Informatique en CPGE (2018-2019)
Les fonctions

S. B.

Lycée des EK

6 novembre 2018

Nous avons déjà rencontré des fonctions, par exemple la fonction `print`, la fonction `abs`, des fonctions du module `math` comme `sqrt`. Nous pouvons aussi définir et utiliser nos propres fonctions.

Définition d'une fonction

En Python la définition d'une fonction est de la forme :

```
def nomDeLaFonction(paramètres) :
    corpsDeLaFonction
```

`def` est un mot clé du langage qui dit à Python qu'une fonction va être définie, ce mot est suivi du nom de la fonction puis d'une parenthèse, entourant un ou plusieurs paramètres séparés par des virgules ou aucun paramètre, et enfin le caractère " :".
Le corps de la fonction est un bloc de code indenté par rapport à la ligne de définition.

Exemple : on étudie le mouvement vertical d'une balle lancée en l'air. Supposons que la position verticale de la balle à l'instant t est donnée par $y(t) = v_0 t - \frac{1}{2}gt^2$, où v_0 est la vitesse initiale et g l'accélération due à la gravité (environ 9,81).
On peut définir une fonction nommée "position" par :

```
def position(v,t) :
    return v*t-0.5*9.81*t**2
```

On appelle la fonction, en précisant la valeur des paramètres, par exemple dans l'interpréteur :

```
>>> position(4,0.6)
>>> 0.6341999999999999
```

Lors de l'appel, on affecte les valeurs 4 et 0.6 respectivement aux paramètres v et t de la fonction (on dit aussi les **arguments** de la fonction) et le corps de la fonction est exécuté.

Si le corps de la fonction contient l'instruction "return" alors l'appel de la fonction est une expression qui a donc une valeur.

```
y=position(4,0.6)
```

Après cette instruction,la variable y a pour valeur 0.6341999999999999.

Un autre exemple :

```
def f(x,y) :
    z=3*x+y
    return z
```

L'appel `f(2,5)` renvoie 11,
l'appel `f("ta","ti")` renvoie "tatatati".

S'il n'y a pas d'instruction "return" dans le corps de la fonction, alors l'appel de la fonction a la valeur "None". Ce type de fonction s'appelle une procédure. Par exemple, la fonction "print" : après l'instruction `y=print(3)`, y a la valeur None.

Autre exemple :

```
def successeur(n) :
    n+=1
    print(n)
```

Il faut bien distinguer les arguments déclarés dans la définition de la fonction et les paramètres qui sont utilisés au moment de l'appel de la fonction. Les paramètres peuvent avoir des noms différents ou identiques à ceux des arguments ; cela n'a aucune importance, car les noms des arguments n'ont de signification qu'à l'intérieur de la définition de la fonction, et sont inconnus en dehors de la fonction. Le code suivant fonctionne parfaitement :

```
def position(v,t) :
    return v*t-0.5*9.81*t**2
vitesse=4
temps=0.6
y=position(vitesse,temps)
```

Si dans la définition d'une fonction on fait référence à une variable x, Python vérifie dans l'**espace local de la fonction** si cette variable existe. Cet espace contient les paramètres qui sont passés à la fonction et les variables définies dans son corps. Si la variable x n'existe pas dans l'espace local de la fonction, Python va regarder dans l'espace local dans lequel la fonction a été appelée. Et là, s'il trouve bien la variable x il peut l'utiliser.

Quand Python trouve une instruction d’affectation, comme par exemple `var = valeur1`, il va changer la valeur de la variable "var" dans l’espace local de la fonction. Mais cet espace local est détruit après l’appel de la fonction.
Donc une fonction ne peut pas modifier, par affectation, la valeur d’une variable extérieure à son espace local.

Dans le code suivant, la variable x utilisée dans la fonction est distincte de la variable x définie au début du programme (x=3) et n'existe plus après l'appel de la fonction. Après l'instruction `f(x)`, l'espace local de la fonction f est détruit. On pourrait d'ailleurs remplacer "x" par n'importe quel autre identificateur dans la définition de la fonction.

```
x=3
def f(x) :
    x+=2
    print(x)
f(x)    # affiche 5
print(x)    # affiche 3
```

De même la fonction d'échange suivante ne produit pas le résultat espéré :

```
a=2
b=5
def echange(x,y) :
    x,y=y,x
echange(a,b)
print(a,b)    #affiche 2 5
```

Il existe un moyen de modifier, dans une fonction, des variables extérieures à celle-ci. On utilise pour cela des **variables globales**. Pour déclarer à Python, dans le corps d'une fonction, que la variable qui sera utilisée doit être considérée comme globale, on utilise le mot-clé **global**.
Le code suivant permet de modifier la variable x extérieure à la fonction.

```
x=3
def f() :
    global x
    x+=2
    print(x)
f()    #affiche 5
print(x)    #affiche 5
```

Plusieurs algorithmes concernant l'étude de fonctions mathématiques sont étudiés au lycée.

L'analyse mathématique, faite au préalable, permet grâce au théorème des valeurs intermédiaires d'affirmer :
si f est continue croissante sur $[a; b]$ et si $k \in [f(a); f(b)]$,
ou si f est continue décroissante sur $[a; b]$ et si $k \in [f(b); f(a)]$,
alors l'équation $f(x) = k$ admet une solution unique α dans $[a; b]$.

Méthode par dichotomie

Cette méthode permet de déterminer un encadrement de la solution α. A chaque étape l'amplitude de l'intervalle contenant la solution est divisé par deux. En dix étapes, on gagne environ trois décimales puisque $2^{10} \simeq 1000$.

Exemple avec f, a, b et e en entrées et un encadrement de la solution en sorties :

```
def dicho(f,a,b,e) :
    while b-a>e :
        m=(a+b)/2
        if f(m)*f(a)>0 :
            a=m
        else :
            b=m
    return a,b
```

On pourrait aussi utiliser une méthode par "balayage" ou une méthode par "calcul de valeurs aléatoires". Mais ces méthodes sont en général plus longues et sont plutôt utilisées pour la recherche d'extremum.

Algorithme 1 : déterministe à pas constant

```
def minmax(f,a,b,n) :
    mini=f(a)
    maxi=f(a)
    dx=(b-a)/n
    x=a
    for k in range(n) :
        x=x+dx
        y=f(x)
        if f(x)>maxi :
            maxi=f(x)
        if f(x)<mini :
            mini=f(x)
    return mini,maxi
```

Algorithme 2 : tabulation « aléatoire » d'une fonction

```
from random import random
def minmax(f,a,b,n) :
    mini=f(a)
    maxi=f(a)
    for k in range(n) :
        x=a+(b-a)*random()
        y=f(x)
        if y>maxi :
            maxi=y
        if y<mini :
            mini=y
    return mini,maxi
```

Test de la monotonie

Attention, cet algorithme peut permettre de montrer que la fonction **n'est pas monotone** ou que la fonction **peut être monotone** mais dans ce cas il se peut qu'elle ait des variations de sens contraires sur certains intervalles très courts. Les variables en entrées sont f, a, b, n.

Programme :

```
def monotone(f,a,b,n) :
    dx=(b-a)/n
    sens=f(b)-f(a)
    x=a
    for k in range(n) :
        if (f(x+dx)-f(x))*sens<0 :
            print("la fonction n'est pas monotone")
            return x,x+dx
        x=x+dx
    print("La fonction semble monotone.")
    return 0
```

On peut encadrer une intégrale pour une fonction monotone positive ou déterminer une valeur approchée d'une intégrale.

Encadrement

Soit f une fonction continue, positive et croissante sur $[a; b]$ et n un entier strictement positif.

On partage $[a; b]$ en n intervalles d'amplitude $h = \dfrac{b-a}{n}$.

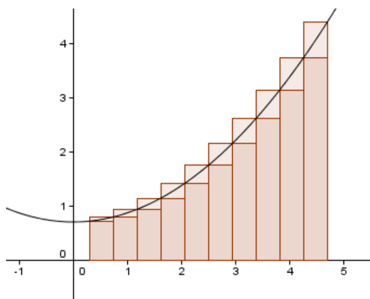
Sur chaque intervalle $[a+ih; a+(i+1)h]$, où i varie de 0 à $n-1$, on a l'encadrement

$$f(a+ih) \leq f(x) \leq f(a+(i+1)h)$$

Donc $hf(a+ih) \leq \displaystyle\int_{a+ih}^{a+(i+1)h} f(x)\mathrm{d}x \leq hf(a+(i+1)h)$, où à gauche et à droite, les produits sont des aires de rectangles.

On obtient alors l'encadrement suivant :

$$h\sum_{i=0}^{n-1} f(a+ih) \leq \int_a^b f(x)\mathrm{d}x \leq h\sum_{i=0}^{n-1} f(a+(i+1)h)$$



On peut alors écrire un programme afin d'effectuer ce calcul.
Les variables en entrées sont f, a, b et n ; en sorties les valeurs encadrant l'intégrale.

Programme

```
def integrale(f,a,b,n) :
    h=(b-a)/n
    integ_inf=0
    integ_sup=0
    x=a
    for i in range(n) :
        integ_inf=integ_inf+f(x)
        x=x+h
        integ_sup=integ_sup+f(x)
    integ_inf=h*integ_inf
    integ_sup=h*integ_sup
    return integ_inf, integ_sup
```

Si la fonction f est simplement continue sur $[a; b]$, on peut obtenir une valeur approchée de l'intégrale en approximant $f(x)$, sur chaque intervalle $[a + ih; a + (i + 1)h]$, par $f(a + (i + 1/2)h)$: c'est la **méthode des rectangles.**

Si on approche $f(x)$, sur chaque intervalle $[a + ih; a + (i + 1)h]$, par $[f(a + ih) + f(a + (i + 1)h)]/2$, c'est la **méthode des trapèzes.**