Master Sciences, Technologies, Santé
Mention Mathématiques - spécialité Enseignement des mathématiques

Algorithmique et graphes, thèmes du second degré

Programmation Python pour l'enseignement de l'algorithmique au lycée

Éric SOPENA, eric.sopena@labri.fr

Plan de la présentation

- Introduction
- Variables, types, instructions de base
- Structures de contrôle
- Dessiner en Python
- Pour aller plus loin...

Première partie

Introduction

Le langage Python

Le langage Python est né dans les années 1990, au CWI Amsterdam, développé par Guido van Rossum.

Python est un langage libre et gratuit, facile d'accès (il possède une syntaxe très simple), puissant, et est utilisé comme langage d'apprentissage par de nombreuses universités.

Site officiel du langage Python :
http://www.python.org/

Gérard Swinnen, *Apprendre à programmer avec Python 3* :
www.inforef.be/swi/python.htm

Robert Cordeau, *Introduction à Python 3* :
http://hebergement.u-psud.fr/iut-orsay/Pedagogie/MPHY/Python/

Le langage Python

Le langage Python peut être utilisé :

- En mode interprété (chaque ligne du code source est analysée et traduite au fur et à mesure en instructions directement exécutées)

`Mode interprété`

- En mode mixte (le code source est compilé et traduit en *bytecode* qui est interprété par la *machine virtuelle* Python), avec l'environnement de développement IDLE

`Mode mixte`

Deuxième partie

Variables, types, instructions de base

Types et variables

Il n'y a pas de déclaration de variables en Python.

La déclaration s'effectue lors de la première affectation. Le type de la variable est alors déterminé par le type de la valeur (de l'expression) qui lui est affectée.

```
i = 4              # i est de type entier (int)
r = 6.25           # r est de type flottant (float)
ch = 'bonjour'     # ch est de type chaîne (string)
ch2 = "hello"      # ch2 est de type chaîne (string)
encore = True      # encore est de type booléen (bool)
j = int(r)         # j est de type entier (int), valeur 6
```


Types de base

bool - int – float - string

Les expressions de type `bool` peuvent prendre les valeurs `True` ou `False`.

Opérateurs logiques : `and`, `or`, `not`.

Opérateurs de comparaison : `==`, `!=`, `<`, `>`, `<=`, `>=`.

```
fini = False
encore = not fini
petits = ( i < 15 ) and ( j < 8 )
bool1 = (( i >= 8 ) or ( i <= 2 )) and ( j != 4 )
```


Types de base

bool - **int** – float - string

Les valeurs d'une expression de type `int` ne sont limitées que par la taille mémoire...

Opérateurs arithmétiques : `+`, `-`, `*`, `//` (division entière), `%` (modulo), `**` (exponentiation), `abs` (valeur absolue)

```
i = 3 ** ( 5 // 2 )      # i vaut 9
j = abs ( 2*i - 32 )     # j vaut 14
k = j % 5                # k vaut 4
```


Types de base

bool - int – **float** - string

Les valeurs d'une expression de type `float` ont une précision finie (modifiable).

Opérateurs arithmétiques : `+`, `-`, `*`, `/`

Fonctions mathématiques : module `math`

```
import math
print(math.cos(math.pi/3))     # 0.5000000000000001
print(math.factorial(6))       # 720
print(math.log(32,2))          # 5.0
```


Types de base

bool - int – float - **string**

Un littéral de type `string` est délimité par des quotes, simples ou doubles.

Les caractères sont en position 0, 1, etc.

```
ch1 = "aujourd'hui"
print(ch1[0])                  # a

ch2 = 'ceci est une chaine'
ch2[17] = 's'
print(ch2)                     # ceci est une chaise
```


Types de base

bool - int – float - **string**

Différentes primitives sont définies sur les variables de type `string` :

```
chaine1 = 'bonjour'
print(len(chaine1))            # 7 (len = longueur)

chaine2 = chaine1 + ' monde'   # + = concaténation
print(chaine2)                 # bonjour monde

chaine2 = chaine1.upper()      # passage en majuscules
print(chaine2)                 # BONJOUR
```

Types de base

bool - int – float - **string**

```
chaine2 = chaine2.lower()      # passage en minuscules
print(chaine2)                 # bonjour

print(chaine2[0])              # b   (1er indice = 0)
print(chaine2[3])              # j

print(chaine2[1:4])            # onj (indice 1 à 4
                               #     non compris)
print(chaine2[:3])             # bo  (du début à l'indice
                               #     3 non compris)

print(chaine2[4:])             # our (de l'indice 4 à la
                               #     fin)
```


Instructions de base

Affectation : symbole =

Saisie : `<var> = <type> (input (<message>))`

Affichage : `print ( <liste-expressions> )`

```
i = 14

nbTours = int(input('Nombre de tours : '))
nomEleve = string(input("Nom de l'élève : "))

print('resultat :',res)
print('le produit de',a,'par',b,'vaut',a*b)
```


Instructions de base

Options d'affichage

`sep = '-'` : séparateur entre éléments de la liste (espace par défaut)

`end = '***\n'` : en fin d'affichage (fin de ligne par défaut)

```
i = 3
j = 17
print(i,j)               # 3 17
print(i,j,sep='')        # 317
print(i,j,sep=' --- ')   # 3 --- 17
print(i,j,i+j,sep='-',end='***')
                         # 3-17-20***
```


Instructions de base

Quelques caractères spéciaux

`\'` : apostrophe
`\"` : guillement
`\n` : fin de ligne
`\t` : tabulation
`\a` : sonnerie (bip)

```
i = 3
j = 17
print(i,j,sep='\t')          # 3   17
print('aujourd\'hui')        # aujourd'hui
print(i,j, sep=' **\n')      # 3 **
                             # 17
```


Instructions de base

Le module random offre un certain nombre de fonctions outils permettant de générer des nombres de façon aléatoire.

```
import random     # on intègre le module random
...
random.seed() # initialise le générateur
i = random.randrange(12)
              # nombre entier aléatoire entre 0 et 11
i = random.randrange(4,12)
              # nombre entier aléatoire entre 4 et 11
i = random.randrange(4,12,3)
              # nombre entier aléatoire parmi 4, 7, 10
r = random.random()
              # nombre flottant aléatoire entre 0.0 et
              # 1.0 non compris (16 décimales)
```


Troisième partie

Structures de contrôle

Blocs d'instructions

Un bloc d'instructions est une séquence d'instructions (une par ligne) ayant même *indentation* (décalage en début de ligne).

(pas de délimiteurs de type `début ... fin`)

L'indentation des blocs imbriqués progresse de 4 en 4 (touche `tabulation` sous IDLE).

```
i = 3              # bloc A
...
    j = 17         # début du bloc B
    print(i,j)
...                # fin du bloc B, suite du bloc A
print(i,j,i+j)
```


La structure if-else

La structure `if-else` est la traduction du si-alors-sinon algorithmique (la partie `else` est facultative).

Format général :

```
if <condition> :
    <bloc>
else :
    <bloc>
```

```
if i>j :
   print('maximum :', i)
else :
   print('maximum :', j)
```


La structure if-elif-else

La structure `if-elif-else` est la traduction du selon-que algorithmique (la partie `else` est facultative).

```
note = float(input('Note du bac :'))

if note >= 16 :
   print ('mention TB')
elif note >= 14 :
   print ('mention B')
elif note >= 12 :
   print ('mention AB')
elif note >= 10 :
   print ('mention Passable')
else :
   print ('désolé...')
```


La boucle while

La boucle `while` est la traduction du tant-que algorithmique.

Format général :

```
while <condition> :
    <bloc>
```

```
n = int(input('nombre :'))
nn = n
fact = 1
while nn > 1 :
   fact = fact * nn
   nn = nn - 1
print('La factorielle de',n,'vaut',fact)
```


La boucle for

La boucle `for` est la traduction du pour algorithmique.

Format général :

```
for <var> in range(<deb>,<fin+1>{,<pas>}) :
        <bloc>
```

```
n = int(input('nombre :'))
   # affichage de la table de multiplication de n
for i in range(0,11) :
   print(n,'*',i,'vaut',n*i)

   # idem... mais à l'envers
for i in range(10,-1,-1) :
   print(n,'*',i,'vaut',n*i)
```


Quatrième partie

Dessiner en Python

Le module turtle

Le module `turtle` permet de dessiner dans une fenêtre graphique à l'aide d'un ensemble de primitives dédiées.

```
# script exemple turtle : dessin de différentes figures
    # import module de dessin
import turtle
    # reinitialise la fenêtre
turtle.reset()
    # titre de la fenêtre
turtle.title("Dessin de différentes figures")
    # paramètres de dessin
turtle.color('red')   # couleur de trait
turtle.width(10)      # épaisseur du trait
turtle.speed(3)       # vitesse d'exécution du tracé
```


Le module turtle

```
    # dessin d'un carré
monCote = 200
turtle.pendown()      # on abaisse le crayon
for i in range(4):
    turtle.forward(monCote)    # la tortue avance
    turtle.left(90)            # la tortue tourne à
                               # gauche


    # partons ailleurs dessiner un cercle vert
turtle.penup()           # on lève le crayon avant
                         #     de se déplacer
turtle.goto(-60,0)       # on se déplace
turtle.pendown()         # on abaisse le crayon
turtle.color('green')    # on change de couleur
turtle.circle(100)       # on trace un cercle
```


Le module turtle

```
    # puis un arc de cercle jaune
turtle.penup()
turtle.goto(-260,0)
turtle.pendown()
turtle.color('yellow')
turtle.circle(100,90)          # arc de cercle 90°


    # et enfin un gros point bleu
turtle.penup()
turtle.goto(-240,100)
turtle.pendown()
turtle.dot(40,'blue')          # diamètre du point 40
```

Démonstration

Troisième partie

Pour aller (un tout petit peu) plus loin...

Le type complex

Python offre le type numérique `complex` (notation cartésienne avec deux flottants, partie imaginaire suffixée par j), qui s'utilise ainsi :

```
print(1j)                    # 1j
print(3.4 + 1.2j)            # (3.4+1.2j)
print((3.4 + 1.2j).real)     # 3.4
print((3.4 + 1.2j).imag)     # 1.2
print(abs(3 + 9j))      # 9.486832980505138 (module)
```

Le module `cmath` donne accès à d'autres primitives. Pour plus de détails, voir :

http://docs.python.org/py3k/library/cmath.html#module-cmath

Les listes

Une liste est une collection ordonnée d'éléments, éventuellement de nature distincte. Les éléments sont repérés par leur numéro d'ordre au sein de la liste (ces numéros démarrent à 0).

```
joursSemaine = ['lun', 'mar', 'mer', 'jeu', 'ven']
pairs = [0, 2, 4, 6, 8]
reponses = ['o', 'O', 'n', 'N']
listeBizarre = [jours, 2, 'hello', 54.8, 2+7j]
```

```
liste = [ 2, 3, 4 ]
print(liste)            # [2, 3, 4]
print(liste[1])         # 3
liste[2] = 28
print(liste)            # [2, 3, 28]
```

Les listes

```
liste = list(range(4))  # convertit en liste
print(liste)            # [0, 1, 2, 3]
print(1 in liste)       # True
print(5 in liste)       # False
liste = [6, 8, 1, 4]
liste.sort()            # tri de la liste
print(liste)            # [1, 4, 6, 8]
liste.append(14)        # ajout en fin de liste
print(liste)            # [1, 4, 6, 8, 14]
liste.reverse()         # retourne la liste
print(liste)            # [14, 8, 6, 4, 1]
liste.remove(8)         # supprime la première
                                  # occurrence de 8
print(liste)            # [14, 6, 4, 1]
```


Les listes

```
liste = [14, 6, 4, 1]
i = liste.pop()         # récupère et supprime le
                        # dernier élément
print(i)                # 1
print(liste)            # [14, 6, 4]
liste.extend([6,5])
print(liste)            # [14, 6, 4, 6, 5]
print(liste.count(6))   # 2 (nombre d'éléments)
print(liste[1:3])       # [6,4] (pos. 1 à 3 non compris)
liste[0:2] = [5,4,3]    # remplace une portion de liste
print(liste)            # [5, 4, 3, 4, 6, 5]
liste[4:] = []          # positions 4 à fin de liste
print(liste)            # [5, 4, 3, 4]
liste[:2] = [1,1,1,1]   # positions 1 à 2 non compris
print(liste)            # [1, 1, 1, 1, 3, 4]
```


Définition et utilisation de fonctions

Les fonctions permettent de décomposer une tâche en tâches « plus simples » et souvent d'éviter des répétitions de portions de code.

Elles permettent par ailleurs la réutilisation de code (en recopiant la fonction d'un script à un autre ou, plus efficacement, en utilisant le mécanisme d'import).

Format *général* de définition d'une fonction :

```
def nomFonction(paramètres):
    """Documentation de la fonction."""
    <bloc_instructions>
```


Définition et utilisation de fonctions

Voici un exemple de fonction simple :

```
def maximum(a,b) :
    """détermine le maximum des valeurs a et b """
    if a>b :
        return a
    else :
        return b
```

Et quelques exemples d'utilisation :

```
a = 8
b = 11
c = maximum(a,b)
print(maximum(6,21))                # 21
print(maximum(16,maximum(43,2)))    # 43
print(maximum('bonjour','salut'))   # 'salut'
print(maximum([6,1],[4,11,3]))      # [6,1]
```


Définition et utilisation de fonctions

On peut également définir des fonctions ayant plusieurs résultats, en utilisant des `return` *multiples* :

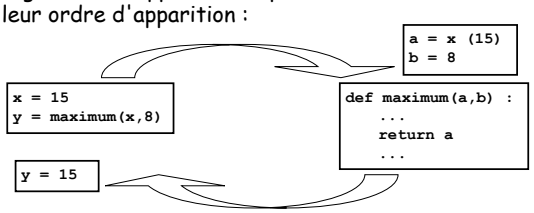
```
def divisionEuclidienne(a,b) :
    """reste et quotient de la division de a par b"""
    return a//b, a%b
```

Exemple d'utilisation :

```
q,r = divisionEuclidienne(17,4)  # q reçoit le quotient,
                                 # r le reste
print(q)      # 4
print(r)      # 1
```


Définition et utilisation de fonctions

Le lien entre paramètres de la fonction et arguments d'appel se fait par affectation, selon leur ordre d'apparition :



Les arguments d'appel sont donc non modifiables (car recopiés dans des variables locales de la fonction).

Merci de votre attention...

Virgil Solis - Apollon tuant Python - *Wikimedia Commons*