Initiation à Python

Séance 3

Valentin Bahier

2020-2021

Plan

Plan

Plan

Un module, qu'est-ce que c'est ?

Un **module** est un fichier Python (avec l'extension *.py*) qui contient des fonctions.

L'intérêt d'un module est de rassembler des fonctions utiles dans un fichier, séparément du programme principal afin de ne pas l'encombrer. En créant des modules on préserve ainsi la lisibilité globale du code.

Création d'un module

Exercice 19

Créer un fichier Python contenant les deux fonctions suivantes

```
1 def carre(valeur):
2     resultat = valeur**2
3     return resultat
4
5 def cube(valeur):
6     resultat = valeur**3
7     return resultat
```

et le sauvegarder sous le nom `puissance.py` sur l'ordinateur.

Création d'un module

Exercice 19

Créer un fichier Python contenant les deux fonctions suivantes

```
1 def carre(valeur):
2     resultat = valeur**2
3     return resultat
4
5 def cube(valeur):
6     resultat = valeur**3
7     return resultat
```

et le sauvegarder sous le nom `puissance.py` sur l'ordinateur.

$\Rightarrow$ Et voilà, nous venons de créer notre premier module ! Il ne reste plus qu'à voir comment s'en servir depuis un autre script.

Plan

Importation

Pour se servir des fonctions d'un module depuis un autre fichier Python (celui qui contient notre programme principal par exemple), il va falloir les **importer**.

⚠ Il est nécessaire que le module se trouve dans le même répertoire que le fichier dans lequel on souhaite l'utiliser (ou alors dans un répertoire où est installé Python).

Importation d'une fonction du module

Exercice 20

Dans le même dossier que celui dans lequel a été enregistré le module `puissance`, enregistrer et exécuter le script suivant

```
1 from puissance import carre
2
3 a = 5
4 b = carre(a)
5 print("le carré de", a, "vaut", b)
```

Importation d'une fonction du module

Exercice 20

Dans le même dossier que celui dans lequel a été enregistré le module `puissance`, enregistrer et exécuter le script suivant

```
1 from puissance import carre
2
3 a = 5
4 b = carre(a)
5 print("le carré de", a, "vaut", b)
```

On peut aussi importer plusieurs fonctions d'un coup comme ceci

```
1 from puissance import carre, cube
```

ou toutes les fonctions du module grâce à l'astérisque

```
1 from puissance import *
```

⚠ Cette dernière technique est très déconseillée dès que plusieurs modules sont utilisés, à cause du risque que des fonctions provenant de différents modules portent le même nom.

Importation du module

Exercice 21

Modifier les lignes 1 et 4 du script précédent comme suit :

```
1 import puissance
2
3 a = 5
4 b = puissance.carre(a)
5 print("le carré de", a, "vaut", b)
```

Importation du module

Exercice 21

Modifier les lignes 1 et 4 du script précédent comme suit :

```
1 import puissance
2
3 a = 5
4 b = puissance.carre(a)
5 print("le carré de", a, "vaut", b)
```

On peut aussi ajouter un **alias** par la commande `as` pour raccourcir l'appel du module

```
1 import puissance as ps
2
3 a = 5
4 b = ps.carre(a)
5 print("le carré de", a, "vaut", b)
```

$\Rightarrow$ Une fois un module importé, on accède à une fonction de celui-ci par la syntaxe

module.fonction()

Plan

Mais alors une librairie c'est quoi ?

Une **librairie** (ou **package**) est un dossier contenant plusieurs modules. Afin que Python sache qu'il s'agit bien d'une librairie et non d'un simple dossier contenant des scripts, une librairie contient généralement un fichier vide nommé `__init__.py`.

Pour accéder à une fonction d'un module d'une librairie, on importe d'abord la libraire (exactement comme un module), puis on écrit

librairie.module.fonction()

⚠ Bien souvent par abus de langage les *librairies* sont également appelées *modules*. Nous nous autoriserons cet abus dans le cadre de ce cours.

Plan

Plan

Motivations

Le module **NumPy** tire son nom de la contraction entre *numerical computing* et *Python*. Il offre un grand nombre d'outils pour effectuer des calculs numériques. Particulièrement, NumPy permet une gestion facilitée des *tableaux de nombres*.

Il est d'usage de l'importer avec l'alias `np` comme ceci

```
1 import numpy as np
```

Plan

Création manuelle d'un tableau de nombres

Exercice 22

Entrer les instructions suivantes dans la console

```
1 import numpy as np
2 a = np.array([[1, 2, 3], [4, 5, 6]])
3 a
4 type(a)
5 a[0,2]
6 a[1,1]
```

Création manuelle d'un tableau de nombres

Exercice 22

Entrer les instructions suivantes dans la console

```
1 import numpy as np
2 a = np.array([[1, 2, 3], [4, 5, 6]])
3 a
4 type(a)
5 a[0,2]
6 a[1,1]
```

La variable a est un tableau avec deux lignes et trois colonnes.
`a[0,2]` vaut 3 car c'est l'élément placé à la ligne d'indice 0, et colonne d'indice 2.

$\Rightarrow$ La fonction `array` transforme des listes en tableaux.
Comme pour les listes, les indices commencent à 0 et non 1.

Plan

Tableaux de nombres à pas constants

Dans l'objectif de tracer des graphes, il est intéressant de pouvoir construire facilement des tableaux de nombres d'une seule ligne (**tableaux à une dimension**) dont les valeurs sont uniformément espacées (l'espacement entre deux valeurs consécutives est alors le **pas**), ce que font précisément les fonctions `arange` et `linspace` .

`np.arange(`début`,`fin`,`pas`)` (par défaut : début $= 0$ et pas $= 1$)

`np.linspace(`début`,`fin`,`nombre de valeurs`)` (par défaut : nombre de valeurs $= 50$)

⚠ Pour `np.arange` la fin est exclue, tandis que pour `np.linspace` la fin est incluse.

Création d'un tableau à pas constant

Exercice 23

- Construire un tableau contenant tous les entiers de 0 à 22 (inclus) dans l'ordre croissant.
- Construire un tableau contenant tous les entiers de -15 à 10 (inclus) dans l'ordre croissant.
- Deviner comment construire les deux précédents tableaux mais en mettant leurs valeurs dans l'ordre décroissant.

Création d'un tableau à pas constant

Exercice 23

- Construire un tableau contenant tous les entiers de 0 à 22 (inclus) dans l'ordre croissant.
- Construire un tableau contenant tous les entiers de -15 à 10 (inclus) dans l'ordre croissant.
- Deviner comment construire les deux précédents tableaux mais en mettant leurs valeurs dans l'ordre décroissant.

Réponse :

```
1 tableauUn = np.arange(23)   # pas besoin de préciser
    le début (car vaut 0 par défaut), ni le pas (car
    vaut 1 par défaut)
2 tableauDeux = np.arange(-15,10)  # pas besoin de pré
    ciser le pas
3 tableauUnRetro = np.arange(22,-1,-1)  # astuce de
    mettre -1 pour le pas
4 tableauDeuxRetro = np.arange(10,-16,-1)
```

$\Rightarrow$ La fonction `arange` est pratique quand on connaît le pas.

Création d'un tableau à pas constant

Exercice 24

- Construire un tableau contenant 50 valeurs régulièrement espacées entre -6 et -2 (inclus).
- Construire un tableau contenant 17 valeurs régulièrement espacées entre 5 et 10 (inclus).
- Deviner comment construire les deux précédents tableaux mais en mettant leurs valeurs dans l'ordre décroissant.

Création d'un tableau à pas constant

Exercice 24

- Construire un tableau contenant 50 valeurs régulièrement espacées entre -6 et -2 (inclus) dans l'ordre croissant.
- Construire un tableau contenant 17 valeurs régulièrement espacées entre 5 et 10 (inclus) dans l'ordre croissant.
- Deviner comment construire les deux précédents tableaux mais en mettant leurs valeurs dans l'ordre décroissant.

Réponse :

```
1 tableauUn = np.linspace(-6,-2) # pas besoin de pré
      ciser la taille (car vaut 50 par défaut)
2 tableauDeux = np.linspace(5,10,17)
3 tableauUnRetro = np.linspace(-2,-6)  # astuce d'
      intervertir le début et la fin
4 tableauDeuxRetro = np.linspace(10,5,17)
```

$\Rightarrow$ La fonction `linspace` est pratique quand on connaît le nombre de valeurs.

Plan

Manipulations de tableaux

Les tableaux de nombres se manipulent beaucoup plus facilement que des listes de nombres, comme nous allons le voir avec cet exercice :

Exercice 25

Entrer les instructions suivantes dans la console

```
a = np.array([[1, 2, 3], [4, 5, 6]])
a+5
a-5
a*5
a/5
a*a
a**2
np.pi
np.cos(a*np.pi)
```

Manipulation de tableaux

Exercice 25

Entrer les instructions suivantes dans la console

```
a = np.array([[1, 2, 3], [4, 5, 6]])
a+5
a-5
a*5
a/5
a*a
a**2
np.pi
np.cos(a*np.pi)
```

$\Rightarrow$ Lorsqu'on applique une fonction mathématique sur un tableau, la fonction est appliquée à chaque valeur du tableau.

Plan

Plan

Motivations

Donner des valeurs aléatoires à des variables s'avère souvent utile dans de nombreux contextes (jeux, prises de décision, simulations du hasard sur des paramètres inconnus, tests statistiques...)

Le module **random** permet de générer des nombres aléatoires. Il répertorie de nombreuses *lois de probabilités* dont nous allons pouvoir nous servir.

Concernant l'importation du module `random`, la communauté n'est pas aussi unanime sur l'alias que pour le module `numpy`. Nous choisissons l'alias `rd` dans ce cours.

```
1 import random as rd
```

Plan

Entiers aléatoires

La loi de probabilité la plus simple à laquelle on peut penser sur un ensemble fini E est celle qui attribue une même probabilité à chacune de ses valeurs. On appelle cette loi la **loi uniforme**. Par exemple, si E est l'ensemble des entiers $\{5, 6, 7\}$, alors chacune des valeurs 5, 6 et 7 a une chance sur trois d'être prise.

En Python avec le module random, la méthode `randint(a,b)` génère un entier (int) entre a et b (inclus).

Lancer de dés

Exercice 26

Créer un programme qui affiche le résultat de dix lancers d'un dé.

Lancer de dés

Exercice 26

Créer un programme qui affiche le résultat de dix lancers d'un dé.

Réponse :

```
1 import random as rd
2
3 for i in range(10):
4     print(rd.randint(1,6))
```

Plan

Nombres réels aléatoires

Il existe aussi une loi uniforme sur des ensembles infinis, par exemple sur l'intervalle de réels d'extrémités a et b (où a et b sont des nombres réels).

La méthode `random()` génère un nombre décimal (float) aléatoire entre 0 et 1.
Plus généralement, la méthode `uniform(a,b)` génère un nombre décimal (float) entre deux nombres a et b.

Épreuve de Bernoulli

Exercice 27

Créer une fonction `bernoulli` qui prend comme paramètre un réel p entre 0 et 1, et qui renvoie le nombre 1 avec probabilité p et le nombre 0 sinon.

Épreuve de Bernoulli

Exercice 27

Créer une fonction `bernoulli` qui prend comme paramètre un réel p entre 0 et 1, et qui renvoie le nombre 1 avec probabilité p et le nombre 0 sinon.

Réponse :

```
1 def bernoulli(p):
2     a = rd.random()  # nombre décimal aléatoire entre
    0 et 1
3     if a<p:  # ceci arrive avec probabilité p
4         x = 1
5     else:
6         x = 0
7     return x
```

$\Rightarrow$ Cette fonction décrit une loi de probabilité très connue appelée **loi de Bernoulli**. Le nombre 1 symbolise un succès, et 0 un échec. Le cas particulier où p vaut $\frac{1}{2}$ permet de modéliser par exemple un lancer de pièce équilibrée à "pile ou face".

Plan

Échantillons aléatoires et tirages

Mélanger les éléments d'une liste peut servir dans beaucoup de situations. Pour ce faire, une possibilité est de générer des indices aléatoirement pour récupérer leurs éléments associés. Une deuxième possibilité est d'utiliser les méthodes suivantes du module `random` :

`choices(liste L, k=un nombre entier n)` donne une liste aléatoire de n tirages **avec remise** d'éléments de L

`sample(liste L, un nombre entier n)` donne une liste aléatoire de n tirages **sans remise** d'éléments de L.

⚠ L'écriture "k=" ne peut pas être enlevée dans la méthode `choices` parce que d'autres arguments peuvent être entrés (voir documentation).

Conclusion

Dans cette séance nous avons compris ce que sont des modules et avons vu quelques fonctionnalités des modules

- `numpy`
- `random`

Dans la prochaine séance nous ferons apparaître diverses représentations graphiques grâce aux modules

- `matplotlib`
- `turtle`