Séance 3: Conditionnelles, boucles et chaînes

Université de Paris

Objectifs:

- S'exercer à écrire du Python.
- Utiliser des boucles.
- Effectuer des calculs arithmétiques dépendant de l'indice d'une boucle.
- Utiliser des expressions booléennes.
- Manipuler des chaînes de caractères.

Vous commencez à savoir *écrire* des programmes Python. Il faut maintenant passer à l'étape suivante : *concevoir* des programmes Python. Cela signifie que vous allez devoir *chercher les programmes* qui répondent au problème posé. Pour cela, on vous propose d'utiliser la méthodologie suivante :

1. Lisez attentivement le sujet. Si le sujet demande d'écrire une fonction, imaginez des exemples d'entrées pour cette fonction et les sorties que l'on devrait obtenir pour chacun de ces exemples.
2. Écrivez en *français*, la description la plus précise d'un programme qui répond au problème posé.
3. Traduisez en Python le programme obtenu à l'étape précédente.
4. Testez votre programme sur *suffisamment* d'exemples pour vous convaincre que votre programme est correct : il faut au moins que chaque instruction de votre programme ait été exécutée par au moins un exemple.

Rappels et remarques sur IDLE : Pour ouvrir un fichier existant, utilisez le menu `File > Open`. Le fichier sera ouvert dans une fenêtre d'édition de IDLE. Pour exécuter le programme, utilisez la touche `F5` ou le menu `Run > Run Module`. Le programme sera exécuté et les résultats affichés dans la fenêtre de l'interpréteur interactif de IDLE. Si le programme définit des fonctions ou des procédures, elles sont maintenant disponibles dans l'interpréteur interactif. Par exemple, si le programme que vous venez d'exécuter, définit une procédure `moyenne`, vous pouvez maintenant l'utiliser dans l'interpréteur ; par exemple avec `print(moyenne(1, 2, 3, 4, 5))` :

```
>>> print(moyenne(1, 2, 3, 4, 5))
15
3

```

Ceci est utile pour expérimenter rapidement. Par contre, pour répondre aux exercices finalement, il faut placer et sauvegarder tout code (ce qui inclut les tests !) dans le fichier `.py` approprié.

1 Quelques procédures et fonctions simples

Exercice 1 (Moyenne, ⋆)
Écrivez une procédure `moyenne` *qui reçoit 5 paramètres et affiche leur somme et leur moyenne entière.*

Contrat:
`moyenne(10,12,8,9,19)` *affichera :*

58
11

□

Exercice 2 (Ô temps !, ⋆)
Écrivez une procédure `secondes` *qui prend un nombre entier s de secondes en entrée et affiche le temps qu'il représente en heures, minutes, et secondes.*

> ***Contrat:***
> `secondes(3725)` *affichera :*
> *1*
> *2*
> *5*
> *car un temps de 3725 secondes correspond à 1 heure, 2 minutes, et 5 secondes.*

□

Exercice 3 (Alea jacta est., ⋆⋆)
La fonction `randint(n, m)` *du module* `random` *permet de tirer un nombre entier a pseudo-aléatoire avec $n \leq a \leq m$. On rappelle que pour utiliser le module* `random`*, on doit écrire l'instruction* `import random` *au début du programme. Ensuite, on appelle la fonction* `randint` *grâce à* `random.randint(n,m)`.

1. *Écrire une fonction* `de` *qui ne prend pas de paramètre et renvoie le résultat du lancer d'un dé cubique non pipé.*
2. *Écrire une procédure* `yams` *qui simule le lancer de 3 dés cubiques et affiche leurs résultats dans l'ordre croissant.*

□

2 Boucles dont chaque étape dépend de l'indice d'itération

Exercice 4 (Filtrer les nombres, ⋆)

1. *Définire une procédure* `count_to` *qui attend un entier* `n` *et qui pour* `i` *allant de* `0` *à* `n` *(exclu), affiche* `i`. *Évaluer* `count_to(100)`.
2. *Définir une procédure* `count_even` *qui attend un entier* `n` *et qui pour* `i` *allant de* `0` *à* `n` *(exclu), affiche* `i` *si* `i` *est pair, sinon elle afffiche la chaîne « ... ». Évaluer* `count_even(356)`.
3. *Définir une procédure* `count_mult` *qui attend deux entiers* `n` *et* `d` *en entrée et qui pour* `i` *allant de* `0` *à* `n` *(exclu), affiche* `i` *si* `i % d == 0` *et affiche la chaîne « ... » sinon. Évaluer* `count_mult(333,3)`.

□

Exercice 5 (Premier, ⋆⋆)
Ecrire une fonction `isPrime` *qui renvoie un Booléen indiquant si son argument est un nombre premier.*

> ***Contrat:***
> *Par exemple,* `isPrime(17)` *renvoie* `True`*, alors que* `isPrime(12)` *et* `isPrime(1)` *renvoient* `False`.

□

3 Conditionnelles et boucles avec chaines de caractères

Dans cette séance, vous résoudrez des exercices de difficulté et longueur variées, concernant surtout les chaînes de caractères, l'utilisation de boucles et d'expressions booléennes.

Exercice 6 (Accord, ⋆)
Écrire une fonction `withSIfNeeded` *qui prend en paramètre un nom* `name` *et un entier* `n`*, et qui renvoie* `name` *suivi d'un* `'s'` *si* `n` *est supérieur ou égal à 2,* `name` *sinon. On ne tentera pas de tenir compte des exceptions.*

> ***Contrat:***
> *Par exemple,* `withSIfNeeded('pomme', 2)` *renvoie* `'pommes'` *alors que* `withSIfNeeded('poire', 1)` *renvoie* `'poire'`*.*

□

Exercice 7 (Cadre, ⋆)

1. *Écrire une procedure* `line` *qui prend en paramètre un entier* `n` *et qui affiche une ligne de* `n` *fois le caractère* #*.*

   > ***Contrat:***
   > *Par exemple, l'appel* `line(7)` *affiche sur le terminal :*
   > `#######`

2. *Écrire une procedure* `frame` *qui prend en paramètre une chaîne de caractères, et affiche cette chaîne de caractères entourée d'un cadre de taille adaptée.*

   > ***Contrat:***
   > *Par exemple,* `frame('Hello World!')` *doit afficher :*
   > ```
   > +--------------+
   > | Hello World! |
   > +--------------+
   > ```

   On rappelle qu'on peut obtenir la taille (le nombre de caractères) d'une chaine de caractères `s` *grâce à l'expression* `len(s)`*.*

3. *(⋆⋆⋆)* ***Question bonus, à faire à la fin*** *Modifier* `frame` *pour qu'elle se comporte correctement dans le cas d'une chaîne de caractères qui contient le caractère* `'\n'`*.*

   > ***Contrat:***
   > *Ainsi,* `frame('Hello\nWorld!')` *doit afficher :*
   > ```
   > +--------+
   > | Hello  |
   > | World! |
   > +--------+
   > ```

□

Exercice 8 (Voyelles, ⋆)
Écrire une fonction `vowels` *qui renvoie le nombre de voyelles dans une chaîne de caractères.*

> ***Contrat:***
> *Par exemple,* `vowels('Hello World!')` *doit renvoyer* `3`*.*

□

Exercice 9 (Concaténation, ⋆)
Écrire une fonction `concatNTimes` *qui prend en paramètre une chaîne de caractères* `s` *et un entier* `n`*, et renvoie la chaîne de caractères* `s` *répétée* `n` *fois.*

> ***Contrat:***
> *Si* `n` *est négatif, on renverra la chaîne de caractères vide.*
> *Si* `n` *est positif,* `concatNTimes(s, n)` *doit renvoyer* `ss...ss` *où* `s` *est présente* `n` *fois.*

□

4 Utiliser des boucles pour accumuler

Exercice 10 (Somme des entiers, ⋆)
Écrire une fonction `somme` *qui renvoie la somme des entiers jusqu'à son paramètre.*

Contrat:
`somme(n)` $= 1 + 2 + 3 + \cdots + n$.

□

Exercice 11 (Nombres amicaux, ⋆⋆)
Vous avez déjà vu les nombres parfaits dans le Cours-TD 2, les nombres amicaux sont une notion proche.

1. *Écrire une fonction* `sumDiv` *qui renvoie la somme des diviseurs propres d'un entier.*

   Contrat:
   Par exemple, `sumDiv(6)` *vaut 6,* `sumDiv(1184)` *vaut 1210.*
2. *Deux entiers* `n` *et* `m` *sont dits amicaux si* `sumDiv(n) == m` *et* `sumDiv(m) == n`.
   Vérifier que 1184 et 1210 sont amicaux.
3. *Utiliser cette caractérisation pour trouver un couple de nombres amicaux inférieurs à 500. Indication : On pourra utiliser une boucle imbriquée.*

□

Exercice 12 (Fibonacci, ⋆⋆)
En 1202, le mathématicien Leonardo Fibonacci inventa l'énigme suivante : Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence ?

1. *Si l'on connaît le nombre de couples de lapins au mois* n *et au mois* $n + 1$*, comment connaître le nombre de couples de lapins au mois* $n + 2$ *?*
2. *Résoudre l'énigme.*
3. *Plus généralement, écrire une fonction* `fibonacci` *qui calcule le nombre de couples de lapins au bout d'un nombre* n *de mois donné en argument.*

□