25.1-TEMPLATES JINJA
BRANCHEMENTS, ITÉRATIONS

https://www.youtube.com/watch?v=kuQ1XxXM5SU

BRANCHEMENT

- L'instruction de contrôle de flux if permet de réaliser des branchements à partir des balises {% if expression %} et {% endif %}. Ces balises permettent de traiter le contenu situé entre elles uniquement si l'expression retourne un résultat pouvant être évalué comme True.
- Comme pour de nombreux autres langages de programmation, le contrôle de flux par branchement prévoit les balises complémentaires {% else %} et {% elif expression %}.
- L'exemple suivant teste le genre de l'utilisateur pour afficher un libellé Monsieur, Madame, etc. devant le nom de l'utilisateur. L'objet user expose un attribut genre qui contient une chaîne de caractères.

EXEMPLES :

Qui produit :
le résultat « Madame Dominique » si genre contient « F » ou « f »,
le résultat « (genre x inconnu) Dominique » si genre contient « x ».

```
<h1>
 {% if user.genre.upper() == 'F' %}
     Madame
 {% elif user.genre.upper() == 'M' %}
     Monsieur
 {% elif user.genre.upper() == 'MS' %}
     Mademoiselle
 {% else %}
    (genre {{user.genre}} inconnu)
 {% endif %}
 {{ nom }}
</h1>
```

Étant donné que l'attribut genre est une chaîne de caractères Python (string), il est possible d'appeler les méthodes qu'elle expose. Par conséquent, genre.user.upper() retourne la valeur du genre en majuscule.

ITÉRATION :

- Jinja prévoit une instruction de contrôle de flux {% for x in collection %}{% endfor %} permettant une itération sur le contenu d'une collection en répétant une partie du template.
- L'exemple suivant parcourt la liste fruits (fruits=['Banane','Mangue', 'Ananas']) pour produire une liste à puce.

LES DICO :

- Il est également possible de parcourir des collections d'éléments plus complexes comme un dictionnaire ou une liste de tuple.

- Par exemple, pour un dictionnaire dico défini par le code Python :

```
dico = { "0": "zéro", "1": "un", "2": "deux", "3": "trois", "4": "quatre",
"5": "cinq", "6": "six", "7": "sept", "8": "huit", "9": "neuf" }
x
```

Il est possible d'écrire les templates suivants :

```
<h1>Les chiffres</h1><br />
<ul>
 {% for cle, valeur in dico.iteritems() %}
 <li>{{ cle }} = {{ valeur }}</li>
 {% endfor %}
</ul>
```

```
. 1 = un
. 0 = zéro
. 3 = trois
. 2 = deux
. 5 = cinq
. 4 = quatre
. 7 = sept
. 6 = six
. 9 = neuf
. 8 = huit
```

Les dictionnaires ne sont pas des éléments triés. Pour obtenir une liste triée, il faut utiliser le filtre sort dans la balise for. Ex. : {% for cle, valeur in dico.iteritems() | sort %}.

FILTRAGE DES ÉLÉMENTS :

- Tout comme cela est possible avec la List Comprehension de Python, il est possible de réduire l'ensemble des données de la boucle for en appliquant une condition de test.
- Si la condition de test est évaluée à true pour l'élément, alors l'élément passe dans l'itération sinon il est ignoré.

- Dans l'exemple précédent, la boucle for est modifiée pour filtrer les éléments supérieurs à 5.

```
<h1>Les chiffres</h1><br />
<ul>
 {% for cle, valeur in dico.iteritems() if cle|int > 5 %}
 <li>{{ cle }} = {{ valeur }}</li>
 {% endfor %}
</ul>
```

- Étant donné que le dictionnaire contient la valeur numérique sous forme de chaîne de caractères, il convient de transformer celle-ci en entier avant de la comparer à la valeur 5. En effet, la comparaison d'une chaîne de caractères > 5 est toujours vraie. La transformation vers un entier se fait à l'aide du filtre Jinja int d'où la notation cle | int.

VARIABLES SPÉCIALES :

- À l'intérieur d'une boucle for, le moteur de template Jinja met à disposition une série de variables spéciales. Ces variables sont utilisables comme n'importe quelle autre variable Jinja.
- Ces variables spéciales peuvent être utilisées pour altérer le flux de sortie en fonction de conditions spécifiques. Grâce à ces variables, il est possible d'alterner la couleur des lignes d'un tableau une ligne sur deux.

Variable	Description
`loop.index`	Numéro d'itération de la boucle (commence à 1). Soit une séquence 1, 2, 3, 4, 5...
`loop.index0`	Numéro d'itération de la boucle (commence à 0). Soit une séquence 0, 1, 2, 3, 4...
`loop.revindex`	Nombre d'itérations restant jusqu'à la fin de la boucle (correspondant à `loop.index`). Soit une séquence 5, 4, 3, 2, 1.
`loop.revindex0`	Nombre d'itérations restant jusqu'à la fin de la boucle (correspondant à `loop.index0`). Soit une séquence 4, 3, 2, 1, 0.
`loop.first`	`True` lors de la première itération.
`loop.last`	`True` lors de la dernière itération.
`loop.length`	Le nombre d'éléments dans la séquence.
`loop.cycle`	Fonction utilitaire permettant de cycliser une valeur parmi différents éléments d'une séquence. À chaque nouvelle itération de la boucle `for`, la valeur suivante est extraite de la séquence. Voir explications ci-dessous.
`loop.depth`	Indique la profondeur de récursivité de la boucle `for`. Démarre au niveau 1.
`loop.depth0`	Indique la profondeur de récursivité de la boucle `for`. Démarre au niveau 0.

- L'exemple suivant fait un rendu des lettres du mot Arc-en-Ciel (code Python lst = list('Arc-en-Ciel')) en utilisant un cycle de couleurs.

```
<strong>
{# lst obtenu avec
{% for element in lst %}
 <font color="{{ loop.cycle( "Tomato", "Orange", "DodgerBlue",
"MediumSeaGreen", "Gray", "SlateBlue", "Violet", "LightGray" ) }}">
{{ element }}</font>
{% endfor %}
</strong>
```

ALTÉRER LE COMPORTEMENT ITÉRATIF :

- Il est possible de modifier le comportement de la boucle for à l'aide des balises {% continue %} ou {% break %}.
- La balise {% continue %} permet de démarrer immédiatement la prochaine itération de la boucle {% for %}.
- La balise {% break %} permet d'interrompre immédiatement la boucle {% for %} et de poursuivre le traitement du template juste après la balise {% endfor %}.