Université Paris 8 — Année 2021–2022

Licence 2 Mathématiques

Calcul Formel – Aide d'utilisation de `python`

31 janvier 2022

Table des matières

1 Présentation 1

2 Fonctionnalités de base 4
- 2.1 Affectation et types de données 4
- 2.2 Opérations arithmétiques 4
- 2.3 Listes 4
- 2.4 Chaînes de caractères et affichage 5
- 2.5 Boucles 6
- 2.6 Fonctions 7
- 2.7 Déclaration de listes par compréhension 8
- 2.8 Commentaires 8

3 Fonctionalités avancées 8
- 3.1 Module `math` 8
- 3.2 Bibliothèque `numpy` 9
- 3.3 Bibliothèque `matplotlib` 9

1 Présentation

Introduction. `python` est un langage de programmation multi-paradigme, de haut-niveau, interprété, et doté d'un typage dynamique. Il a été créé dans les années 1990, et sa version la plus récente[1] est `python 3.10`.

Modes d'utilisation. On peut utiliser `python` sous deux modes.

1. En **ligne de commande** (souvent couplé avec un éditeur de texte). Pour cela, tapez

```
python
```

dans un terminal. L'invite de commande débute alors par

```
>>>
```

Vous pouvez y taper des commandes, et la réponse vous est donnée directement :

```
>>> 1+1
2
```

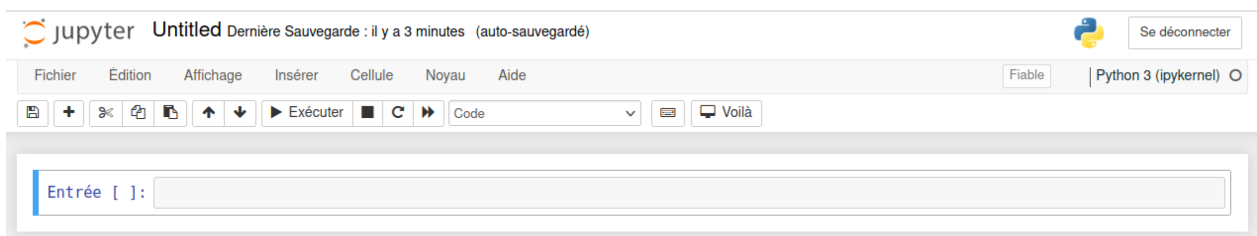
Après avoir édité un script python dans un éditeur de texte (*Notepad*, *gedit* pour les plus simples), vous pouvez aussi l'importer en ligne de commande. Par exemple, si le script est nommé `script.py`), alors on entre

1. au 19/01/2022

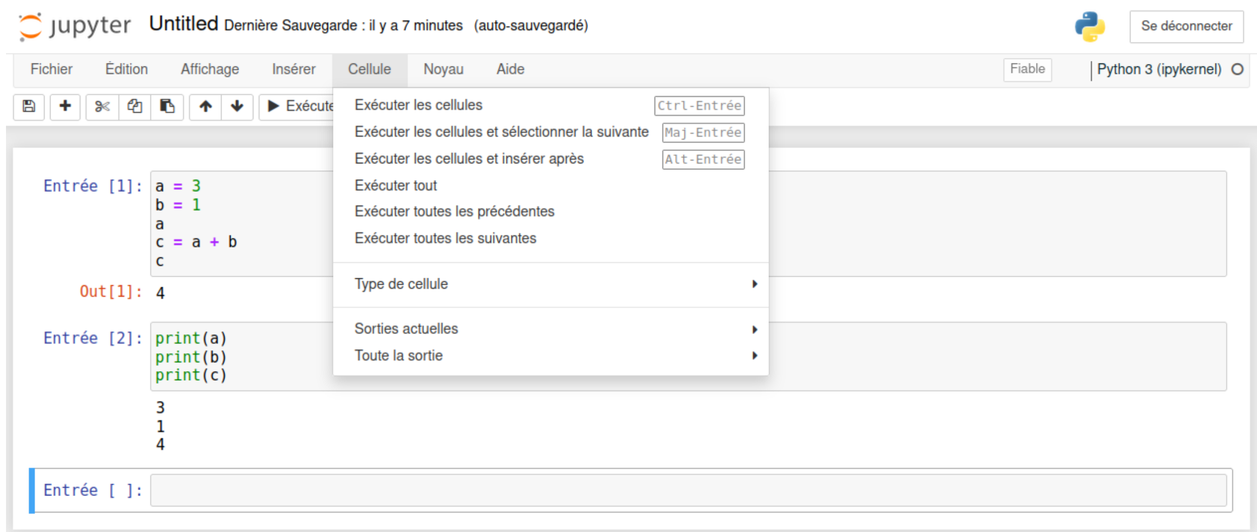
```
>>> from script import *
```

2. En mode **bloc-note**, dans un *notebook Jupyter*. Dans ce cas, une page de navigateur s'ouvre, et pouvez créer un nouveau *notebook* via l'onglet « Nouveau » à droite :

Dans l'exemple ci-dessus, on utilise la version 3 de `python`, mais le fonctionnement est similaire lorsque la version 2.7 est installée. Lorsqu'on crée un nouveau *notebook*, un nouvel onglet de navigateur s'ouvre alors :



Vous entrez vos commandes dans des *cellules*, que vous pouvez ensuite éxécuter globalement. La valeur de sortie de la dernière commande de la cellule est affichée dans une sous-cellule (via le bouton « Exécuter » de la barre de commande, ou les différentes commandes d'exécution dans le menu déroulant « Cellule »). Vous pouvez également forcer l'affichage d'autres valeurs par la commande `print()`.



Documentation. Vous trouverez un tutoriel pour `python` (versions 2.7 et 3.10) aux adresses suivantes :

`https://docs.python.org/fr/2.7/tutorial`
`https://docs.python.org/fr/3.10/tutorial`

Certains livres en français sont également disponibles en ligne. Par exemple :

— *Programmation en Python pour les sciences de la vie* par Patrick Fuchs et Pierre Poulain, à l'adresse

`https://python.sdv.univ-paris-diderot.fr`

— *Une introduction à Python 3* par Bob Cordeau et Laurent Pointal, à l'adresse

https://perso.limsi.fr/pointal/python:courspython3

— *Apprendre à programmer avec Python* par Gérard Swinnen, à l'adresse

https://www.inforef.be/swi/python.htm

Objectif du document. Vous trouverez dans la suite de ce document une aide succincte d'utilisation de python.

2 Fonctionnalités de base

2.1 Affectation et types de données

Un type de donnée est la nature d'une valeur informatique. Il définit également quelles opérations on peut lui appliquer. Le langage `python` est typé dynamiquement : c'est l'ordinateur qui affecte au mieux un type à chaque valeur, suivant l'opération qu'on lui a appliqué précédemment.

Des exemples de types sont :

— `int` pour un entier

— `float` pour un nombre flottant (attention : la séparation entre unité et dixième est notée « à l'américaine », par un point : `1.5` pour le nombre $1,5$).

— `bool` pour un booléen : `True` ou `False`

— `list` pour une liste

— ...

Contrairement à d'autres langages (comme le langage `C`), il n'est pas nécesaire de forcer le typage d'une variable. Pour déclarer et assigner à 1 une variable `x`, on utilise alors la syntaxe `x = 1`. En ligne de commande, on peut alors afficher la valeur de `x` comme suit :

```
>>> x = 1
>>> x
1
```

2.2 Opérations arithmétiques

La somme, la différence, le produit et le quotient sont respectivement notés `+`, `-`, `*` et `/`. L'ordre des opérations suit les priorités usuelles : `*` et `/` avant `+` et `-`.

Attention, si elle n'est pas exacte, la division `/` effectue une division flottante. Si l'on souhaite faire une division entière (c'est-à-dire, obtenir le quotient de la division euclidienne), il faut utiliser l'opérateur `//`. Le reste par la division euclidienne, autrement dit l'opérateur « modulo », est quand à lui obtenu avec `%`. Par exemple :

```
>>> a = 15
>>> b = 4
>>> q = a // b
>>> r = a % b
>>> a * q + r - b
0
```

L'opérateur de passage à la puissance est `**`. Pour des opérations numériques/arithmétiques plus avancées, voir la bibliothèque `math` en Section **??**.

On peut également comparer différentes valeurs :

— l'opérateur `==` effectue un test d'égalité, et `!=` un test d'inégalité ;

— les opérateurs `<` et `<=` effectuent les tests « inférieur strict » et« inférieur ou égal » ;

— les opérateurs `>` et `>=` effectuent les tests « supérieur strict » et« supérieur ou égal ».

2.3 Listes

Une liste est un collection ordonnée de valeurs pouvant avoir des types différents. Par exemple, on affecte à `L` la liste constituée de l'entier 1 et du nombre flottant $-0,5$:

```
>>> L = [1, -0.5]
>>> L
[1, -0.5]
```

Il existe une liste vide, notée `[]`.

Les indices d'une liste de n éléments sont $0, \ldots, n-1$. Pour $i \geq 0$, on peut accéder à l'élément d'indice i d'une liste `L` par `L[i]`. On peut également accéder à des sous-listes d'éléments consécutifs par :

- `L[i:]` pour obtenir la sous-liste des éléments dont les indices sont supérieurs ou égaux à i ;
- `L[:i]` pour obtenir la sous-liste des éléments dont les indices sont strictement inférieurs à i ;
- `L[i:j]` pour obtenir la sous-liste des éléments dont les indices sont compris entre i (inclus) et j (exclus).

```
>>> L = ['a', 'b', 'c', 'd', 'e']
>>> L[1]
'b'
>>> L[:3]
['a', 'b', 'c']
>>> L[1:4]
['b', 'c', 'd']
```

Une manière efficace d'ajouter un élément en fin de liste est d'utiliser la *méthode*[2] `.append()`. On l'utilise comme suit :

```
>>> L = ['a', 'b', 'c']
>>> L.append('d')
>>> L
['a', 'b', 'c', 'd']
```

Voici d'autres fonctions permettant de manipuler une liste `L` :

- `L.insert(i, x)` permet d'insérer l'élément `x` à la position d'indice `i` dans la liste `L` ;
- `y = L.pop(i)` permet de retirer l'élément d'indice `i` de la liste `L`, et de l'affecter à `y`. Si `i` n'est pas précisé, c'est le dernier élément de la liste qui est retiré.

```
>>> L = ['a', 'b', 'd']
>>> L.insert(2, 'c')
>>> L
['a', 'b', 'c', 'd']
>>> L.pop()
['a', 'b', 'c']
```

On peut concaténer deux listes par l'opérateur `+`. Pour répéter m fois une liste `L`, avec m un entier naturel, on utilise `L*m`. Si $m = 0$, on obtient la liste vide.

La longueur d'une liste est obtenue par la fonction `len()` :

```
>>> L = ['a', 'b', 'c', 'd', 'e']
>>> len(L)
5
```

Si les éléments d'une liste peuvent être comparés, on peut également accéder sa valeur maximale et minimale par les fonctions `max()` et `min()`.

2.4 Chaînes de caractères et affichage

Les chaînes de caractères sont représentées entre guillemets. Les guillemets simples ou doubles peuvent être utilisés.

```
>>> x = "calcul"
>>> y = 'calcul'
>>> x == y
True
```

On peut ....

2. une méthode est une fonction spécifique à une classe d'objets

2.5 Boucles

Il existe deux types de boucles : les boucles *for* qui font répéter une séquence d'opérations **pour** un certain ensemble de valeurs, et les boucles *while* qui font répéter la séquence **tant qu'** une certaine condition est respéctée.

Boucles *for*. Dans une boucle *for*, on parcourt successivement tous les éléments d'une structure, et pour chaque élément, on effectue la série d'instructions spécifiée au cœur de la boucle. La syntaxe des boucles *for* en `python` est la suivante :

```
>>> x = 0
>>> for i in [1, 2, 3]:
...     x = x + i
>>> x
6
```

D'un point de vue de la syntaxe, trois points sont à préciser.

— On parcourt l'ensemble des valeurs de la structure par « `for <valeur> in <structure>:` ». Il ne faut pas oublier le « `:` » (erreur courante).

— Les instructions à répéter à chaque itération sont précédées d'une indentation (tabulation ou suite de 4 espaces), représentée par ▬ dans l'exemple ci-dessus.

— il n'y a pas de mot-clé pour terminer la boucle `for`. Simplement, on retourne à la ligne (en supprimant l'indentation).

Itérateur *range*. Il existe une structure particulière qui permet de parcourir efficacement les éléments successifs d'une suite arithmétique d'entiers : elle s'invoque grâce au mot-clef `range()`. Son utilisation la plus simple est la suivante :

```
>>> for i in range(4):
...     print(i)
0
1
2
3
```

Plus précisément, la commande `range(n)` permet de parcourir les entiers de 0 à $n-1$ de manière incrémentale. Attention : c'est bien $n-1$ qui est le dernier terme parcouru.

Généralement, on peut parcourir les entiers de a (inclus) à b (non inclus) en utilisant `range(a, b)`. Enfin, si l'on souhaite parcourir une suite d'entiers qui commence par a, qui se termine avant que l'élément b soit dépassé, et qui a pour pas`nstep`, on utilise `range(a, b, step)`. Par exmeple :

```
>>> for i in range(3, 41, 10):
...     print(i)
3
13
23
33
```

Notons que cela fonctionne également avec des pas négatifs :

```
>>> for i in range(101, 76, -5):
...     print(i)
101
96
91
86
81
```

Boucles *while*. Les boucles *while* ont un fonctionnement légèrement différent des boucles *for*. Cette fois-ci, la série d'instructions que l'on spécifie dans le cœur de la boucle s'effectue **tant qu'** une certaine condition est vérifiée.

Dans l'exemple ci-dessous, on calcule la somme des entiers de 0 à 3 avec une boucle *while*. Les résultats intermédiaires du calcul sont stockés dans une variable `somme`.

```
>>> somme = 0
>>> i = 1
>>> while i < 4:
...     somme = somme + i
>>> somme
6
```

Par la suite et pour un meilleur confort de lecture, les indentations ne seront plus surlignées en bleu.

2.6 Fonctions

Une fonction représente un ensemble d'instructions éventuellement paramétrée par des valeurs en entrée (appelées *paramètres*) et qui peut également retourner une valeur en sortie.

La syntaxe `python` pour une fonction nommée `ma_fonction` est la suivante :

```
def ma_fonction(param1, param2):
    <instruction1>
    <instruction2>
    return sortie
```

où `instruction1` et `instruction2` sont des instructions qui peuvent dépendre des paramètres `param1` et `param2` donnés en entrée.

Les fonctions peuvent être appelées dans d'autres parties du code et dans d'autres fonctions. Notons également que les variables déclarées dans des fonctions sont propres à la fonction, et ne peuvent pas être utilisées en dehors.

Par exemple, pour le script suivant :

```
def perimetre_rectangle(longueur, largeur):
    p = (longueur + largeur) * 2
    return p

def perimetre_carre(cote):
    p = perimetre_rectangle(cote, cote)
    return p
```

la variable `p` n'est pas affectée après l'exécution de l'une ou l'autre des fonctions `perimetre_rectangle` et `perimetre_carre`. En revanche, on peut obtenir le périmètre d'un carré de côté 2 en effectuant

```
x = perimetre_rectangle(2)
```

Notons qu'il n'est pas nécessaire de spécifier le type des paramètres d'une fonction. Par exemple, on peut exécuter aussi bien `perimetre_carre(2)` que `perimetre_carre(1.5)`.

Il est également possible qu'une fonction s'appelle elle-même. Ce type de fonction est nommé *fonction récursive*, par opposition aux *fonctions itératives*. Voici un exemple de fonction récursive qui calcule le n-ème terme d'une suite arithmétique de premier terme `init` et de raison `raison` :

```
def suite_arithmetique(init, raison, n):
    if n == 0:
        return init
    return raison + suite_arithmetique(init, raison, n-1)
```

2.7 Déclaration de listes par compréhension

Le langage python permet de créer des listes avec une syntaxe proche du langage mathématique. Par exemple, supposons que l'on souhaite créer la liste des carrés des entiers compris entre 1 et 7. Mathématiquement, l'ensemble des éléments de cette liste peut être décrit comme :

$$\{x^2 \mid x \in [1,7]\}.$$

En python, la liste pourra être créée de la sorte :

```
L = [ x**2 for x in range(1, 8) ]
```

On appelle ce type de déclaration de listes une déclaration **par compréhension**. Formellement, la syntaxe est la suivante :

```
[ <fonction de x> for x in <structure> ]
```

où `<fonction de x>` est une fonction de la variable x qui parcourt la structure `<structure>`.

Il est également possible de conditionner l'ajout de l'élement dans une liste. Par exemple, pou sélectionner seulement les chaînes de caracètes de longueur égale à 7 dans la liste `L = ["bonjour", "maison", "ecole",  "voiture", "telephone"]`, on effectuera la série d'instructions suivantes :

```
L = ["bonjour", "maison", "ecole",  "voiture", "telephone"]
M = [ x for x in L if len(x) == 7]
```

Formellement, la syntaxe est donc la suivante :

```
[ <fonction de x> for x in <structure> if <condition sur x> ]
```

2.8 Commentaires

Le symbole # indique à python de ne pas exécuter ce qui suit le symbole sur la ligne. Par exemple, les deux lignes de codes suivantes auront la même exécution :

```
x = 2
x = 2  # + 3
```

Cela permet d'ajouter des commentaires à votre code :

```
x = 2
y = 3
z = x + y    # on effectue la somme de x et y
```

ou encore de momentanément empêcher une ligne de s'exécuter :

```
x = 2
y = 3
# y = 4
z = x + y
```

(dans ce dernier exemple, la variable y vaut 3 et la variable z vaut 5).

3 Fonctionalités avancées

3.1 Module math

Le module math permet d'ajouter des fonctionalités mathématiques de base à celle préexistantes dans python. En voici une liste non-exhaustive :

Pour plus de détails, voir :

https://docs.python.org/3/library/math.html

commande/code	description
`math.pi`	retourne une approximation flottante de π
`math.exp(x)`	retourne e^x
`math.log(x [, b])`	retourne $\ln(x)$ si b n'est pas spécifié, et $\log_b(x)$ sinon
`math.sqrt(x)`	retourne $\sqrt{x}$
`math.cos(x)`	retourne $\cos x$
`math.sin(x)`	retourne $\sin x$
`math.tan(x)`	retourne $\tan x$
`math.erf(x)`	retourne la fonction d'erreur en x, c'est-à-dire $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{dt}$

Table 1 – Quelques variables et fonctions du module `math`

3.2 Bibliothèque `numpy`

3.3 Bibliothèque `matplotlib`

...

Elle peut être couplée avec la bibliothèque `numpy`.

Pour plus de détails, voir :

`https://matplotlib.org`