01-structure-programme

April 1, 2019

1 La structure d'un programme

(code sous licence creative commun CC BY-NC-SA BY Alexis Dendiével) ## structure général d'un programme

elle correspond pour un programme simple au déroulé suivant

- importation des bibliothèques
- définition des constantes, des fonctions
- appel des entrées
- corps de programme
- affichage des résultats.

1.1 l'importance du commentaire

Les commentaires dans un programme sont d'une importance capitale. ils servent: - à l'auteur du programme - au lecteur du programme - à celui qui souhaite modifier le programme.

Sans aucun impact sur l'execusion, ils gagnent à être le plus précis possible.

```
In [1]: # ceci est un commentaire de programme
```

Tout ce qui est précédé du signe # dans un programme est un commentaire. Voyons ceci sur un exemple:

le programme suivant, calculant la force de gravitation entre deux masses m1 et m2 séparées d'une distance d, est donnée en version brute et en version commentée:

```
In [2]: G = 6.67e-11
        print("nous nous nous proposons de calculer la force de gravitation "
              "s'exerçant entre deux masses m1 et m2 séparées d'une distance d")
        m1 = float(input('Entrer la valeur de la masse m1 (en kg): '))
        m2 = float(input('Entrer la valeur de la masse m2 (en kg): '))
        d = float(input('Entrer la distance (en mètre) séparant m1 et m2 '))
        FG = G*m1*m2/d**2
        print('{:50}'.format("la valeur de la force de gravitation est: "),
              "{0:.2e}".format(FG), ' N')
```

```
nous nous nous proposons de calculer la force de gravitation s'exerçant entre deux masses m1 et
Entrer la valeur de la masse m1 (en kg): 80
Entrer la valeur de la masse m2 (en kg): 65
```

```
Entrer la distance (en mètre) séparant m1 et m2 2
la valeur de la force de gravitation est:          8.67e-08  N
```

```
In [3]: # programme calculant la force de gravitation entre deux masses m1 et m2 séparées par

        # la constantes utilisée
        G = 6.67e-11

        # présentation du programme
        print("nous nous nous proposons de calculer la force de gravitation "
              "s'exerçant entre deux masses m1 et m2 séparées d'une distance d")

        # entrée des données
        m1 = float(input('Entrer la valeur de la masse m1 (en kg): '))
        m2 = float(input('Entrer la valeur de la masse m2 (en kg): '))
        d = float(input('Entrer la distance (en mètre) séparant m1 et m2 '))

        # calcul de la force
        FG = G*m1*m2/d**2

        # impression du résultat avec trois chiffres dsignificatifs.
        print('{:50}'.format("la valeur de la force de gravitation est: "),
              "{0:.2e}".format(FG), ' N')
```

```
nous nous nous proposons de calculer la force de gravitation s'exerçant entre deux masses m1 et
Entrer la valeur de la masse m1 (en kg): 80
Entrer la valeur de la masse m2 (en kg): 65
Entrer la distance (en mètre) séparant m1 et m2 2
la valeur de la force de gravitation est:          8.67e-08  N
```

Nous voyons dans cet exemple simple que la lecture du programme est beaucoup plus aisée avec les commentaires. De plus, le saut d'une ligne n'a aucun impact sur la programmation, et permet une plus grande lisibilité.

Il est donc fortement recommandée de prendre le temps de commenter ses programmes.

(Il en sera d'ailleurs de même pour les fonctions)

1.2 les noms de variable

Rien ni personne n'oblige à utiliser tel ou tel nom de variable dans python. Ce langage étant orienté objet, la variable n'a pas à être défini au début du programme, elle pointe sur un objet et en présente les caractéristiques.

Il est cependant fortement recommandé d'utiliser des noms de variables donnant sens au programme, pour plus de clarté envers les personnes susceptibles de le lire ou de le modifier. exemples: - "atome" plutôt que "a" - "forcedegravitation" plutôt que "f" - "temps" plutôt que "t" ...

1.3 l'importance de l'indentation

Un programme python n'a pas de code pour son début et sa fin. Sa structuration suit une indentation qui définit la structuration du programme.

le ":" après les instructions tabule automatiquement la ligne suivante. Pour une instruction else suivant un if par exemple, il faut revenir en arrière afin que la structure soit correcte et lisible.

```
In [4]: n = 3
        if n>5:
            print("n est plus grand que 5")
        else:
            print("n est plus petit ou égal à 5")
```

```
n est plus petit ou égal à 5
```

Cette règle d'écriture structure le programme et demande de la rigueur. Il est par ailleurs fortement conseillé de ne pas faire des lignes de code trop grandes. Si celle-ci est importante, un passage à la ligne en milieu d'instruction donnera davantage de lisibilité au programme.

1.4 quelques bibliothèques en python

Même si la bibliothèque standard python est bien développée, certaines bibliothèques sont bien utiles: ### numpy C'est une bibliothèque très utile pour les calculs numériques.

```
In [5]: # importation de la bibliothèque numpy
        import numpy as np
```

La bibliothèque de calcul, dont vous trouverez facilement les utilisations sur le net, est ici importée en début de programme sous le nom np.

```
In [6]: # quelques exemples de calcul
        print(np.pi)
        print (np.sin(np.pi/4))
        print (np.cos(np.pi/4))
```

```
3.141592653589793
0.7071067811865475
0.7071067811865476
```

1.4.1 matplotlib

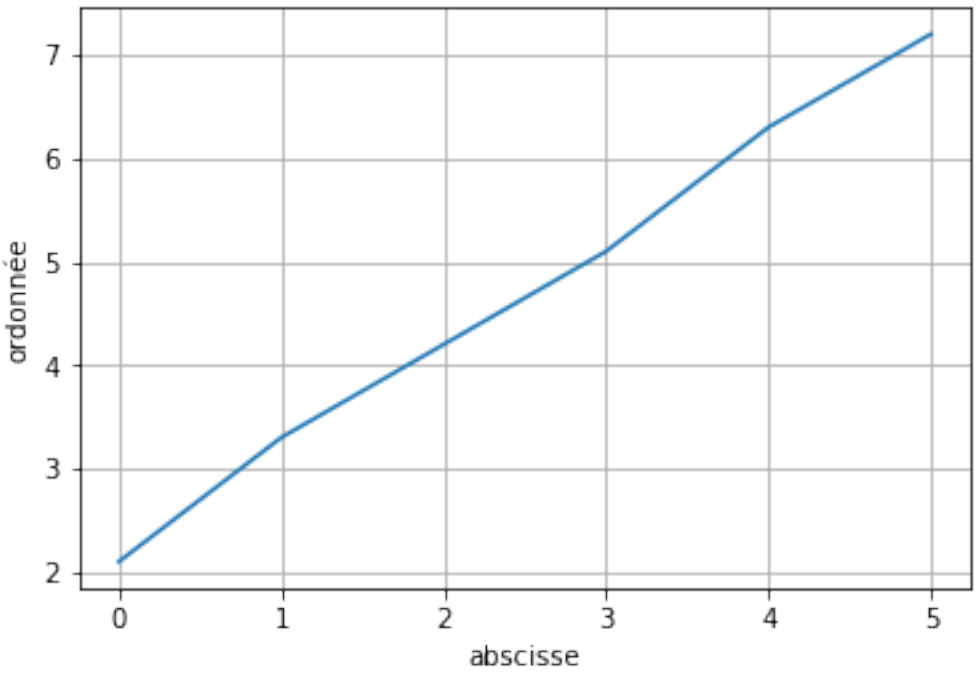
C'est une bibliothèque permettant de faire facilement des graphiques.

```
In [8]: # programme de tracé simple avec des valeurs d'abscisse et d'ordonnée

        # importation de la bibliothèque matplotlib
        import matplotlib.pyplot as plt
```

```
# initialisation des données
x = (0,1,2,3,4,5)
y = (2.1, 3.3, 4.2, 5.1, 6.3, 7.2)

# affichage
plt.plot(x, y)
plt.grid()
plt.xlabel("abscisse")
plt.ylabel("ordonnée")
plt.show()
plt.close()
```



1.4.2 math

C'est une bibliothèque qui permet de faire facilement les calculs mathématiques

```
In [9]: # importation de la bibliothèque math
        import math as math

        # quelques exemples
        print (math.cos(math.pi))
        print (math.sqrt(5))
        print ("la valeur approchée du nombre d'or est", (1+math.sqrt(5))/2)
```

```
-1.0
2.23606797749979
```

```
la valeur approchée du nombre d'or est 1.618033988749895
```

1.4.3 random

C'est une bibliothèque permettant de créer des nombres aléatoires

```
In [11]: # importation de la bibliothèque random
         import random as rd

         # impression de 5 nombres aléatoires compris entre 1 et 100
         for i in range (5):
             print(rd.randint(1,100))
```

```
63
70
64
91
78
```

1.4.4 turtle

Cette bibliothèque permet de tracer des graphiques en suivant la "tortue. (Cette bibliothèque & cet exemple ne fonctionnent pas sur un environnement jupyter en ligne)

```
In [ ]: # importation de la bibliothèque
        import turtle as tu

        tu.reset()
        tu.speed(1)
        for i in range(6) :
          tu.fd(100)
          tu.rt(360/6)
```

liste non exaustive des fonctions turtle

- fd(n) avance de n
- bk(n) recule de n
- rt(n) tourne à droite de n degrés
- lt(n) tourne à gauche de n degrés
- clear() efface l'écran
- penup() lève le stylo
- pendown() baisse le stylo
- reset() efface l'écran, remet la tortue au centre et réinitialise ses paramètres
- showturtle() montre la tortue
- hideturtle() cache la tortue
- speed(n) Change la vitesse de 1(lent) à 10 (rapide). La valeur spéciale 0 est la plus rapide.

- tracer(n,d)

- update() Force l'affichage des graphismes en attente
- bye() Referme la fenêtre
- setup(w,h) Ouvre une fenêtre de taille wxh

1.5 conclusion

Ce notebook donne les premiers conseils afin de structurer un programme python. Ce qu'il faut retenir, c'est la recherche de la clarté tant dans le code que dans les commentaires.