Atelier d'informatique

Épisode III : Fonctions

27 février 2017

Atelier d'informatique

Épisode III : Fonctions

27 février 2017

Dans ce chapitre, nous verrons une autre façon d'éviter de réécrire 9001 fois la même chose : les *fonctions*.

Fonction ? Comme $f(x)$?

Dans un langage de programmation, les *fonctions* sont des objets qui prennent en argument des variables et effectuent une série d'instructions.

Fonction ? Comme $f(x)$?

Dans un langage de programmation, les *fonctions* sont des objets qui prennent en argument des variables et effectuent une série d'instructions.

Tout d'abord, débarrassons-nous d'idées préconçues :

Fonction ? Comme $f(x)$?

Dans un langage de programmation, les *fonctions* sont des objets qui prennent en argument des variables et effectuent une série d'instructions.

Tout d'abord, débarrassons-nous d'idées préconçues :

- Pas exactement la même chose qu'en mathématiques : une fonction n'est pas obligée de renvoyer une valeur !

Fonction ? Comme $f(x)$?

Dans un langage de programmation, les *fonctions* sont des objets qui prennent en argument des variables et effectuent une série d'instructions.

Tout d'abord, débarrassons-nous d'idées préconçues :

- Pas exactement la même chose qu'en mathématiques : une fonction n'est pas obligée de renvoyer une valeur !
- Une fonction ne prend pas nécessairement d'argument ; dans ce cas-là, on parle souvent de *procédure*. Python ne fait pas la différence au niveau de sa syntaxe.

Sémantique

Sous Python, on définit une fonction selon la syntaxe suivante :

Sémantique

Sous Python, on définit une fonction selon la syntaxe suivante :

```
def nom_fonction(*args, **kwargs):
    < instructions >
    return < valeur >
```

Sémantique

Sous Python, on définit une fonction selon la syntaxe suivante :

```
def nom_fonction(*args, **kwargs):
    < instructions >
    return < valeur >
```

où `args` est un ensemble d'*arguments*, séparés par des virgules, et `kwargs` est un ensemble d'argument-clés ou *keyword arguments*, qu'il faut introduire via un mot-clé lorsqu'on fait appel à la fonction.

Sémantique

Exemple

Par exemple, on définit la fonction `f` suivante :

Sémantique

Exemple

Par exemple, on définit la fonction `f` suivante :

```
def f(nombre, nom="eddy malou"):
```

Sémantique

Exemple

Par exemple, on définit la fonction `f` suivante :

```
def f(nombre, nom="eddy malou"):
    if n == 42:
        return "Je suis le 1er savant de la"+\
        "République Démocratique du Congo"+\
        nom
```

Sémantique

Exemple

Par exemple, on définit la fonction `f` suivante :

```
def f(nombre, nom="eddy malou"):
    if n == 42:
        return "Je suis le 1er savant de la"+\
        "République Démocratique du Congo"+\
        nom
    else:
        print("Vous me décevez")
```

Sémantique

Exemple

Par exemple, on définit la fonction `f` suivante :

```
def f(nombre, nom="eddy malou"):
    if n == 42:
        return "Je suis le 1er savant de la"+\
        "République Démocratique du Congo"+\
        nom
    else:
        print("Vous me décevez")
```

Que fait cette fonction ?

Sémantique

Pour faire appel à une fonction qui s'appelle `f`, par exemple, on utilise la syntaxe évidente :

```
f(arg1, arg2, ..., kwarg1 = < valeur >, ...)
```

Sémantique

Pour faire appel à une fonction qui s'appelle `f`, par exemple, on utilise la syntaxe évidente :

```
f(arg1, arg2, ..., kwarg1 = < valeur >, ...)
```

Exemple

`f(41, nom = "jean-marc")` fait appel à la fonction `f`, qui a un argument dépendant de la position, qui a pour valeur `41`, et un argument-clé nommé `nom`, qui prend pour valeur la chaîne `"jean-marc"`.

Exercices

Exercice 1

Écrire une fonction f qui correspond à la fonction réelle

$$\begin{array}{rcl} \mathbb{R} & \longrightarrow & \mathbb{R} \\ x & \longmapsto & \frac{1}{1+x^2} \end{array}$$

L'essayer sur quelques valeurs.

Exercices

Exercice 2

Écrire une fonction `f` qui correspond à la fonction valeur absolue $|x|$ définie par

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

L'essayer sur quelques valeurs.

Exercices

Exercice 2

Écrire une fonction `f` qui correspond à la fonction valeur absolue $|x|$ définie par

$$|x| = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{cases}$$

L'essayer sur quelques valeurs.

On verra plus tard comment représenter graphiquement des fonctions numériques, via le module `matplotlib`.

Exercice 3

Écrire une fonction `divise` prenant en argument deux entiers `n` et `d` et renvoie un booléen qui vaut `True` si d divise n, et `False` sinon.

Exercice 3

Écrire une fonction `divise` prenant en argument deux entiers `n` et `d` et renvoie un booléen qui vaut `True` si d divise n, et `False` sinon.
Indication L'entier d divise l'entier n lorsque le reste de la division de n par d est nul.

Exercice 4 (Maximum d'une liste)

Écrire une fonction `maxListe` qui prend en argument une liste `l` et renvoie son plus grand élément, et l'indice du premier endroit où il se situe.

Exercice 4 (Maximum d'une liste)

Écrire une fonction `maxListe` qui prend en argument une liste `l` et renvoie son plus grand élément, et l'indice du premier endroit où il se situe.

Indication Utiliser deux variables `valMax` et `posMax`, initialement égales à `l[0]` et `0` respectivement, puis une boucle itérative `for` parcourant la liste pour les mettre à jour.

Exercice 4 (Maximum d'une liste)

Écrire une fonction `maxListe` qui prend en argument une liste `l` et renvoie son plus grand élément, et l'indice du premier endroit où il se situe.

Indication Utiliser deux variables `valMax` et `posMax`, initialement égales à `l[0]` et `0` respectivement, puis une boucle itérative `for` parcourant la liste pour les mettre à jour.

Trouver le maximum (ou le minimum) d'une liste donnée est un problème classique. On le rencontrera encore plus tard quand on s'intéressera aux problèmes de tri de listes.

Exercice 5 (Suite)

On définit la suite (u_n) suivante : $u_0 \in \mathbb{R}$, et pour tout $n \in \mathbb{N}$,

$$u_{n+1} = \sin(u_n).$$

Exercice 5 (Suite)

On définit la suite (u_n) suivante : $u_0 \in \mathbb{R}$, et pour tout $n \in \mathbb{N}$,

$$u_{n+1} = \sin(u_n).$$

Écrire une fonction `suite` qui prend en argument un entier `N` et un réel `u0` puis génère la liste des termes de (u_n) de 0 à N.

Exercice 5 (Suite)

On définit la suite (u_n) suivante : $u_0 \in \mathbb{R}$, et pour tout $n \in \mathbb{N}$,

$$u_{n+1} = \sin(u_n).$$

Écrire une fonction `suite` qui prend en argument un entier `N` et un réel `u0` puis génère la liste des termes de (u_n) de 0 à N. Pour utiliser la fonction $\sin$, on l'importera au début du script comme suit :

```
from numpy import sin
```

Exercice 5 (Suite)

On définit la suite (u_n) suivante : $u_0 \in \mathbb{R}$, et pour tout $n \in \mathbb{N}$,

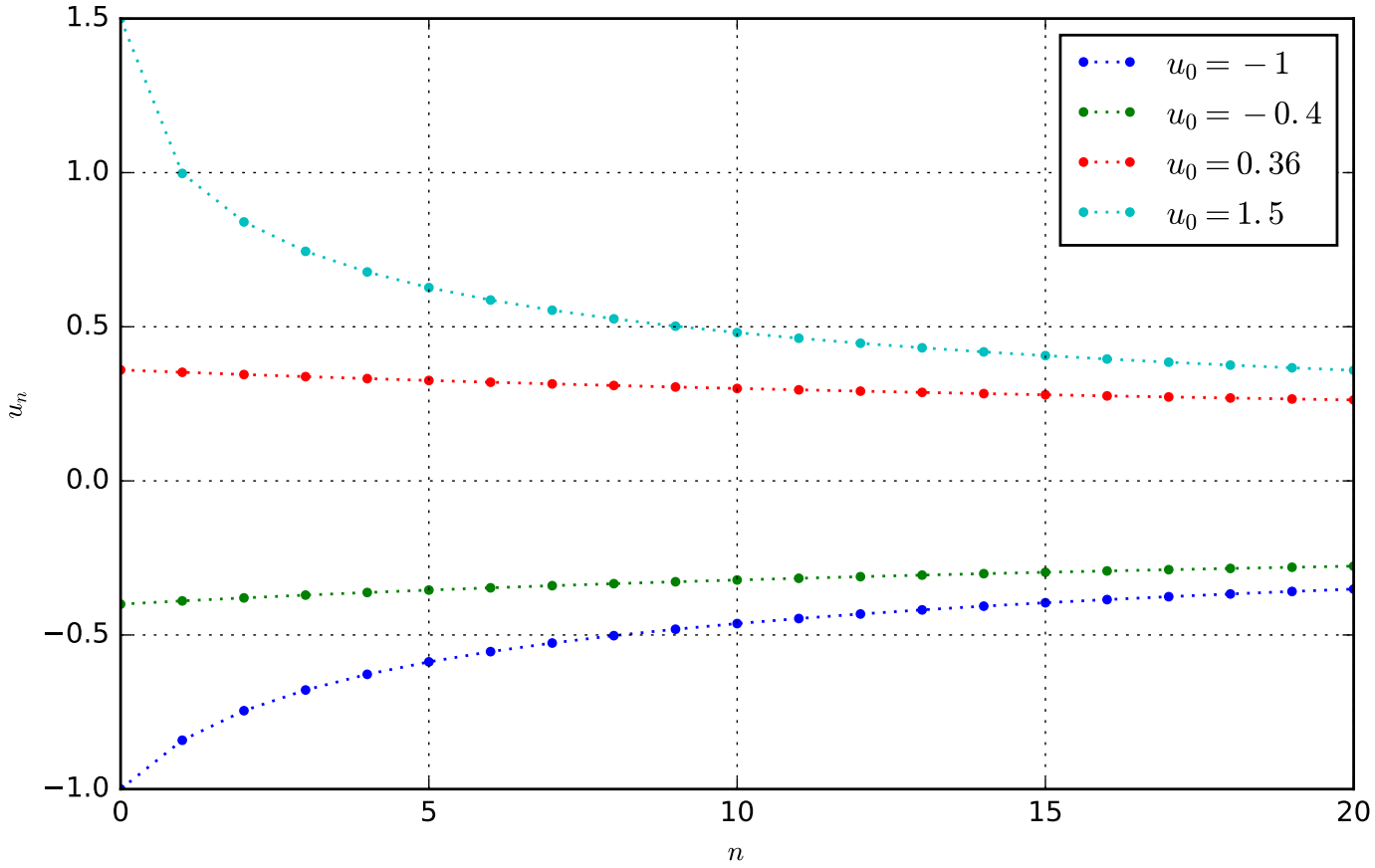
$$u_{n+1} = \sin(u_n).$$

Écrire une fonction `suite` qui prend en argument un entier `N` et un réel `u0` puis génère la liste des termes de (u_n) de 0 à N. Pour utiliser la fonction $\sin$, on l'importera au début du script comme suit :

```
from numpy import sin
```

Indication On initialisera la liste en donnant l'instruction `termes = [u0]` au début de la fonction, avant de passer dans une boucle `for i in range(N)`. À la i-ème étape de la boucle, le dernier élément de la liste, `termes[-1]`, est égal à u_i.

Pour info, voici quelques tracés de la suite :



Correction de l'exercice 5

```
def suite(u0, N):
    termes = [u0]
    for i in range(N+1):
        ui = termes[-1]
        termes.append(sin(ui))
    return termes
```