Info113 - TD3

L'objectif du troisième TD est de se familiariser avec la notion de boucle, tout en continuant d'utiliser des structures conditionnelles. Ceci nous permettra d'écrire des programmes *Python* répétant une instruction un nombre donné de fois. Les concepts vus dans ce TD sont :

- les fonctions (encore !),
- les structures conditionnelles (`if ... elif ... else ...`),
- les boucles (`for i in range(0, n, 1) : ...`),

Boucles « for » et nombres entiers

Exercice 1. Que calcule la fonction *Python* `f` suivante ?

```
def f(n):
    r = 0
    for i in range(0, n, 1):
        r = r + 1
    return r
```

Répondez aux questions suivantes sans utiliser votre ordinateur. Quelle sont les valeurs de `f(0)`, `f(3)`, `f(42)` ? Que se passe-t-il si on appelle la fonction avec comme argument un entier négatif ?

Exercice 2. Définissez une fonction `sum` qui prend en argument un entier `n`, et retourne la somme des entiers de `1` à `n`.

Exercice 3. Définissez une fonction `sum2` qui prend en argument un entier `n`, et retourne la somme des carrés des entiers de `1` à `n`.

Exercice 4. Définissez une fonction `fact` qui prend en argument un entier `n`, et retourne le produit des entiers de `1` à `n` (c'est à dire `n!`).
Que renvoie votre fonction sur l'entier `0` ?

Exercice 5. Définissez, de deux manières différentes, une fonction `mult3` qui prend en argument un entier `n`, et qui retourne la somme des `n` premiers multiples de `3`. Par exemple, `mult3(4)` donnera `30` car $3+6+9+12=30$.

Exercice 6. Inspirez vous de votre réponse à l'exercice précedent pour définir une fonction `multk`, prenant en argument deux entier `n` et `k`, et calculant la somme des `n` premiers multiples de `k`.

Exercice 7. Définissez une fonction `est_premier` qui prend en argument un entier `n`, et qui retourne `True` si `n` est premier, et `False` sinon.
Rappel : un nombre est premier s'il n'est multiple d'aucun nombre, à part 1 et lui même.

Exercice 8. Définissez une fonction `sum_primes` qui prend en argument un entier `n`, et qui retourne la somme de tous les nombres premiers entre `1` et `n`.

Boucles « for » et chaînes de caractères

Exercice 9. Définissez une fonction `nb_espace` qui prend en argument une chaîne de caractères `s`, et retourne le nombre d'espaces contenu dans `s`. Définissez ensuite une fonction `nb_char` qui prend en argument un caractère `c` et une chaîne de caractère `s`, et qui retourne le nombre d'occurences de `c` dans `s`.

Exercice 10. Utilisez la fonction `nb_char` définie dans l'exercice précédent pour écrire une fonction `apparait_dans` qui prend en argument un caractère `c` et une chaîne `s`, et qui retourne `True` si `c` apparaît dans `s` et `False` sinon.

Exercice 11. Définissez une fonction `est_voyelle` prennant en argument un caractère `c` et retournant `True` si `c` est une voyelle et `False` sinon. Faites de même avec des fonction `est_consonne`, `est_minuscule`, `est_majuscule`, `est_ponctuation` et `est_chiffre`.
Indice : la fonction `apparait_dans` définie à l'exercice précédent pourrait être utile.

Exercice 12. Définissez une fonction `nb_voyelles` qui compte le nombre de voyelles dans une chaîne de caractères. Définissez ensuite une fonction `nb_consonnes` qui compte le nombre de consonnes.

Exercice 13. Définissez une fonction `sup_espace` qui prend en argument une chaîne de caractères, et retourne une copie de cette chaîne, à laquelle on a retiré tous les espaces. Définissez ensuite des fonctions `sup_voyelles` et `sup_consonne`.

Exercice 14. Définissez de deux manières différentes une fonction `renverse` qui prend en argument une chaîne de caractères et retourne un copie de cette chaîne qui a été renversée, c'est à dire que l'ordre de ses caractères a été inversé. Par exemple :

```
>>> renverse("salut")
"tulas"
```

Exercice 15. Définissez une fonction `est_un_palindrome` qui prend en argument une chaîne de caractères, et retourne `True` si la chaîne est un palindrome, et `False` sinon. *Exemples de palindromes :* SOS, RADAR, ROTOR, KAYAK, ...

Exercice 16. Définissez une fonction `chaine_mul` qui prend en argument une chaîne de caractères `s`, et un entier `n`, et qui multiplie la chaîne `s` par `n` de la même manière que l'opérateur `*`, à l'exception prêt qui si `n` est négatif, la chaîne `s` est multipliée en sens inverse.

```
>>> chaine_mul("abc", -2)
"cbacba"
```

Exercice 17. Définissez une fonction `coupe_chaine` qui prend en argument une chaîne de caractères `s` et un entier `n` et qui retrourne une chaîne contenant les `n` premiers caractères de `s`. Si `s` contient moins de `n` caractères, la chaîne entière est retournée.

```
>>> coupe_chaine("abcd", 3)
"abc"
>>> coupe_chaine("hello", 0)
""
>>> coupe_chaine("blabla", 10)
"blabla"
```

Exercice 18. Définissez une fonction `sous_chaine` qui prend en argument une chaîne de caractères `s` et deux entiers `p1` et `p2` et qui rend la sous-chaîne de `s` commencant à la position `p1` et terminant à la position `p2`. Le comportement à adopter si `p1` est plus grand que `p2`, où si `p1` et/ou `p2` ne sont pas des indices valides est décrit par les exemples ci-dessous :

```
>>> sous_chaine("blabla", 0, 2)
"bla"
>>> sous_chaine("abcd", 2, 10)
"cd"
>>> sous_chaine("blabla", 2, 0)
""
>>> sous_chaine("aa", 2, 4)
""
```

Exercice 19. Définissez une fonction `chaine_mul_float` qui prend en argument une chaîne de caractère `s`, et un nombre flottant positif `f`. Cette fonction devra se comporter d'une manière similaire à l'opérateur `*` sur les chaînes de caractères, à l'exception prêt que si `f` n'est pas une valeur entière, la dernière copie de `s` sera coupée. Par exemple :

```
>>> chaine_mul_float("abcd", 2.5)
"abcdabcdab"
```

Note : vous aurez besoin de la fonction `int`, qui permet de tronquer un flottant au niveau de la virgule pour obtenir un entier.

Suites et sommes complexes

Exercice 20. Définissez une fonction `approx_e` qui prend en argument un nombre `n` et calcule la somme $\sum_{i=0}^{n} \frac{1}{i!}$.

Exercice 21. Définissez une foncton `approx_pi` qui prend en argument un nombre `n` et calcule la somme $4 \times \sum_{i=0}^{n} \frac{(-1)^i}{2i+1}$.

Exercice 22. On considèle la fonction *Python* `f` suivante.

```
def f(n):
    a = 1
    b = 2
    for i in range(1,n,1):
        b = a+b
        a = b-a
    return a
```

Quelle est la valeur de `f(k)` pour `k` entre `0` et `5` ? Pouvez-vous en déduire ce que calcule cette fonction ?

Exercice 23. Exprimez en termes mathématiques, c'est à dire avec une somme, la fonction *Python* `g` suivante.

```
def g(n, x):
    r = 0
    for i in range (0, n+1, 1) :
        a = (-1) ** n
        b = x ** (2 * n)
        c = fact(2 * n)
        r = r + (a * b) / c
    return r
```