*\*ou\*\* `"b+c"` (pas `"a"` ou `"b"` suivi de `"c"`).

### ---

### 

### **2. \*\*`(...)` (Groupe capturant)\*\***

- \*\*Signification\*\* :

  - \*\*Groupe\*\* une partie du motif pour y appliquer des opérations (comme `+`, `\*`, `|`).

  - \*\*Capture\*\* le texte correspondant pour pouvoir le réutiliser (avec `\1`, `\2`, etc.).

- \*\*Exemple\*\* :

  - `(\d{2})-(\d{2})-(\d{4})` pour une date `JJ-MM-AAAA` :

- Capture le jour (`\1`), le mois (`\2`), l’année (`\3`).

  - `(abc)+` correspond à `"abc"`, `"abcabc"`, etc.

---

### 

### 3. \*\*`(?:...)` (Groupe non capturant)\*\*

- \*\*Signification\*\* :

  - \*\*Groupe\*\* une partie du motif \*\*sans la capturer\*\*.

  - Utile pour appliquer des opérations (`+`, `|`, etc.) sans générer de "sous-groupe" inutile.

- \*\*Exemple\*\* :

  - `(?:https?://)?(www\..+)` :

- `https?://` est optionnel (`?`), mais n’est pas capturé.

- Seul `www.example.com` est capturé.

**---**

### 

### Comparaison entre `(...)` et `(?:...)` :

| \*\*Aspect\*\*              | \*\*`(...)`\*\*                          | \*\*`(?:...)`\*\*                        |

|-------------------------|---------------------------------------|---------------------------------------|

| Capture le texte        | Oui (accessible avec `\1`, `group()`) | Non                               |

| Utilisation mémoire     | Oui                                   | Non                                   |

| Cas d’usage             | Extraire des parties du texte         | Grouper sans besoin de réutilisation  |

---

### 

### Exemples concrets en Python :

### 

### **Cas 1 : Extraction de date avec groupes capturants**

```python

import re

date = "25-12-2023"

match = re.search(r"(\d{2})-(\d{2})-(\d{4})", date)

if match:

print(f"Jour: {match.group(1)}, Mois: {match.group(2)}, Année: {match.group(3)}")

# Output: Jour: 25, Mois: 12, Année: 2023

```

### ---

### 

### **Cas 2 : Grouper sans capturer avec `(?:...)`**

```python

texte = "http://www.example.com et https://www.google.com"

matches = re.findall(r"(?:https?://)?(www\.[a-z]+\.com)", texte)

print(matches)  # Output: ['www.example.com', 'www.google.com']

```

- `(?:https?://)?` : Le `https://` ou `http://` est optionnel, mais non capturé.

- Seul `www....com` est extrait.

---

### 

### Cas 3 : Alternance avec `|`

```python

texte = "j'aime les chats et les chiens"

matches = re.findall(r"chat|chien", texte)

print(matches)  # Output: ['chat', 'chien']

```

---

### Pourquoi utiliser `(?:...)` ?

- \*\*Performance\*\* : Moins de mémoire utilisée si la capture est inutile.

- \*\*Clarté\*\* : Évite de surcharger les groupes capturants (utile dans les motifs complexes).

- \*\*Compatibilité\*\* : Fonctionne avec des méthodes comme `re.findall` qui retournent tous les groupes par défaut.

---

### 

### Résumé :

- `A|B` : \*\*OU logique\*\* entre motifs.

- `(...)` : \*\*Groupe + capture\*\* pour réutilisation.

- `(?:...)` : \*\*Groupe sans capture\*\* pour appliquer des opérations.