TD3

February 17, 2020

1 Les boucles

boucle : *répéter un bloc d'opération autant de fois que nécessaire.*

Imaginons que l'on veuille afficher les valeurs de x^2 pour $x = 1, 2, 3...10$. On pourrait écrire :

```
In [ ]: print("1^2 = ",1*1)
        print("2^2 = ",2*2)
        print("3^2 = ",3*3)
        print("4^2 = ",4*4)
        print("5^2 = ",5*5)
        print("6^2 = ",6*6)
        print("7^2 = ",7*7)
        print("8^2 = ",8*8)
        print("9^2 = ",9*9)
        print("10^2 = ",10*10)
```

Cela fait beaucoup de lignes de codes donc beaucoup de risque d'erreur. Et si je veux maintenant aller jusqu'à 1000 ou afficher x^3 ou X^4, ce n'est pas très adapté.

Si on regarde bien, on réalise en fait plusieurs fois la même chose en modifiant des paramètres. Dans ce cas, nous allons pouvoir utiliser les boucles.

Il y a deux types de boucle : les boucles `while` et les boucles `for`.

1.1 La boucle `while`

*On répète un bloc **tant que** (`while`) une condition est respectée.*

Cela s'écrit de la façon suivante :

```
while condition :
    instruction 1
    instruction 2
    instruction 3...
```

- Il est nécessaire d'avoir une condition qui finit par ne pas être satisfaite. Sinon la boucle ne s'arrête jamais.
- Les instructions doivent donc agir sur la condition.

- Les conditions peuvent être multiples.
- Il est possible de répéter le bloc sans savoir à l'avance combien de fois on va le répéter.

Essayons de faire un simple compteur de 0 à 10 :

```
In [5]: nb = 0    # initialisation du compteur

        while nb < 10 :    # condition : tant que nb est inférieur à 10
            print(nb)      # on affiche la valeur de nb
            nb = nb + 1    # on incrémente le compteur, et donc on modifie la condition qui
                           # finira par ne plus être satisfaite
```

```
0
1
2
3
4
5
6
7
8
9
```

Cette boucle offre de nombreuses possibilités : - Si je veux compter jusqu'à 100, on modifie simplement la condition `nb < 100` - Si je veux compter de 2 en 2, on modifie l'avancé du compteur `nb = nb + 2` - Si je veux faire un compte à rebours, on initialise `nb = 10`, on modifie la condition `nb > 0` et l'incrémentation `nb = nb -1`

1.1.1 Exercice 1 :

1) Reprener l'exemple du début (copié ci-dessous) et remplacer ces 10 lignes par une boucle `while` équivalente.

```
In [ ]: print("1^2 = ",1*1)
        print("2^2 = ",2*2)
        print("3^2 = ",3*3)
        print("4^2 = ",4*4)
        print("5^2 = ",5*5)
        print("6^2 = ",6*6)
        print("7^2 = ",7*7)
        print("8^2 = ",8*8)
        print("9^2 = ",9*9)
        print("10^2 = ",10*10)
```

2) Ajouter la possibilité de choisir facilement le nombre de valeurs à afficher (de 1 à 100, de 1 à 1000...).

3) Adapter ensuite ce code pour que l'on puisse facilement changer l'exposant.

1.1.2 Exemples de boucle `while` qui ne connait pas à l'avance le nombre de fois que la boucle va être exécutée

Dans la boucle suivante, l'utilisateur va devoir entrer un nombre. Tant que ce nombre est négatif, l'ordinateur redemande à l'utilisateur de rentrer un nombre. A vous d'essayer.

```
In [6]: valeur = -1
        while valeur < 0 :
            valeur = float(input("Entrez un nombre positif : "))
        print("Merci")

Entrez un nombre positif : -1
Entrez un nombre positif : -3
Entrez un nombre positif : 2
Merci
```

Dans l'exemple suivant, nous allons faire une boucle totalement infini. Pour sortir de la boucle, il est alors possible d'utiliser la commande `break`.

```
In [2]: while True :
            valeur = input("Pour quitter, entrez Q :")
            if (valeur == "Q") :
                break

Pour quitter, entrez Q :1
Pour quitter, entrez Q :2
Pour quitter, entrez Q :3
Pour quitter, entrez Q :Q
```

1.2 La boucle `for`

On répète un bloc d'instruction **en parcourant une liste**.

Cela s'écrit de la façon suivante :

```
for element in liste :
    instruction 1
    instruction 2
    instruction 3....
```

Dans cette boucle `for`, la variable `element` va prendre tour à tour toutes les valeurs de la variable `liste`. Et à chaque fois, les instructions vont être exécutées.

Mais attendez, nous n'avons pas encore vu ce qu'était une liste !!!!

Pour commencer, en voici trois types. Mais nous en rencontrerons de nouveaux au fur et à mesure du cours.

1.2.1 Les listes - Simples ([]):

Pour fabriquer une liste en python, c'est très simple. Il suffit de mettre les éléments de la liste entre crochet `[]` et de séparer les éléments avec une virgure `,`. Voici un exemple :

```
liste = [1,"lundi",0.45]
```

- Il est possible de mélanger les types dans une liste, mais attention à ce que vous ferez après. Ici nous avons un `int`, un `str` entre "et un `float`.

L'exemple suivant présente l'utilisation de la boucle `for` avec la liste précédente :

```
In [7]: liste = [1,"lundi",0.45]

        for element in liste :
            print("La variable element vaut : ", element)
```

```
La variable element vaut :  1
La variable element vaut :  lundi
La variable element vaut :  0.45
```

Dans cet exemple, la variable `element` prend tour à tour la valeur `1` puis `"lundi"` puis `0.45`. A chaque fois, l'ensemble des instructions est exécuté.

1.2.2 Les liste - Chaînes de caractères (`str`) :

Les chaînes de caractères (`str`) sont des listes. Grâce à la boucle `for` nous pouvons parcourir tous les caractères qui composent une chaîne de caractères. Voyons un exemple, ce sera plus parlant :

```
In [8]: phrase = """Il fait 30řC."""

        for lettre in phrase : # grace à cette ligne, on parcourt toutes les lettres de la phras
            print("La variable lettre vaut : ", lettre)
```

```
La variable lettre vaut :  I
La variable lettre vaut :  l
La variable lettre vaut :
La variable lettre vaut :  f
La variable lettre vaut :  a
La variable lettre vaut :  i
La variable lettre vaut :  t
La variable lettre vaut :
La variable lettre vaut :  3
La variable lettre vaut :  0
La variable lettre vaut :  ř
La variable lettre vaut :  C
La variable lettre vaut :  .
```

Ici la variable `lettre` parcourt les caractères (espaces compris) de la variable `phrase`.

1.2.3 Les listes - Suites d'entier (fonction `range()`)

Une liste bien pratique est une suite d'entier. Par exemple : $0, 1, 2, 3, 4...$ On peut le faire de façon manuelle :

```
suite = [0,1,2,3,4,5,6,7,8,9]
```

La boucle `for`s'écrit alors :

```
In [9]: suite = [0,1,2,3,4,5,6,7,8,9]

        for entier in suite :
            print("La variable entier vaut : ", entier)
```

```
La variable entier vaut :  0
La variable entier vaut :  1
La variable entier vaut :  2
La variable entier vaut :  3
La variable entier vaut :  4
La variable entier vaut :  5
La variable entier vaut :  6
La variable entier vaut :  7
La variable entier vaut :  8
La variable entier vaut :  9
```

Mais si la liste devient très grande, ce n'est pas bien pratique. On utilise donc la fonction `range()`. Cela fonctionne de la façon suivante :

```
range(debut,fin)
```

Une suite d'entier de 0 à 10 s'écrit ainsi : `range(0,10)`.
La boucle `for` s'écrit alors :

```
In [10]: suite = range(0,10)

         for entier in suite :
             print("La variable entier vaut : ", entier)
```

```
La variable entier vaut :  0
La variable entier vaut :  1
La variable entier vaut :  2
La variable entier vaut :  3
La variable entier vaut :  4
La variable entier vaut :  5
La variable entier vaut :  6
La variable entier vaut :  7
La variable entier vaut :  8
La variable entier vaut :  9
```

Il est également possible d'utiliser directement la fonction `range()` dans l'appel de la boucle `for`:

```
In [11]: for entier in range(0,10) :
             print("La variable entier vaut : ", entier)
```

```
La variable entier vaut :  0
La variable entier vaut :  1
La variable entier vaut :  2
La variable entier vaut :  3
La variable entier vaut :  4
La variable entier vaut :  5
La variable entier vaut :  6
La variable entier vaut :  7
La variable entier vaut :  8
La variable entier vaut :  9
```

1.2.4 Exercice 2 : Reprenons l'exemple initial :

1) Reprener l'exemple du début et remplacer ces 10 lignes par une boucle `for` équivalente.

```
In [ ]: print("1^2 = ",1*1)
        print("2^2 = ",2*2)
        print("3^2 = ",3*3)
        print("4^2 = ",4*4)
        print("5^2 = ",5*5)
        print("6^2 = ",6*6)
        print("7^2 = ",7*7)
        print("8^2 = ",8*8)
        print("9^2 = ",9*9)
        print("10^2 = ",10*10)
```

Vous savez tout ce dont nous aurons besoin sur les boucles `for`et `while`. N'oubliez pas que dans la boucle vous pouvez utiliser toutes les instructions que vous voulez. Par exemple, on peut mettre un `if` dans une boucle ou imbriquer plusieurs boucles. A vous de jouer.

1.2.5 Exercice 3 : Notes de contrôle

Lors d'un semestre, un lycéen a eu les notes suivantes : 10.5; 12.5; 19; 4.5; 10.5; 15; 8; 6.5; 14; 17; 13; 8.5; 12; 15, 5; 2; 7; 10; 15.5; 20; 19, 5; 1.5

1) Ecrire un code qui calcule la moyenne de ces notes.

```
In [ ]:
```

2) Modifiez ce code pour qu'il recherche en même temps la meilleure note que le lycéen a eu.

3) Modifiez ce code pour qu'il compte en même temps combien de notes sont au dessus de la moyenne.

1.2.6 Exercice 4 : Suite de Fibonacci

La suite de Fibonacci peut être considérée comme le tout premier modèle mathématique en dynamique des populations ! Elle y décrit la croissance d'une population de lapins sous des hypothèses très simplifiées, à savoir : chaque couple de lapins, dès son troisième mois d'existence, engendre chaque mois un nouveau couple de lapins, et ce indéfiniment.

Mathématiquement, la suite F_n s'écrit comme cela :

$ F_0 = 0< /div >< divalign = "center" > F_1 = 1< /div >< divalign = "center" > F_n = F_{n-1} + F_{n-2}$

Ecrire un code pour déterminer combien de mois (i.e. la valeur n) pour avoir plus de 100 lapins.

In [ ]:

1.2.7 Exercice 5 : Nombres premiers

Nombre premier : *nombre qui ne peut être divisé que par lui-même et par* 1.

Ecrire un programme qui affiche les nombres premiers inférieurs à 1000.

In [ ]: