Projets Python

June 11, 2015

1 Projets Python

Dans ce cours autour de la programmation Python et de l'algorithmie, nous verons beaucoup de petits morceaux de programmes.

Il est vivement recommander de ne pas faire **que** les questions poser mais aussi d'**expérimenter** par soi même.

- Introduction à Python
- Générateur d'excuses
- Magic 8 ball
- Deviner un nombre
- Pierre feuille ciseau
- Deviner un nombre (améliorations)
- Generateur de Smiley
- Chiffrement de Cesar

• 1.1 Polynome du 2nd degré

Notions de programmation

- input
- Les conditions if
- Les boucles for
- Les boucles while

• 1.2 Les fonctions

Ressouces Python

- site du zero (cours en ligne)
- codeacademy (cours en ligne)
- code conbat (jeu en ligne qui se joue en codant)
- checkio (jeu en ligne - anglais - avec beaucoup de langage différents)

1.3 Introduction Python

Dans les encadrés, on peut lancer des commandes Python et écrire des petits programmes.

La touche **F5** ou l'icône en triangle vert permet d'executer tout le programme écrit dans la partie de droite.

```
In [59]: print "Bonjour"
```

```
Bonjour
```

- Modifier la commande précédente pour qu'elle n'affiche plus bonjour mais votre nom puis executer cette commande.

Il est possible d'afficher plusieurs messages:

```
In [2]: print "bonjour"
        print 'monde'
```

```
bonjour
monde
```

- Modifier le programme précédent pour qu'il affiche:

  Bonjour

  tous

  le

  monde

Quand une ligne commence par un #, Python ne la lit pas. On appelle cela des commentaires. Nous les utiliserons pour expliquer ce que l'on fait.

```
In [3]: # la commande suivante affiche Bonjour
        print('bonjour')
        # Quand on veut afficher quelque chose, on utilisera la commande print()
```

```
bonjour
```

1.4 Les variables

Les variables sont des *enveloppes* où l'on stockera des nombres, des mots, des phrases ou des choses plus compliqués.

```
In [4]: un_nombre = 10
```

Ainsi dans *l'enveloppe* appelée **un_nombre** on y met le nombre 10.

C'est ce que l'on appelle **assigner** une **valeur** à une variable ou **stocker** une valeur dans une variable.

```
In [5]: un_nombre = 10
        print un_nombre
```

```
10
```

- Modifier le programme précédent pour stocker 2 dans **un_nombre** puis l'afficher.

Pour stocker des mots ou des phrases, il faut les mettre entre guillemets

```
In [6]: un_mot = "Coucou"
        une_phrase = "ceci est une phrase"
```

- Modifier le programme précédent pour changer le contenu des variables puis ajouter deux lignes pour afficher ce que contiennent ces variables (avec la commande `print`).

1.5 Générateur d'excuses

Quand Loïc fait des bétises, il se fait souvent prendre et doit trouver une excuse le plus rapidement possible. Aidons le en créant un programme qui créé des excuses sur commande.

Ces excuses auront le modèle suivant

```
C'est pas moi! C'est (Bob).
Il a (mangé) une (grenouille) (volante). Alors que moi j'étais tranquillement en train de (faire) un (su
Je suis innocent (monsieur)!
```

Les nombres entre parenthèses seront remplacé par des mots choisis par l'utilisateur de ce générateur.

Pour écrire du texte avec une variable, il faut séparer les parties par une virgule:

```
In [7]: age = 14
        print "J'ai ", age, "ans"
```

```
J'ai  14 ans
```

- Modifier les lignes précédentes pour se présenter **Bonjour, je m'appelle (Bob)**.

Pour demander quelque chose à l'utilisateur, on utilise la fonction `input`

```
In [13]: coupable = input("Nom du coupable (masculin)? ")
         print "C'est pas moi! C'est ", coupable
```

```
Nom du coupable (masculin)? "Marc"
C'est pas moi! C'est  Marc
```

- Finir d'écrire le programme qui permettra de générer une excuse

1.6 Magic 8 ball

Pour ceux qui ne connaissent pas un Magic 8 ball est un boule magique qui répond à toutes les questions (vidéo).

Même si je ne voudrai pas gacher votre âme d'enfant, cette boule magique répond au hasard... Pour gérer le hasard en Python, on utilise **random**. Il faut donc placer la ligne suivante au début du fichier à chaque fois que l'on veut faire quelque chose au hasard.

```
In [14]: import random
```

Nous allons mettre toutes les réponses possibles dans une liste

```
In [15]: reponses = ["reponse 1", "reponse 2", "reponse 3"]
```

Et Python choisira une réponse au hasard

```
In [16]: random.choice(reponses)
```

```
Out[16]: 'reponse 3'
```

- En réutilisant la fonction **input** vue précédement, écrire un programme qui propose de poser une question et qui lui apporte une réponse!

1.7 Deviner un nombre

Nous allons maintenant écrire un programme qui va chercher à faire deviner à l'utilisateur un nombre choisi aléatoirement par Python.

Pour faire choisir un nombre entier à Python, on utilise la fonction `randint(min,max)` (`randint` signifie random integer = entier aléatoire)

```
In [17]: nbr_aleatoire = random.randint(1,20)
         print nbr_aleatoire
```

```
5
```

Faites executer plusieurs fois cette commande pour remarquer qu'elle ne retourne pas toujours le même résultat.

- Completer votre programme pour demander une réponse à l'utilisateur *(utilisez `int(input("question"))` plutôt que `input("question")` pour que la réponse soit transformée en entier)*
- Regarder la section sur Condition if pour adapter l'exemple et savoir lequel du nombre choisi par l'ordinateur ou de l'utilisateur est le plus grand.
- Écrire le programme final qui fait choisir un nombre à l'ordinateur, qui demande un nombre à l'utilisateur puis qui dit si le nombre donné par l'utilisateur est plus grand ou plus petit.
- Pour améliorer notre programme, il faudrait que l'utilisateur puisse rententer de deviner le nombre. Regardez la section Boucle while pour améliorer votre programme.

1.8 Pierre Feuille Ciseau

Vous devez connaître le jeu Pierre-Feuille-Ciseau. Deux adversaires s'affrontent choissicent en même temps Pierre, Feuille ou Ciseau.

Pierre gagne contre ciseau qui gagne contre feuille qui gagne contre pierre...

Nous allons écrire un jeu qui fera s'affronter l'utilisateur contre l'ordinateur.

1.8.1 Choix du joueur et choix de l'ordinateur

- Écrire un programme qui demande le choix de l'utilisateur
- Completer ce programme pour que l'ordinateur choisisse parmi les 3 choix

1.8.2 Comparaison des résultats

- En utilisant les conditions if comparer les deux choix pour determiner qui est le vaincueur.

1.9 Deviner un nombre (amélioration)

Reprenez votre programme qui fait deviner à l'utilisateur un nombre choisi par l'ordinateur. Le soucis de ce programme est que nous n'avons qu'un essai ce qui n'est pas très pratique pour trouver le nombre...

Nous allons améliorer ce programme laisser plus de chance à l'utilisateur. Pour cela nous allons faire une boucle `while` (tant que en français).

```
In [19]: a = 11
         while a < 20: #tant que a est plus petit que 20
             print "un tour"
             a = a + 1
         print "Fini! Et a vaut ", a
```

```
un tour
un tour
un tour
un tour
un tour
un tour
un tour
un tour
un tour
Fini! Et a vaut  20
```

On peut aussi créer une variable `aPasTrouve = True` (True en anglais signifie vrai) et dès que l'utilisateur a trouvé la bonne valeur, on change `aPasTrouve = False` (False en anglais signifie faux).

- Écrire cette amélioration du programme.
- Améliorer votre programme, en comptant le nombre de tentatives.

1.10 Generateur de simleys

Pour ce projet, nous allons programmer un générateur de smileys. Nous allons définir quelquels éléments de ce smiley et l'ordinateur choisira aléatoirement parmi ces éléments pour créer le smiley.

Voici les éléments du smiley.

```
In [58]: LISTE_CHAPEAU       = [u"      \n _===_", "  ___ \n .....", "   _  \n  /_\ ", "  ___ \n (_*_)"]
         LISTE_NEZ       = [u",",    ".",    "_",    " "]
         LISTE_OEIL       = [u".",    "o",    "O",    "-"]
         LISTE_GAUCHE_CORPS_HAUT  = [u" ",    "\\",   " ",    " "]
         LISTE_GAUCHE_CORPS_BAS  = [u"<",    " ",    "/",    " "]
         LISTE_DROITE_CORPS_HAUT = [u" ",    "/",    " ",    " "]
         LISTE_DROITE_CORPS_BAS = [u">",    " ",    "\\",   " "]
         LISTE_TORSE      = [u" : ",      "] [",  "> <",  "   "]
         LISTE_BASE       = [u" : ",      '" "',  "___",  "   "]
```

- Le programme qui séléctionnera aléatoirement des éléments pour construire un smiley de ce type:

```
  _
 /_\
(o O)
<( : )
 (" ")
```

1.11 Polynome du 2nd degre

On veut écrire un programme qui manipule les polynomes du 2nd degré qui l'on améliorera petit à petit.

- Écrire un programme qui demande les valeurs de `a`, `b` et `c` (les coefficients du polynome) et renvoie les racines du polynôme $ax^2 + bx + c$

```
Valeur de a?
4
Valeur de b?
4
Valeur de c?
1
Delta = 0, il y a une racine:
x = 0.5
```

- On veut maintenant améliorer notre programme pour qu'il donne le tableau de signe du polynôme.

```
Valeur de a?
4
Valeur de b?
4
Valeur de c?
1
Delta = 0, il y a une racine:
x = 0.5
Comme a = 4 donc le tableau de signe est

-------------------------
x | -inf     0.5    +inf
-----------------------------
P |    +      0     +
-----------------------------
```

1.12 Chiffrement de Cesar

Le chiffrement de César, utilisé par Jules César pour transmettre des messages secrets dans l'Antiquité, consiste décaler les lettres dans l'alphabet pour écrire un message.

Voici un exemple avec un décalage de 4

```
message >>> bonjour
chiffré >>> fsrnsyv
```

Le `b` devient `f`, `o` devient `s` etc...

- Avec un crayon et du papier chiffrer `bonjour` avec un décalage de 6.

Python est plus fort pour manipuler les nombres que les lettres. Il faudra donc commencer par transformer les lettres en nombre (on appelle cela **encoder**). Il existe une fonction faite pour ça: `ord`

```
In [43]: codage_a = ord("a")
         print codage_a
         codage_b = ord("b")
         print codage_b
         codage_A = ord("A")
         print codage_A
```

```
97
98
65
```

Pour faire la transformation inverse, on utilisera la fonction `chr`

```
In [46]: lettre_100 = chr(100)
         print lettre_100
         lettre_43 = chr(43)
         print lettre_43
```

```
d
+
```

Vous retrouverez tous les caractères et leurs nombres ici. On peut remarquer qu'il y en a 256 (souvenez en vous on s'en servira plus tard)

Il faut donc transformer le message en liste de nombre. Pour cela on utilisera des Boucles for:

```
In [47]: message = "Bonjour"
         message_code = [] #le message codé est pour le moment une liste vide
         for lettre in message:
             print "la lettre ", lettre, "est transformée en le nombre", ord(lettre)
             message_code.append(ord(lettre)) #append signifie ajouter en anglais

         print message_code
```

```
la lettre  B est transformée en le nombre 66
la lettre  o est transformée en le nombre 111
la lettre  n est transformée en le nombre 110
la lettre  j est transformée en le nombre 106
la lettre  o est transformée en le nombre 111
la lettre  u est transformée en le nombre 117
la lettre  r est transformée en le nombre 114
[66, 111, 110, 106, 111, 117, 114]
```

Il faut maintenant décaler tous ces nombres

```
In [48]: message_decale = []
         decalage = 6
         for n in message_code:
             message_decale.append((n + decalage)%256)
         print message_decale
```

```
[72, 117, 116, 112, 117, 123, 120]
```

Le %256 est là pour être sûr qu'aucun nombre ne dépasse 255. Si un nombre dépasse 255, on recommence par 0 (par exemple 259 devient 3 c'est le reste de la division euclidienne de 259 par 256)

Il faut maintenant décodé le `message_decale` pour avoir à nouveau des lettres

```
In [49]: message_chiffre = []
         for n in message_decale:
             message_chiffre.append(chr(n))
         print message_chiffre
```

```
['H', 'u', 't', 'p', 'u', '{', 'x']
```

Ensuite on lie toutes les lettres pour en faire un seul mot

```
In [51]: message_chiffre = ''.join(message_chiffre) # join en anglais signifie joindre
         print message_chiffre
```

```
Hutpu{x
```

Voila votre programme sais chiffrer des messages.

- Écrire un programme qui sais déchiffrer les messages.
- Pour améliorer votre programme, on peut utiliser des fonctions.

1.13 input

Contrairement à ce que l'on a fait hier on ne met plus rien dans les parenthèses d'un `input`.
`input()` est une commande qui va demander à l'utilisateur une information

```
In [34]: print "Comment t'appelles tu?"
         nom = input()
         print "Ah! Tu t'appelles", nom

Comment t'appelles tu?
"plop"
Ah! Tu t'appelles plop
```

Après avoir afficher la question, l'ordinateur se met en pose et attend que l'utilisateur réponde quelque chose. Une fois que l'utilisateur a répondu (a appuyé sur `entrer`) la réponse est stocké, ici, dans la variable `nom`.

1.14 Condition if

Pour différencier plusieurs cas, on utilisera la commande `if` (si), `elif` (sinon si) et `else` (sinon).

```
In [35]: a = 2
         b = 3

         if a > b:
             print "a plus grand que b"
         elif a == b:
             print "a est égal à b"
         else:
             print "a est plus petit que b"

a est plus petit que b
```

1.15 Boucles for

Les boucles **for** permettent de répéter un certain nombre de fois la même action.

- Ajouter 5 fois 100 d'affilé

```
In [52]: a = 2
         print "Au debut a vaut", a
         for i in range(5):
             a = a + 100
             print "a vaut maintenant", a
         print "Finalement a vaut", a

Au debut a vaut 2
a vaut maintenant 102
a vaut maintenant 202
a vaut maintenant 302
a vaut maintenant 402
a vaut maintenant 502
Finalement a vaut 502
```

- On peut aussi parcourir les lettres d'un mot ou d'une phrase

```
In [55]: phrase = "boujour toi"
         print "On enumere >>> ", phrase
         for l in phrase:
             print l
         print "Toutes les lettres ont été énumérées"
```

```
On enumere >>>  boujour toi
b
o
u
j
o
u
r

t
o
i
Toutes les lettres ont été énumérées
```

- Ou encore tous les éléments d'une liste

```
In [57]: liste = ["premier element", 2, "coucou", 1, "aie", 0]
         print "On enumère >>>", liste
         for i in liste:
             print i
         print "Tout a été énuméré"
```

```
On enumère >>> ['premier element', 2, 'coucou', 1, 'aie', 0]
premier element
2
coucou
1
aie
0
Tout a été énuméré
```

1.16 Boucles while

En anglais, `while` signifie **tant que**. On l'utilisera quand on veut refaire quelque plusieurs fois **tant que** quelque chose n'est pas vrais.

Ajouter 3 **tant que** que `u` n'a pas dépassé 19:

```
In [37]: u = 2
         while u < 19:
             u = u + 3
             print u
         print "u a depassé 50 et vaut", u
```

```
5
8
11
14
17
20
u a depassé 50 et vaut 20
```

Poser toujours la même question quand que l'utilisateur n'a pas bien répondu:

```
In [39]: aPasTrouve = True # True signifie vrai en anglais.
         while aPasTrouve:
             print "Quelle est la couleur du cheval blanc d'Henri 4?"
             rep = input()
             if rep == "blanc":
                 print "Bravo!"
                 aPasTrouve = False # False signifie faux en anglais
             else:
                 print "Non! Recommence!"
         print "Voila tu as réussi!"
```

```
Quelle est la couleur du cheval blanc d'Henri 4?
"bleu"
Non! Recommence!
Quelle est la couleur du cheval blanc d'Henri 4?
"jaune"
Non! Recommence!
Quelle est la couleur du cheval blanc d'Henri 4?
"blanc"
Bravo!
Voila tu as réussi!
```

1.17 Fonction

```
In [ ]:
```