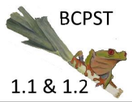

TP d'informatique n° 2

— Tests et boucles —

Pour écrire un algorithme en langage Python, il faut identifier les **tâches élémentaires**, c'est-à-dire les tâches simples, qui constitueront cet algorithme.
Par exemple, ajouter un à une variable, tester la parité d'un entier, ...
Chaque tâche élémentaire va correspondre à un bloc de commandes.

La structuration du code repose sur **l'indentation** , c'est-à-dire sur le décalage plus ou moins marqué du début de la ligne.
Par défaut ce décalage est de 4 espaces (qui correspondent à une tabulation dans `Pyzo`). Il se produit automatiquement lorsqu'une ligne se termine par le caractère : (deux points) qui signifie qu'un bloc d'instructions commence.

Objectifs du TP :

- Savoir écrire un test
- Savoir écrire une boucle while
  - Choisir la condition
  - Incrémenter le variant de boucle.
- Savoir écrire une boucle for.
  - Bien choisir les valeurs du variant de boucle.
  - Savoir calculer une somme (comme dans l'exercice 10).
  - Savoir calculer les termes d'une suite récurrente (comme dans l'exercice 12).

1 Le branchement conditionnel

1.1 Sans alternative

Si on veut effectuer une tâche seulement lorsqu'une condition est remplie, on utilise **un branchement conditionnel sans alternative** :

```
if condition:
    bloc de commandes
    exécutées si la condition est vraie (= True)
```

La condition est un test dont la réponse est booléen (`True` ou `False`). Les tests peuvent porter sur tous les types de variables. Voici quelques exemples :

```
x == y    # x est-il égal a y ?
x != y    # x est-il différent de y ?
x > y     # x est-il plus grand que y ?
x < y     # x est-il plus petit que y ?
x >= y    # x est-il supérieur ou égal a y ?
x <= y    # x est-il inférieur ou égal a y ?
x < y < z # y est-il encadré par x et z (sens strict) ?
(1<3) and (4>2) # la réponse est True
(5>8) or not (2>4) # la réponse est True
```

1.2 Avec alternative

Si on veut effectuer une tâche lorsqu'une condition est remplie et une autre tâche lorsque la condition n'est pas remplie, on utilise **un branchement conditionnel avec alternative** :

```
if condition:
    bloc de commandes
    exécutées si la condition est vraie
else:
    bloc de commandes
    exécutées si la condition est fausse
# remarquer la suppression de l'indentation devant else qui indique la fin du bloc du if
```

On remarque qu'il n'y a pas de condition après le `else` puisque le test a déjà été fait au niveau du `if`.
On retient donc qu'il faut sortir du bloc du `if` et taper simplement " `else` : ".

Exercice 1. Majorité
On s'intéresse à la fonction suivante.

```
def majeur_mineur(age):
    if age >= 18:
        rep = 'majeur'
    else:
        rep = 'mineur'
    return rep
```

1. Quelles sont les variables d'entrée et de sortie ?
2. De quel type sont-elles ?

Exercice 2. Valeur absolue
Écrire une fonction `valeur_absolue` qui reçoit en entrée un réel x et renvoie sa valeur absolue (sans utiliser la commande `abs`).

Exercice 3. Maximum de deux réels
Écrire une fonction `max2` qui calcule le maximum de deux nombres `a` et `b` qu'elle reçoit en entrée.

Exercice 4. Maximum de trois réels
En utilisant la fonction `max2`, écrire une fonction `max3` qui calcule le maximum de trois nombres `a`, `b` et `c`.

Exercice 5. Médiane de trois nombres
Écrire une fonction `mediane` qui, lorsqu'on lui donne trois valeurs `a`, `b`, `c`, renvoie le nombre qui est compris entre les deux autres (on suppose que les trois nombres sont distincts deux à deux).

2 Les boucles « tant que » et « pour »

2.1 La boucle « tant que »

On peut vouloir effectuer une tâche tant que la condition est remplie. Ceci correspond à une succession de tests `if` mais sans savoir combien de tests seront nécessaires.
On utilise alors une boucle **while**.

Exercice 6. Puissance

On s'intéresse à la fonction suivante.

```
def puissance_de_3(n):
    p = 0
    while 3**p < n:
        p = p + 1
    return 3**p
```

1. Quelles sont les variables d'entrée et de sortie et quel est leur type ?
2. Que renvoie la fonction si on l'exécute pour `n=10` ?
3. A quoi sert la ligne `p=p+1` ?

La structure générale est la suivante :

```
while condition:
    bloc de commandes
    exécutées tant que la condition est vraie
```

Tant que la condition est remplie (tant que le test renvoie `True`), on effectue les commandes du bloc. Dés que la commande est fausse, on sort de la boucle et on reprend la suite des instructions qui suivent le bloc du while.

Il faut être sûr que la condition sera fausse au bout d'un moment (même un temps long). Sinon, la commande va être effectuée encore et encore indéfiniment. Par exemple :

```
i=1
while i < 2 :
    i=i-1
```

Il est alors nécessaire d'interrompre manuellement la boucle infinie. Pour cela il suffit de cliquer sur la touche représentant un éclair jaune .

Exercice 7. Partie entière

Écrire une fonction `partie_entiere` qui prend en entrée un réel x strictement positif et qui renvoie le plus grand entier inférieur à x.

2.2 La boucle « pour »

Si on sait combien de fois on va réaliser la tâche, on utilise plutôt une **boucle for**.

Exercice 8. Découverte du range

On s'intéresse au programme suivant.

```
x=1
for i in range(5):
    x=x+1
    print('i=', i, 'et x = ',x)
print('la boucle est finie')
```

1. Que représente la commande `range(5)` ?
2. Que va-t-on obtenir dans le shell si on exécute ce programme ?

Pour signifier les bornes entre lesquelles le variant de boucle va varier, on utilise la commande `range`.

- Si on écrit `range(n)`, où n est un entier naturel, alors le variant prendra les valeurs de 0 à $n-1$. Cela fait n valeurs différentes.
- Si on écrit `range(a,b)`, où a et b sont deux entiers naturels, alors le variant prendra les valeurs de a à $b-1$. Cela fait $b-a$ valeurs différentes.

```
for i in range(a,b): # attention la valeur b n'est pas atteinte
    bloc de commandes
    exécutés b-a fois
    (avec i qui varie automatiquement de a à b-1)
```

Exercice 9. Somme

On s'intéresse à la fonction suivante :

```
def somme(n):
    s = 0
    for i in range(n):
        s = s + i
    return s
```

Que renvoie la fonction si on tape dans le shell `somme(4)` ?

Exercice 10. Sommation

1. Écrire une fonction `somme2` qui prend en entrée deux entiers `n` et `p` et qui calcule la somme des entiers compris (au sens large) entre `n` et `p`.
2. Tester la fonction avec `somme2(3,8)` qui doit renvoyer 33.
3. Écrire une fonction `somme_carres` qui prend en entrée deux entiers `n` et `p` et qui calcule la somme des carrés des entiers compris (au sens large) entre `n` et `p`.
4. Tester la fonction avec `somme_carres(3,8)` qui doit renvoyer 199.

Exercice 11. Factorielle

1. Écrire une fonction `factorielle` qui prend en entrée un entier `n` et qui calcule `n!` (sans utiliser la fonction `factorial`).
2. Tester la fonction avec `factorielle(5)` qui doit renvoyer 120.

2.3 Application à l'étude des suites récurrentes

Exercice 12. Suite définie par récurrence

On s'intéresse à la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et $\forall n \in \mathbb{N}$, $u_{n+1} = 2u_n + 3$.

1. Écrire une fonction `suite1` qui prend en entrée `n` et renvoie la valeur de u_n.
2. Vérifier que le résultat obtenu pour `n=4` est 61.
3. La suite $(u_n)_{n\in\mathbb{N}^*}$ est maintenant définie par $u_1 = 1$ et $\forall n \in \mathbb{N}^*$, $u_{n+1} = 2u_n + 3$.
   Écrire la fonction `suite2(n)` qui renvoie la valeur de u_n.

Exercice 13. Récurrence double

On s'intéresse à la suite $(u_n)_{n\geq 5}$ définie par $u_5 = 1$, $u_6 = 2$ et $\forall n \geq 5$, $u_{n+2} = u_n * u_{n+1}$.

1. Ecrire une fonction `suite3` qui prend en entrée `n` et renvoie la valeur de u_n.
2. Vérifier le résultat obtenu pour $n = 10$.

Exercice 14. Suite convergente

On s'intéresse à la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par $u_0 = 2$ et $\forall n \in \mathbb{N}$, $u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right)$.

1. Ecrire une fonction `appro` qui prend en entrée une paramètre ε et qui renvoie le premier entier `n` tel que $|u_{n+1} - u_n| < \varepsilon$.
2. Déterminer une valeur approchée de la limite de cette suite à 10^{-3} près puis à 10^{-10} près.

2.4 Comparaison entre les deux boucles

	Boucle `while`	Boucle `for`
Initialisation	à faire avant la boucle	automatique
Incrémentation	à faire dans la boucle	automatique
Arrêt	bien choisir la condition	automatique

3 Pour aller plus loin

Exercice 15. Algorithme de Syracuse :
Voici les étapes de l'algorithme proposé par Syracuse :

- choisir un entier positif non nul (la **graine**),
- s'il est pair on le divise par 2, sinon on le multiplie par 3 et on ajoute 1,
- répéter le processus.

On constate que quelle que soit la graine choisie, la suite des valeurs passe par la valeur 1 (puis 4, 2, 1, 4, 2, 1, …). Personne ne sait démontrer que cette affirmation est vraie !
On appelle **longueur de la chaine** le nombre d'étapes nécessaires pour arriver à la première valeur 1.

1. Compléter, à la main, la suite des valeurs pour la graine 7 : 7, 22, 11, 34, 17, 52…
2. Quelle est la longueur de la chaine pour la graine 7 ?
3. Écrire une fonction qui prend en entrée la graine, qui affiche la suite des nombres jusqu'à 1 et renvoie la longueur de la chaine. On pourra utiliser la commande `print` pour afficher la suite des nombres.