Le langage Python

Pierre Cladé

Apr 24, 2024

1 Feuilles de cours

1.1 Les fonctions

Les fonctions sont des sous-programme que l'on peut exécuter. Elles sont en particulier utilisées pour effectuer des tâches répétitives.

Utiliser les fonctions

Il existe un grand nombre de fonctions déjà définies en Python. Certaines sont des fonctions natives, disponibles directement en Python (par exemple la fonction `print`). D'autres sont dans une librairie, par exemples, les fonctions mathématiques sont dans la librairie `math`.

Pour exécuter une fonction (on utilise aussi le mot appeler - call en anglais), il faut faire suivre le nom de la fonction par des parenthèses avec à l'intérieur les arguments séparés par des virgules.

Une fonction peut avoir zero argument, un ou plusieurs. Le nombre d'arguments n'est pas forcement fixé à l'avance.

```
list() # Fonction sans arguments

from math import cos # on importe la fonction cosinus
cos(1) # Cette fonction possède un seul argument

print('Bonjour', 'Hello') # La fonction print peut avoir autant d'argument qu'on le␣
↪souhaite
```

Lorsqu'une fonction possède plusieurs arguments, il est important de respecter l'ordre. Lorsque l'on ne connait pas cet ordre, il faut regarder la documentation, ce qui peut se faire avec la commande `nom_de_la_fonction?` ou `help(nom_de_la_fonction)`.

Regardons par exemple la fonction date du module datetime. Cette fonction permet de manipuler des dates avec Python.

```
from datetime import date
#date?
```

La documentation nous donne ;

```
date(year, month, day)
```

Cette fonction nécéssite donc 3 arguments (l'année, le mois et le jour).

On peut l'utiliser de la façon suivante :

```
bataille_marignan = date(1515, 9, 13)
print(bataille_marignan)
```

Pour lever l'ambiguité lorsqu'il y a plusieurs arguments, il est possible de nommer explicitement ceux-ci

```
bataille_marignan = date(year=1515, month=9, day=13)
```

Lorsque les arguments sont nommés, l'ordre n'a plus d'importance.

```
assert date(year=1515, month=9, day=13)==date(day=13, month=9, year=1515)
```

Attention, Python ne peut pas deviner ne nom de l'argument à partir du nom de la variable. Ceci ne fonctionnera pas :

```
year=1515
month=9
day=13

date(day, month, year)
```

Lorsqu'une fonction contient beaucoup d'arguments, il peut être utile de regrouper les variables dans une seule variable. Il peut s'agir soit d'une liste ou d'un n-uplet (tuple) auquel cas l'ordre sera important, soit d'un dictionnaire, auquel cas, les clés doivent correspondre aux noms des variables. Pour une liste on précéde l'objet d'une `*` pour un dictionnaire de `**`.

```
date_tpl = (1515, 9, 13)
date_dct = {"year":1515, 'month':9, "day":13}

print(date(*date_tpl))
print(date(**date_dct))
```

Création d'une fonction

Principe général

Le mot clé `def` est utilisé pour créer une fonction. Il doit être suivit du nom de la fonction, des arguments placés entre parenthèse. Comme toute instruction qui sera suivi d'un bloc d'instruction, il faut terminer la ligne par un `:`. Le bloc d'instruction sera alors indenté.

```
# Fonction sans argument
def affiche_bonjour():
    print('Bonjour tout le monde')
    print('Hello World!')

affiche_bonjour()
```

Pour renvoyer une valeur, il faut utiliser l'instruction `return`.

```
from math import pi

def surface_d_un_disque(r):
    return pi*r**2

surface_d_un_disque(3)
```

Python quitte la fonction imédiatement après le return. Il peut y avoir plusieurs return dans une fonction. Python quitte la fonction après le premier return executé. Si il arrive à la fin de la fonction, alors il y a un return implicite. La valeur renvoyée est `None`.

Dans cet exemple, le `print('B')` ne sera jamais executé

```
def f():
    print('A')
    return
    print('B')

f()
```

```
def f(x):
    a = x**2
    # Il n'y a pas de return. Cette fonction ne sert donc à rien

print(f(1))
```

```
from math import sin, cos, tan

def f(x, case):
    if case=='A':
        return sin(x)
    if case=='B':
        return cos(x)
    if case=='C':
        return tan(x)
```

Une fonction peut renvoyer plusieurs valeurs. Pour cela on les sépare par des virgules. On peut récupérer les valeurs en séparant les variables par des virgules à gauche du signe =.

```
from math import cos, sin

def coordonnees(r, theta):
    return r*cos(theta), r*sin(theta)

x, y = coordonnees(1, 3)
```

Chaîne de documentation

Si la première ligne du bloc d'instruction d'une fonction est une chaîne de caractère litérale, alors cette chaîne sera la chaîne de documentation de la fonction. Cette chaîne correspond à la description de la fonction.

```
def surface_d_un_disque(r):
    "Calcule la surface d'un disque de rayon r"
    return pi*r**2

help(surface_d_un_disque)
```

Lorsque la chaîne de caractère prend plusieurs lignes (ce qui est en générale le cas…), alors il est préférable d'utiliser une chaîne avec triple guillemets.

```
def surface_d_un_disque(r):
    """Calcule la surface d'un disque

    Utilise simplement la formule $\pi r^2$

    Arguments :
```

(continues on next page)

(continued from previous page)

```
        r (float) : rayon du disque

    Renvoie :
        float : le surface du disque
    """
    return pi*r**2

help(surface_d_un_disque)
```

La chaîne de documentation est différente des commentaires (#): ces derniers ne sont pas disponibles aux utilsateurs, mais seulement à celui qui lira et voudra comprendre le code.

Variables locales

Une variable est locale lorsqu'elle n'est définie que à l'intérieur d'une fonction. Une variable est globale lorsque sa definition est à l'extérieure de la fonction. Il n'y a pas d'interaction entre une variable locale et une variable portant le même nom à l'extérieur de la fonction

Toutes les variables qui sont affectées dans la fonction (i.e. pour lesquels on a une ligne `variable=...`) ainsi que les arguments de la fonction sont des variables locales.

```
x = 1
def f(): # x est une variable globale
    print(x)

f()
```

```
x = 1
def f(): # x est une variable locale
    x = 3
    print(x)

print(x)
f()
print(x)
```

La valeur prise par une variable globale est la valeur au moment de l'exection de la fonction et non au moment de la définition de la fonction

```
#### NE FONCTIONNE PAS ###
coef = 2
def double(x):
    return coef*x

coef = 3
def triple(x):
    return coef*x

print(double(3))
print(triple(3))
```

```
9
9
```

Arguments optionels

Il est possible de donner une valeur par défaut à un argument. Celui-ci sera alors optionnel. Cela se fait avec la syntaxe `def f(.., arg=default_value)`. Les arguments optionels doivent être définis à la fin de la liste des arguments.

```
def f(a, b=1):
    print(a*b**2)

f(1)
f(1, 5)
f(b=4, a=1)
```

Fonction avec un nombre arbitraire d'arguments

Il est possible de définir une fonction avec un nombre arbitraire d'arguments en utilisant la syntaxe `*arg`. Dans ce cas, la variable arg sera un n-uplet (tuple) qui contiendra tous les arguments suplémentaires.

```
def mafonction(a, b, *args):
    print('a = ', a)
    print('b = ', b)
    print('args =', args)

mafonction(1, 2, 3, 4)
```

Il est aussi possible de définir une fonction avec un nombre arbitraire d'arguments nommés. Dans ce cas, il faut utiliser la syntaxe `**kwd`. La variable `kwd` est alors un dictionnaire dont les clés sont les noms des variables.

```
def mafonction(a, b, **kwd):
    print('a = ', a)
    print('b = ', b)
    print('kwd = ', kwd)

# mafonction(1, 2, 3, 4) # erreur
mafonction(1, 2, alpha=1, beta=3)
```

1.2 Les nombres

Les entiers

Type `int` en Python.

Il existe plusieurs façons d'entrer un entier sous forme litérale

```
a = 5 # Décimal
a = 0b1001 # binaire
a = 0x23 # hexadécimal
```

En Python la taille des entiers est illimitée. Par exemple:

```
print(3**100)
```

```
515377520732011331036461129765621272702107522001
```

Attention, ce n'est plus le cas lorsque l'on utilise des librairies de calcul numérique (comme numpy ou pandas). Dans ce cas, les nombres sont enregistrés sous une taille finie. Par défaut, il s'agit de nombre enregistré avec 64 bits, en tenant compte du signe, les entiers sont alors compris entre -2^63 et 2^63 (exclu)

Attention, lorsqu'il y a un débordement (overflow), il n'y a pas d'erreur et le comportement est inattendu.

```
import numpy as np

a = np.array([3])
a**100
```

```
array([-2984622845537545263])
```

Lorsqu'un nombre est enregistré sous un format de taille finie, il faut s'imaginer qu'il fonctionne comme une calculatrice dont ont aurait caché les premiers chiffres. Nous allons raisonner en décimal, mais dans la réalité ce sont des bits qui sont manipulés.

Si on ne regarde que les trois derniers chiffres (resp. 64 bits) alors les opérations sont faire modulo 1000 (resp. module 2^64). Par exemple $50 \times 50 = 2500 = 500$. C'est ce que l'on appel un débordement (overflow).

Les nombres négatifs sont enregistré en utilisant une astuce : regardons l'opération (modulo 1000) suivante : $997+3 = 0$. Le nombre 997 est donc le nombre qui lorsqu'on lui rajoute 3 donne 0, c'est donc -3. Cela explique pourquoi dans l'exemple précédent on obtient un nombre négatif.

Les nombres à virgule flottante

Type `float`. Il existe plusieurs façon d'entrer un entier sous forme litérale : soit en mettant explicitement un `.` décimal, soit en utilisant le `e` de la notation scientifique

```
a = 1234.567
c = 3e8 # ou 3E8 soit 3 fois 10 à la puissance 8
```

Attention, le comportement d'un nombre à virgule flottante est différent de celui d'un entier, même lorsqu'il représente un entier

```
a = 3
b = 3.
print(a**100)
print(b**100)
```

```
515377520732011331036461129765621272702107522001
5.153775207320113e+47
```

Les nombres sont enregistrés en double précision, sur 64 bits. Il sont enregistrés sous la forme $s \times m \times 2^e$ où s est le signe (± 1 sur un bit), m la mantisse, un nombre entre 0 et 1 sous la forme $0.xxxxx$ avec en tout 52 bits, et e l'exposant, un nombre entier signé sur 11 bits (soit entre -1024 et 1023).

Attention, la précision des nombre à virgule flottante est limitée. Elle vaut 2^{-52}, soit environ 10^{-16}

```
a = 3.14
print(a == a + 1E-15)
print(a == a + 1E-16)
```

```
False
True
```

Les nombres complexes

Type `complex`

Il sont toujours enregistrés sous la forme de deux nombres à virgules flottantes (partie réelle et partie imaginaire). Il faut utiliser le `J` ou `j` pour écrire un nombre complexe sous forme litérale

```
a = 1 + 3j
a = 1.123j
```

Il faut forcement précéder le `j` d'un nombre. Le symbole `j` seul désignant une variable. Notons que si il est possible de placer des chiffres dans le nom d'un variable (par exemple `x1`), il n'est pas possible de commencer une variable par un chiffre. Par exemple `j1` pourra désigner une variable mais pas `1j`.

On peut facilement accéder à la partie réelle et imaginaire des nombres complexe, ce sont des attributs du nombre

```
a = 1 + 3J
print(a.real)
print(a.imag)
```

```
1.0
3.0
```

Opérations sur le nombres

Les opérations sur les nombres sont les suivantes :

- somme : `+`
- produit : `*`
- différence ou négation : `-`
- division : `/`
- division entière : `//`
- modulo (reste de la division euclidienne) : `%` (par exemple `7%2`)
- puissance : `**` (par exemple `2**10`)

Les booléens et comparaison

Il existe deux valeurs : `True` et `False` (attention à la casse).

Les comparaisons se font à l'aide des symboles <, <=, ==, > et >=. Pour savoir si deux valeurs sont différentes, on utilise `!=`.

Les opérations sont par ordre de priorité : `not`, `and` et `or`.

```
print(False and False or True)
print(False and (False or True))
```

```
True
False
```

Les opérations `and` et `or` effectuent en fait un test conditionnel. L'instruction `A and B` est interprétée comme `B if not A else A`, de même `A or B` équivaut à `A if A else B`.

```
from math import sqrt

x = -1
#if sqrt(x)>.2:
#         print('Hello')

if x>0:
    if sqrt(x)>.2:
        print('Hello')

if (x>0) and (sqrt(x)>.2):
    print('Hello')
```

Warning: Les symboles & et | sont des opérateurs binaires. Ils réalisent les opérations and et or sur les entiers bit par bit en binaire (par exemple 6 & 5 donne 4). Il ne faut pas les utiliser pour les opérations sur des booléens. Ils ont aussi une priorité sur les comparaisons

```
#if (x>0) & (sqrt(x)>0):
#    print('Hello')
```

```
x = 3

if x==7 & x==3:
    print('Bonjour')

if x==7 and x==3:
    print('Hello')
```

```
Bonjour
```

Conclusion : il est préférable de toujours mettre des parenthèses….

1.3 Les conteneurs en Python

Notions globales

On appelle conteneur (container) un objet ayant vocation à en contenir d'autres

Il existe plusieurs types de conteneurs :

- liste
- dictionnaire
- ensemble
- n-uplet

Dans un certaine mesure, on peut aussi considérer les chaînes de caractères comme des conteneurs

Voici quelque exemples :

```
s = "Bonjour" # chaîne de caractère
l = [1, 2, "bonjour", [1, 2]] # liste
d = {'key1':123.45, 3:"bonjour"} # Dictionnaire
e = {1, 2, 4} # ensemble
t = (1, 2, [1, 2]) # n-uplet / tuple
```

L'opérateur in

Il permet de tester si un le conteneur contient un objet donné.

```
print(1 in l)
print(3 in d) # Pour les dictionnaires, c'est la clé
print('on' in s) # Pour les chaînes de caratères, n'importe quelle sous-chaîne
```

```
True
True
True
```

Longueur

Un autre point commun partagé par de nombreux conteneurs est qu'ils possèdent une taille. C'est-à-dire qu'ils contiennent un nombre fini et connu d'éléments, et peuvent être passés en paramètre à la fonction len.

```
print(len(t))
```

```
3
```

Objet subscriptables

Cela désigne les objets sur lesquels l'opérateur [] peut être utilisé. L'ensemble des types cités sont subscriptables, à l'exception de l'ensemble (set), qui n'implémente pas l'opération []

Parmis les objets subscriptables, on distingues ceux qui sont indexables par un entier et ceux qui sont sliceables.

Les dictionnaires ne sont pas indexables. Les liste et les chaînes de caractères sont indexable et sliceables.

```
print(d["key1"])

print(l[2])
print(s[1])
```

```
123.45
bonjour
o
```

Pour les objets indexables, on rappel que le premier élément est l'élément 0. Le dernier est dont n-1 ou n est la taille de l'objet.

Les slices permettent de récupérer une partie de l'objet initial. La syntaxe est `[start:stop:step]`. Si on omet step, alors le pas est de 1. Si on omet stop, alors il s'agit de n, si on omet start, il s'agit de 0.

La taille de l'objet renvoyé est (stop - start)//step .

ATTENTION : si on indexe avec [i:j], alors le dernier élément est j-1

```
print(s[1:4])
print(l[1:2])
print(s[::2])
```

```
onj
[2]
Bnor
```

Les indices négatifs, sont pris modulo la taille du conteneur.

```
print(l[:-1]) # Tous les éléments sauf le dernier
```

Conteneurs modifiables

Les listes, les dictionnaires et les ensembles sont modifiables. Les n-uplet et les chaines de caratères ne le sont pas.

Par modifiable, on entent par exemple que l'on peut rajouter, suprimer ou remplacer un élément.

```
liste1 = ['Bonjour']
liste1.append('Hello')
liste1.insert(1, 'Salut')
del liste1[0]
liste1[1] = 'Coucou'
liste1
```

```
['Salut', 'Coucou']
```

Une liste, c'est comme un classeur, on peut rajouter ou suprimer des feuilles. Ce sera toujours le même classeur. Un objet non modifiable ne possède par la méthode append, insert. On ne peut pas faire objet[i] = qqc. C'est comme un livre, il n'est pas possible de le modifier. La seule chose que l'on peut faire, c'est imprimer un nouveau livre avec une modification.

Si vous acheter deux livres identiques, il seront toujours identiques. Si vous achetez deux classeurs identiques, leur contenu pourra être différent.

En Python, la plupart des objets sont modifiables. Les exceptions sont les nombres, les n-uplets et les chaines de caractères.

Attention : lorsque l'on passe un objet à une fonction, il n'est pas dupliqué. Si la fonction modifie l'objet, alors l'objet est modifié.

```
# Dans cet exemple, il n'y a qu'une seule liste
def f(c):
    c.append(3)

a = [1, 2]
b = a
f(b)
print(a)
```

```
[1, 2, 3]
```

Conteneurs iterables

C'est le cas des conteneurs que l'on peut utiliser dans un boucle for. Tous les conteneurs ci dessus sont iterables. Dans la mesure du possible, il est important de faire la boucle for directement sur l'objet, plutôt que par exemple sur ses indices.

```
for lettre in s:
    print(lettre)

for item in l:
    print(item)
```

```
B
o
n
j
o
u
r
1
2
bonjour
[1, 2]
```

Pour les dictionnaires, il est possible d'itérer sur les clés, les valeurs, ou les deux:

```
for key in d: # On peut utiliser d.keys()
    print(key)

for val in d.values():
    print(val)

for key, val in d.items():
    print(key, val)
```

```
key1
3
123.45
bonjour
key1 123.45
3 bonjour
```

Si on souhaite parcourir une liste et avoir l'indice, il est possible d'utiliser la fonction enumerate:

```
for i, item in enumerate(l):
    print(f"L'item numero {i} est {item}")
```

```
L'item numero 0 est 1
L'item numero 1 est 2
L'item numero 2 est bonjour
L'item numero 3 est [1, 2]
```

Si on souhaite parcourir deux listes en même temps, on peut utiliser la fonction zip

```
liste1 = ['A', 'B', 'C']
liste2 = [10, 4, 24]
for lettre, nombre in zip(liste1, liste2):
    print(lettre, nombre)
```

```
A 10
B 4
C 24
```

List comprehension

Il arrive fréquement que l'on souhaite créer un conteneur à partir d'un autre. Par exemple, on a une liste et on souhaite appliquer une fonction sur tous les élements. On souhaite filter un dictionnaire, …

Un façon simple consiste à créer un conteneur vide et ensuite le remplir au fur et à mesure :

```
ancienne_liste = [1, 4, 6, 3]
nouvelle_liste = []
for val in ancienne_liste:
    nouvelle_liste.append(val/2)
```

La technique de list comprehension permet de le faire en une seule ligne

```
nouvelle_liset = [val**2 for val in ancienne_liste]
```

Cette methode fonctionne aussi pour les dictionnaires ou les ensembles

```
liste2 = ["bonjour", "hello"]
print({val:i for i, val in enumerate(liste2)})

{i**2 for i in range(-5, 5)}
```

```
{'bonjour': 0, 'hello': 1}
```

```
{0, 1, 4, 9, 16, 25}
```

Il est possible en plus de filtrer une liste

```
[i for i in range(20) if i%2==0 and i%3!=1]
```

```
[0, 2, 6, 8, 12, 14, 18]
```

Les différents types de conteneurs

Les listes

Pour créer une liste, on utiliser les []. Il est aussi possible de créer une liste à partir d'un objet itérable.

```
l = [1, 2]
l = list('Bonjour')
```

```
['B', 'o', 'n', 'j', 'o', 'u', 'r']
```

Voici quelques méthodes et fonctions :

- append : rajoute un élément à la fin
- insert : rajoute un élément à la position i

- extend : étend la liste en rajoutant les éléments d'un autre liste
- l1 + l2 : crée une nouvelle liste en concaténant les deux listes.
- sort : modifie la liste en la triant (la fonction sorted renvoie une nouvelle liste)
- index : trouve l'indice d'un élément (ou renvoie une ValueError si il n'existe pas)
- count : compte le nombre d'élément ayant la valeur donnée en argument
- pop : suprime et renvoie le dernier élément

Les dictionnaires

Pour créer un dictionnaire :

```
# ces trois dictionnaires sont identiques
d = {'key1':'Bonjour', 'key2':'Hello'}
d = dict(key1="Bonjour", key2="Hello")
l = [('key1', 'Bonjour'), ('key2', 'Hello')]
d = dict(l)
```

Quelques méthodes:

- keys, values, items : renvoie une 'liste' (en fait ce n'est pas vraiment une liste) sur laquelle on peut faire une boucle for (voir ci-dessus)
- get : récupère une clé, l'intérêt est la possibilité d'utiliser une valeur par défaut si la clé n'existe pas
- setdefault : défini une valeur si celle ci n'existe pas
- update : modifie le dictionnaire à partir d'un nouveau dictionnaire. Il n'est pas possible de faire un '+' ente deux dictionnaires

Remarques sur les clés : souvent les clés sont des chaînes de caractères, mais ce n'est pas obligatoire. On peut utiliser n'importe quel objet non modifiable ne contenant pas d'objet modifiable: nombre, chaine de caractère ou tuple contenant des objets non modifiables.

Les ensembles

Correspond à la notion mathématique. Il ne peuvent contenir deux objets identiques. Ils sont rarement utilisés, mais pratique lorsque l'on en a besoin. On peut aussi créer un ensemble à partir de n'importe quel objet itérable

```
s = {1, 5}
s = set(range(3))
```

Opérations sur les ensembles :

- `&` : intersection
- `|` : union
- `-` : difference
- `^` : difference symétrique

```
s1 = {1, 2, 3, 4}
s2 = {2, 3, 4, 5}
```

(continues on next page)

(continued from previous page)

```
print(s1 & s2)
print(s1 | s2)
print(s1 - s2)
print(s1 ^ s2)
```

```
{2, 3, 4}
{1, 2, 3, 4, 5}
{1}
{1, 5}
```

Les n-uplets

Similaires aux listes, il ne sont pas modofiables, ce qui permet de les utiliser comme clé dans un dictionnaires. Les n-uplets (tuple en anglais) sont aussi utilisé lorsqu'un fonction renvoie plusieurs éléments.

Il sont créé avec des (). Attention aux cas particulier du 1-uplet

```
t = ()
t = (1,) # (1) n'est pas un 1-uplet, mais juste le nombre 1
t = (1, 2, 56)
```

Seules les méthode count et index existent (et font la même chose que pour une liste). Le + permet la concaténation

1.4 Chaînes de caractères

Création d'une chaine

On peut les créer avec des `'` ou `"`. Ces caractères servent à délimiter les début et la fin du texte de la chaîne de caractère. Les guillemets simples `'` et doubles `"` sont équivalents. On pourra choisir l'un ou l'autre. Il sera cependant judicieux, si une chaîne de caractère doit contenir un de ces guillemets, d'utiliser l'autre pour le début et la fin de la chaîne.

```
s = "Bonjour"
s = 'Bonjour'
s = "Aujourd'hui"
```

Pour créer une chaîne de caractère sur plus d'une ligne on utilise `'''` ou `"""`

```
s = """Bonjour,
Comment allez-vous ?"""
```

Les **caractères spéciaux** sont les caractères qui ne sont pas affichables et en tant que tel. Par exemple, il existe un caractère pour le retour à la ligne. Il est possible d'utiliser ce caractère dans une chaîne en utilisant `\n`. L'antislash sert ici de caractère d'échappement pour indiquer que l'on va entrer un caractère spécial. La lettre `n` indique ici qu'il s'agit d'un retour à la ligne.

Dans les exemples suivants, le retour à la ligne est un caractère. On peut le créer en utilisant `\n`.

```
s = """a
b"""
print(len(s))
s2 = "a\nb"
assert s==s2
```

```
3
```

L'antislash sert aussi à insérer un guillemet dans une chaîne :

```
s = 'Aujourd\'hui'
```

Manipulation des chaînes de caractères

Comme tout conteneur indexable, il est possible d'accéder à chaque caractère d'une chaîne ou à une partie d'une chaîne. La longueur de la chaîne s'obtient avec la fonction `len`. On peut aussi faire une boucle `for` sur chacun des éléments de la chaîne.

```
s = "Pierre"
print(s[0])
print(s[2:4])
```

```
P
er
```

Cependant, il n'est pas possible de modifier une chaîne de caractères (l'opération `s[0]='p'` échoue).

L'opérateur + permet de concaténer des chaînes de caractères. L'opérateur * permet de répeter `n` fois la même chaîne de caractère

```
s1 = "Bonjour"
s2 = 'tout le monde'
print(s1 + ' ' + s2)
```

```
Bonjour tout le monde
```

```
print('ha!'*10)
```

```
ha!ha!ha!ha!ha!ha!ha!ha!ha!ha!
```

Formatage des chaînes de caractère

Le formatage d'une chaîne de caractère consiste à mettre dans une chaine un élément variable. Cette opération est souvent utilisée lorsque l'on veut afficher proprement un résultat

```
heure = 15
minute = 30
"Il est {0}h{1}".format(heure, minute)
```

```
'Il est 15h30'
```

Pour insérer un élément ou plusieurs éléments variables dans une chaîne de caractère, on crée d'abord cette chaîne en mettant à la place des ces éléments une accolade avec un numéro d'ordre `{i}`. En appliquant la méthode `format` sur cette chaîne, les accolades seront remplacées par le ième argument.

Il est possible de passer l'argument par nom dans ce cas la clé est le nom de l'argument.

```
"Il est {heure}h{minute}".format(heure=heure, minute=minute)
```

```
'Il est 15h30'
```

Depuis la version 3.6 de Python, il est possible de demander à Python d'utiliser automatiquement les variables locales à l'aide du préfix `f`.

```
f"Il est {heure}h{minute}"
```

```
'Il est 15h30'
```

Il est aussi possible de demander d'utiliser un attribut d'un objet :

```
z = 1 + 2J
print(f'Re(z) = {z.real}')
```

```
Re(z) = 1.0
```

En utilisant le formatage de chaîne de caractère, il est possible de spécifier en détail comment ce nombre doit s'afficher. Par exemple, si il s'agit d'un nombre à virgule flottante, combien de décimales faut-il afficher, faut il utiliser la notation scientifique, etc. Pour cela, on rajoute à l'intérieur des accolades un code particulier. Ce code est précédé du signe ':'.

```
from math import pi
'{0:.5f}'.format(pi)
c = 299792458. # Vitesse de la lumière en m/s
'c = {0:.3e} m/s'.format(c)
```

```
'c = 2.998e+08 m/s'
```

Le 'f' indique que l'on veut une notation a virgule fixe, le 'e' une notation scientifique. Le chiffre que l'on indique après le '.' donne le nombre de chiffre après la virgule que l'on souhaite.

L'aide en ligne de Python fournit d'autres exemples et des détails.

Quelques méthodes utiles

- `strip` (enlève les espaces blancs au début et fin de la chaîne)
- `split` (coupe la chaine et renvoie une liste de chaîne)
- `join` (inverse de slit : rassemble une liste de chaîne avec un chaîne)
- `startswith`, `endswith`
- `lower`, `upper` (convertit en minuscule ou majuscule)
- `replace`

```
s="     bonjour   "
print(s)
print(s.strip())

s='un deux trois'
print(s.split())

l = ['pomme', 'pêche', 'poire', 'abricot']
s = ', '.join(l)
print(s)
```

```
     bonjour
bonjour
['un', 'deux', 'trois']
pomme, pêche, poire, abricot
```

Unicode

Unicode est un standard informatique qui permet des échanges de textes dans différentes langues, à un niveau mondial. Il vise au codage de texte écrit en donnant à tout caractère de n'importe quel système d'écriture un nom et un identifiant numérique, et ce de manière unifiée, quelle que soit la plateforme informatique ou le logiciel utilisé.

On peut créer des chaînes directement en unicode. On peut aussi utiliser le code en hexadecimal.

```
s1 = "Rayon γ"
s2 = "Rayon \u03B3"
print(s2)
assert s1==s2
```

```
Rayon γ
```

Il est possible de convertir un caractère en nombre et vice-versa

```
print(ord('€'))
print(hex(ord('€')))
```

```
8364
0x20ac
```

```
' '.join([chr(97 + i) for i in range(26)])
```

```
'a b c d e f g h i j k l m n o p q r s t u v w x y z'
```

On trouve même des émoticones

```
s = "\U0001f600"
print(s)
```

```
😀
```

On peut aussi utilier le nom unicode

```
print("\N{slightly smiling face}")
```

```
🙂
```

1.5 Programmation orientée objet

Comment créer se propres objets en Python.

Vocabulaire

En python tout ce que l'on manipule est un objet :

- Nombre, liste, dictionnaire, …
- Tableau numpy
- Fonction, module, ….

Un objet possède des **attributs**. Exemple :

```
z = 1.1J
z.real
a = np.array([1, 2, 4])
a.shape
```

Un objet possède des **méthodes**. Exemple :

```
a.mean()
a.conjugate()
z.conjugate()
```

Les méthodes sont des fonctions attachées à l'objet. La méthode conjugate d'un tableau n'est pas la même fonction que la méthode conjugate d'un complexe.

Comment créer un objet

Créer un object vecteur

```
from math import sqrt

class Vecteur():
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __repr__(self):
        return f'Vecteur({self.x}, {self.y}, {self.z})'

    def norme(self):
        return sqrt(self.x**2 + self.y**2 + self.z**2)

    def __add__(self, other):
        if isinstance(other, Vecteur):
            return Vecteur(self.x + other.x, self.y+other.y, self.z+other.z)
        return NotImplemented


v1 = Vecteur(1, 2, 3)
```

(continues on next page)

(continued from previous page)

```
v2 = Vecteur(3, 5, 1)
v1 + v2
```

```
Vecteur(4, 7, 4)
```

Attributs d'objet et de class

Les classes sont des objets comme les autres. Elles ont des attributs. Ce sont tous les objets définis dans la classe. Si un objet ne possède pas l'attribut auquel ou souhaite accéder, c'est l'attribut de la classe qui est renvoyée si celui ci existe.

```
class Test():
    a = 1

t = Test()
print(t.a) # Ici on va chercher l'attribut de Test
t.a = 2 # A est maintenant un attribut de l'objet
print(Test.a) # Test possède toujours sont attribut a
```

```
1
1
```

Les méthodes sont des fonctions qui sont des attributs de la classe. C'est lorsque l'on accède à cette attribut par l'objet, une transformation se passe et Python rajoute automatiquement l'objet comme premier argument.

Instance et héritage

L'héritage est un mécanisme qui permet, lors de la déclaration d'une nouvelle classe, d'y inclure les caractéristiques d'une autre classe.

Un objet est une instance de sa classe. On peut le tester avec la méthode `isinstance`. Il est aussi une instance de toutes les classes dont hérite sa classe.

```
import math

class FormeGeometrique:
    pass

class Polygone(FormeGeometrique):
    pass

class Cercle(FormeGeometrique):
    def __init__(self, r):
        self.r = r

    def surface(self):
        return math.pi*r**2

class Rectangle(Polygone):
    def __init__(self, largeur, longueur):
        self.largeur = largeur
        self.longueur = longueur
```

(continues on next page)

(continued from previous page)

```
    def surface(self):
        return self.largeur * self.longueur

objet = Rectangle(2, 3)
print(objet.surface())

isinstance(rect, Polygone)

def volume_prisme(base, hauteur):
    return base.surface()*hauteur
```

```
6
```

```
True
```

L'utilsation de classes permet d'écrire des méthodes ayant le même nom, mais différentes pour chaque objet. La fonction volume_prisme fonctionne pour tout type de FormeGeometrique possédant une méthode surface.

Méthodes spéciales

- `__init__`
- `__repr__`, `__str__`

Lorsque c'est possible, `__repr__` doit représenter au mieux l'objet. Souvent il s'agit d'une chaine de caractère qui en étant évaluée renvoie un objet similaire. La méthode `__str__` renvoie `__repr__` par défaut. Sinon, elle doit être plus simple.

Opérateur unaire et binaire

- `__neg__`
- `__add__`, `__sub__`, `__mul__`, `__truediv__`, `__mod__`, `__pow__`
- `__radd__`, ...
- `__eq__` (==), `__ne__` (!=), `__lt__` (<), `__le__` (<=), `__gt__`, `__ge__`
- `__or__` (|), `__and__` (&), `__xor__`(^)

Emulateur de contenu

- a[key] => `a.__getitem__(key)`
- a[key] = val => `a.__setitem__(key, val)`
- del a[key] => `a.__delitem__(key)`
- len(a) => `a.__len__()`
- for elm in a => `for elm in a.__iter__()`

1.6 Tableaux de nombres

La librairie numpy permet de traiter de façon efficace des tableaux numpy. Nous découvrirons cette librairie, sachant que la librairie Pandas repose sur numpy.

`numpy` est la librairie qui permet de manipuler de larges tableaux de données. Elle offre plusieurs avantages par rapport aux listes : elle est beaucoup plus rapide et surtout permet de faire des calculs sans utiliser de boucles for. Il est indispensable de savoir manipuler correctement les tableaux numpy pour d'une part gagner du temps lors de l'exécution du programme, mais surtout gagner du temps lors de l'écriture du programme.

Contrairement aux listes, la taille et le type de donnée est fixé à la création d'un tableau numpy, ce qui fait que la mémoire est immédiatement allouée au tableau.

Voici quelques exemples :

```
import numpy as np

a = np.arange(10)
print(a**2)
print(np.sin(a))
```

```
[ 0  1  4  9 16 25 36 49 64 81]
[ 0.          0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427
 -0.2794155   0.6569866   0.98935825  0.41211849]
```

Création d'un tableau

Il existe plusieurs fonctions pour créer un tableau.

- Création d'un tableau à partir d'une liste :

```
a = np.array([1, 2, 4])
print(a.dtype)

# data type is calculated automatically
a = np.array([1.2, 2, 4])
print(a.dtype) # all numbers are float

# data type can be forced
a = np.array([1, 2, 4], dtype=float)
print(a.dtype)
```

```
int64
float64
float64
```

- Création d'un tableau uniforme

```
N = 10
a = np.zeros(N)
b = np.ones(N)
```

- Créer un range

```
N = 10
a = np.arange(N)
```

(continues on next page)

(continued from previous page)

```
print('Valeur calculée', a.sum())
print('Valeur théorique', (N*(N+1))/2)
```

```
Valeur calculée 45
Valeur théorique 55.0
```

- Répartition uniforme de points:

```
N = 300
x = np.linspace(0, 2*np.pi, num=N)
```

- Répartition sur une échelle logarithmique:

```
x = np.logspace(0, 2, 11)
print(x)
```

```
[  1.           1.58489319   2.51188643   3.98107171   6.30957344
  10.          15.84893192  25.11886432  39.81071706  63.09573445
 100.        ]
```

Fonctions numpy

Le fonctions arithmétiques ainsi que toutes les fonctions mathématiques de la librairie numpy (ce sont plus ou moins les mêmes que la librairie math) fonctionnent avec des tableaux, c'est à dire qu'il n'est pas nécéssaire d'utiliser une boucle for pour créer un nouveau tableau.

Une fonction pourra être utilisée avec un tableau à condition que toutes les fonctions qu'elle utilise fonctionnent avec un tableau.

```
def gaussienne(x, moyenne=0, ecart_type=1):
    return np.exp( -(x-moyenne)**2 / (2*ecart_type**2))

gaussienne(np.linspace(-3, 3, 11))
```

```
array([0.011109  , 0.05613476, 0.1978987 , 0.48675226, 0.83527021,
       1.        , 0.83527021, 0.48675226, 0.1978987 , 0.05613476,
       0.011109  ])
```

Dans cet exemple, le boucle for est réalisée au niveau de chaque opération élémentaire. Comme toutes les fonctions sont compatible avec numpy, la fonction `gaussienne` l'est. Si on remplace np.exp par math.exp, alors il y a une erreur.

Les fonctions de base de statistiques existent soit sous forme de fonction dans la librairie numpy soit sous forme de méthode sur les tableaux. Il s'agit de `np.mean`, `np.std`, `np.var`. Il existe aussi d'autre fonction de reductions (elle prennent un tableau est renvoie un nombre) : `np.sum`, `np.mean`, `np.max`, `np.min`.

```
x = np.random.rand(1_000_000)
print(x.mean()) # ou print(np.mean(x))
print(x.std())
```

```
0.4999435020786929
0.2886119645283682
```

Remarquons que l'instruction `if` n'est pas une fonction. Il n'y a pas de sens à l'utiliser avec un tableau.

```
x = np.random.rand(10)
if (x>.5):
    y = 1-x
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-85ec1d204b32> in <module>
      1 x = np.random.rand(10)
----> 2 if (x>.5):
      3     y = 1-x

ValueError: The truth value of an array with more than one element is ambiguous. Use
↪a.any() or a.all()
```

Indexer un tableau numpy

On peut indexer un tableau numpy de plusieurs façon différentes

- Comme les listes, avec des entier ou des slices
- Avec un tableau d'entier
- Avec un tableau de booléens (les deux tableaux doivent avoir la même taille)

Les tableaux numpy sont modifiables. On peut aussi écrire de la même façon à condition que les tableaux aient une taille compatible

```
a = np.array([1, 5, 4, 6])

indx = np.array([3, 1])

print(a[indx])

mask = np.array([True, False, False, True])

a[mask]
```

```
[6 5]
```

```
array([1, 6])
```

Remaquons qu'il est possible de créer un tableau de booleen avec les opérateurs de comparaison

```
a = np.random.normal(size=10)
print(a)
print(a>0)
print(a[a>0])
```

```
[ 1.90777979 -0.27021906  0.20357936 -1.01244181  0.48567974  0.10253538
  1.8550102   1.14398201  0.3825322  -0.31723682]
[ True False  True False  True  True  True  True  True False]
[1.90777979 0.20357936 0.48567974 0.10253538 1.8550102  1.14398201
 0.3825322 ]
```

Exemple : créer une fonction valeur absolue compatible numpy

```python
def abs_np(x):
    output = x.copy()
    output[output<0] *= -1
    return output

abs_np(a)
```

```
array([1.90777979, 0.27021906, 0.20357936, 1.01244181, 0.48567974,
       0.10253538, 1.8550102 , 1.14398201, 0.3825322 , 0.31723682])
```

Quelques opérations courantes

Il existe beaucoup de fonction numpy permettant de faire simplement des opérations sur les tableaux :

Voici quelques exemples. Parfois les fonctions numpy existent sous forme d'une méthode (`objet.method`).

```python
import numpy as np
a = np.random.normal(size=10000)

print(np.max(a))
print(a.max()) # Sous forme d'une méthode

np.sum(a)
np.mean(a)
np.var(a)
np.std(a)
np.min(a)
np.max(a)
```

```
4.3373002062093935
4.3373002062093935
```

```
4.3373002062093935
```

Lorsque l'on a un tableau 2D (ou plus), il est possible d'éxecuter l'opération ligne par ligne (ou colonne par colonne).

```python
n = 10
m = 4

# Tableau 2D avec les notes
# Les notes sont aléatoires !
data = np.random.rand(n, m)

moyenne_A = np.mean(data, axis=0) # Sum_i a_ij
moyenne_B = np.mean(data, axis=1) # Sum_j a_ij
```

Opération de tri :

```python
a = np.random.rand(10)
b = np.sort(a)
print(b)
```

```
[0.08764322 0.1230581  0.17004889 0.18385438 0.45941352 0.55188637
 0.61358225 0.70715072 0.73755352 0.95182282]
```

Il est parfois utile de connaître l'indice du maximum ou minimum ou de connaître l'ordre du tri. Cela s'obtient avec les fonctions `argmax`, `argmin` ou `argsort`.

```
i_max = moyenne_eleves.argmax()

print(f'Le meilleur élève est le numéro {i_max}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-fa5dcea6eb43> in <module>
----> 1 i_max = moyenne_eleves.argmax()
      2
      3 print(f'Le meilleur élève est le numéro {i_max}')

NameError: name 'moyenne_eleves' is not defined
```

Autres fonctions utiles:

```
N = 1000
a = np.random.normal(size=N)
# Différence entre deux éléments
np.diff(a)

# Marche aléatoire
b = np.cumsum(a)
```

Lire et écrire dans un fichier

Il existe deux type d'enregistrement : l'enregistrement sous forme d'un fichier texte et celui sous forme binaire. Dans un fichier texte, le tableau doit être de dimension un ou deux. Il est écrit ligne par ligne sous forme de nombre décimaux. Dans un fichier binaire, c'est la mémoire de l'ordinateur qui est recopier dans le fichier. Les fichiers textes ont l'avantage d'être lisible par un humain et d'être compatible avec beaucoup de logiciel, cependant l'écriture et surtout la lecture du fichier prend beaucoup de temps. Les fichiers binaire seront beaucoup plus rapide.

Pour enregistrer au format texte :

```
filename = 'test.dat'
a = np.array([1, 2, 4])
np.savetxt(filename, a)
with open(filename) as f:
    print(f.read())
# Numbers are converted to float
```

```
1.00000000000000000e+00
2.00000000000000000e+00
4.00000000000000000e+00
```

Il est possible de lire des fichiers au format 'csv' tels que ceux exportés par un tableur. Voici un exemple.

```
# Création du fichier
csv_content = """# Tension; courant
1; 2.3
2; 4.5
3; 7.0"""
filename = 'test.csv'
```

(continues on next page)

(continued from previous page)

```
with open(filename, 'w') as f:
    f.write(csv_content)

data = np.loadtxt(filename, delimiter=';')
print(data[:,1])
```

```
[2.3 4.5 7. ]
```

Dans le cas présent, il est aussi possible de lire chaque colonne dans une variable directement :

```
# Utilisation de l'argument unpack
tension, courant = np.loadtxt(filename, delimiter=';', unpack=True)
print(tension/courant)
```

```
[0.43478261 0.44444444 0.42857143]
```

Pour les fichiers binaires, on utilise la fonction `np.load` et `np.save`. Le tableau sera strictement identique après relecture.

```
filename='test.npy'
a = np.array([1, 2, 4])
np.save(filename, a)
new_a = np.load(filename)
print(new_a)
# a est toujours un tableau d'entier
```

```
[1 2 4]
```

1.7 Tracer des graphiques

Nous utiliserons la librairie matplotlib. La page https://matplotlib.org/stable/gallery/index.html contient de nombreux exemples.

Les fonctions sont dans le module `matplotlib.pyplot` qu'il est courant d'importer sous le nom de `plt`.

Exemples

Le plus simple est d'étudier des exemples.

Graphiques simples

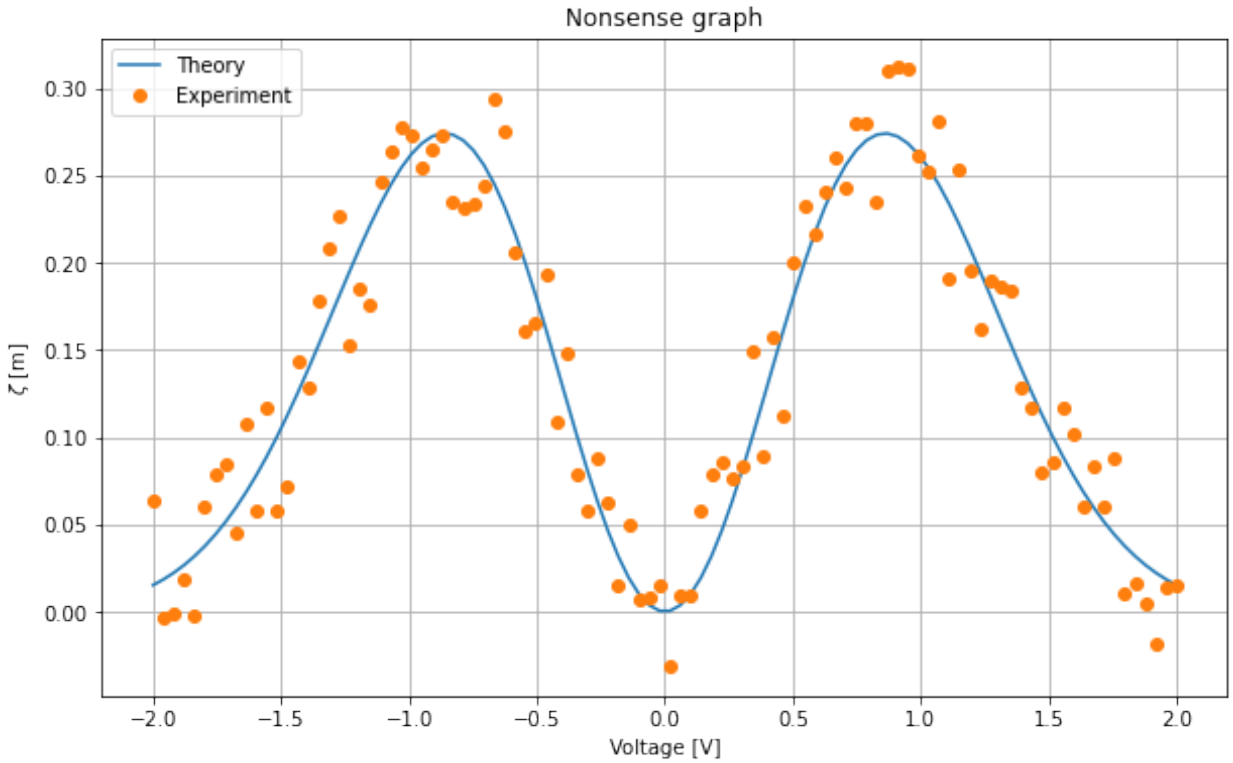
```
import numpy as np

X = np.linspace(-2,2, 100)
Y = np.sin(X)**2*np.exp(-X**2)
Y_noise = Y + .1*(np.random.rand(len(X))-0.5)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
```

(continues on next page)

(continued from previous page)

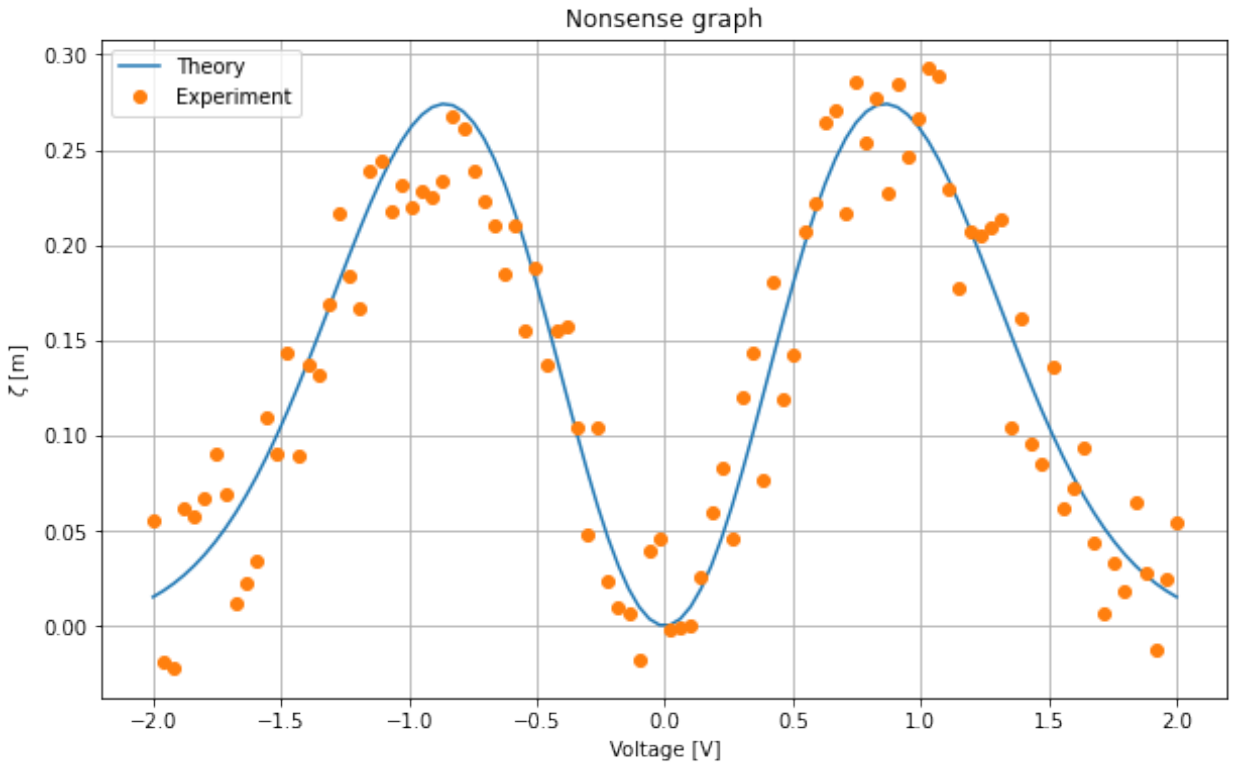
```
plt.plot(X,Y, label=u"Theory")
plt.plot(X,Y_noise,'o', label=u"Experiment")
plt.xlabel(r'Voltage [V]')
plt.ylabel(r'$\zeta$ [m]')
plt.title("Nonsense graph")
plt.legend(loc='upper left')
plt.grid(True)

plt.savefig('mafigure.pdf')
```



Il existe deux syntaxes pour matplotlib, la syntaxe ci-dessus à base de fonctions (syntaxe historiquement utilisée par beaucoup de personnes) et une syntaxe utilisant sur des objets. L'idée est d'utiliser des méthodes des objets figure ainsi que des graphiques (appelé `axes`, il peut y avoir plusieurs `axes`, à ne pas confondre avec `axis` qui sont les abscisses et ordonnées).

Voici le même exemple en orienté objet:

```
from matplotlib.pyplot import figure

fig = figure(figsize=(10, 6))
ax = fig.subplots(1, 1) # Création d'un graphique

ax.plot(X, Y, label=u"Theory")
ax.plot(X, Y_noise, 'o', label=u"Experiment")
ax.set_xlabel(r'Voltage [V]')
ax.set_ylabel(r'$\zeta$ [m]')
ax.set_title("Nonsense graph")
ax.legend(loc='upper left')
ax.grid(True)
```

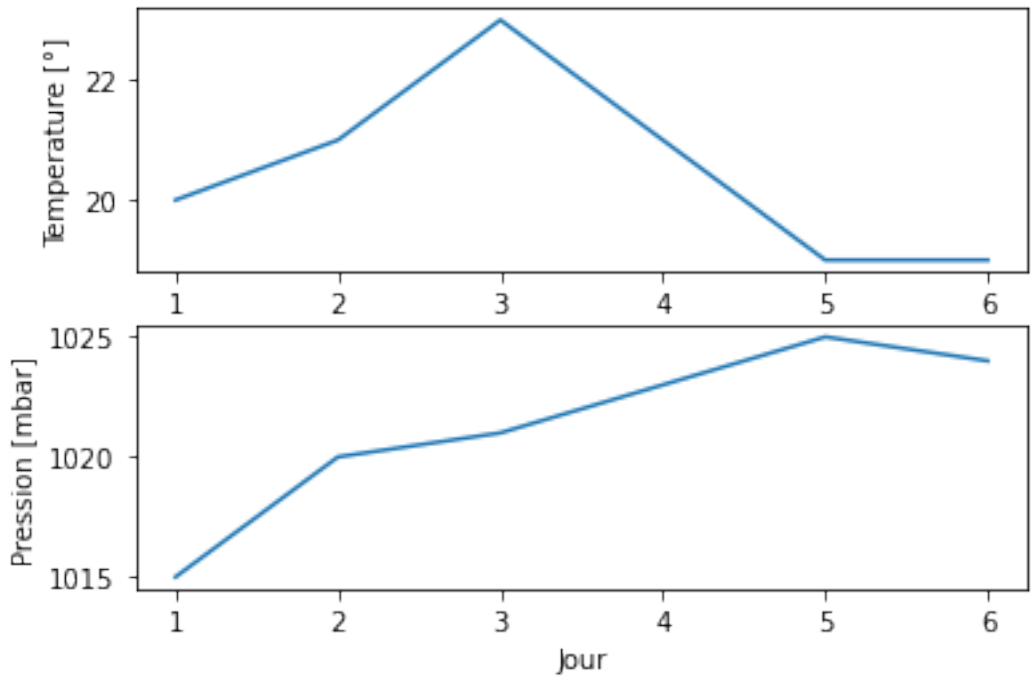


Le syntaxe orientée objet est plus simple lorsque l'on veut mettre plusieurs graphs dans une même figure.

Graphiques multiples

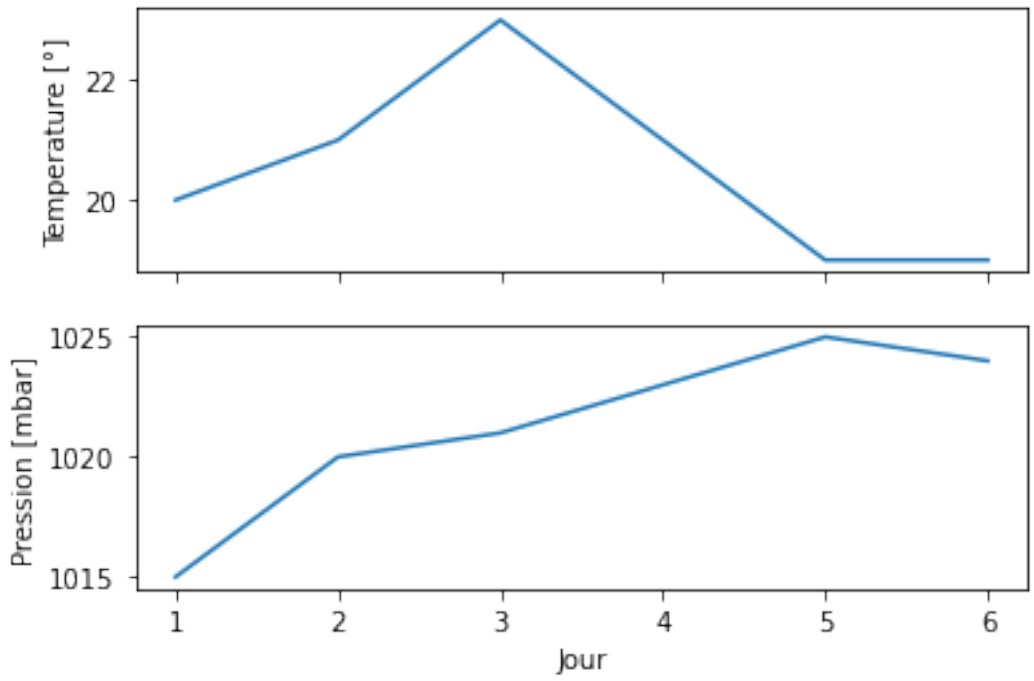
```
jours = np.array([1, 2, 3, 4, 5, 6])
temperature = np.array([20, 21, 23, 21, 19, 19])
pression = np.array([1015, 1020, 1021, 1023, 1025, 1024])

plt.figure()
plt.subplot(2, 1, 1)
plt.plot(jours, temperature)
plt.ylabel('Temperature [°]')
plt.subplot(2, 1, 2)
plt.plot(jours, pression)

plt.xlabel('Jour')
plt.ylabel('Pression [mbar]')
```

```
Text(0, 0.5, 'Pression [mbar]')
```




```
fig = figure()
ax1, ax2 = fig.subplots(2, 1, sharex=True)
ax1.plot(jours, temperature)
ax1.set_ylabel('Temperature [°]')

ax2.plot(jours, pression)

ax2.set_xlabel('Jour')
ax2.set_ylabel('Pression [mbar]')
```

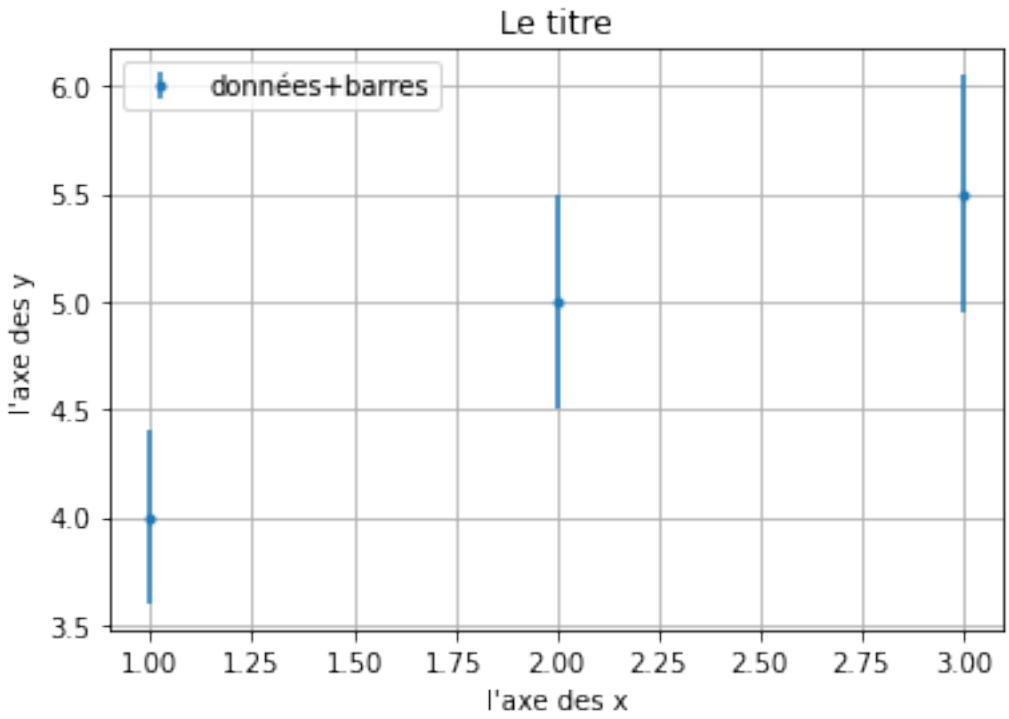
```
Text(0, 0.5, 'Pression [mbar]')
```



Barres d'erreur

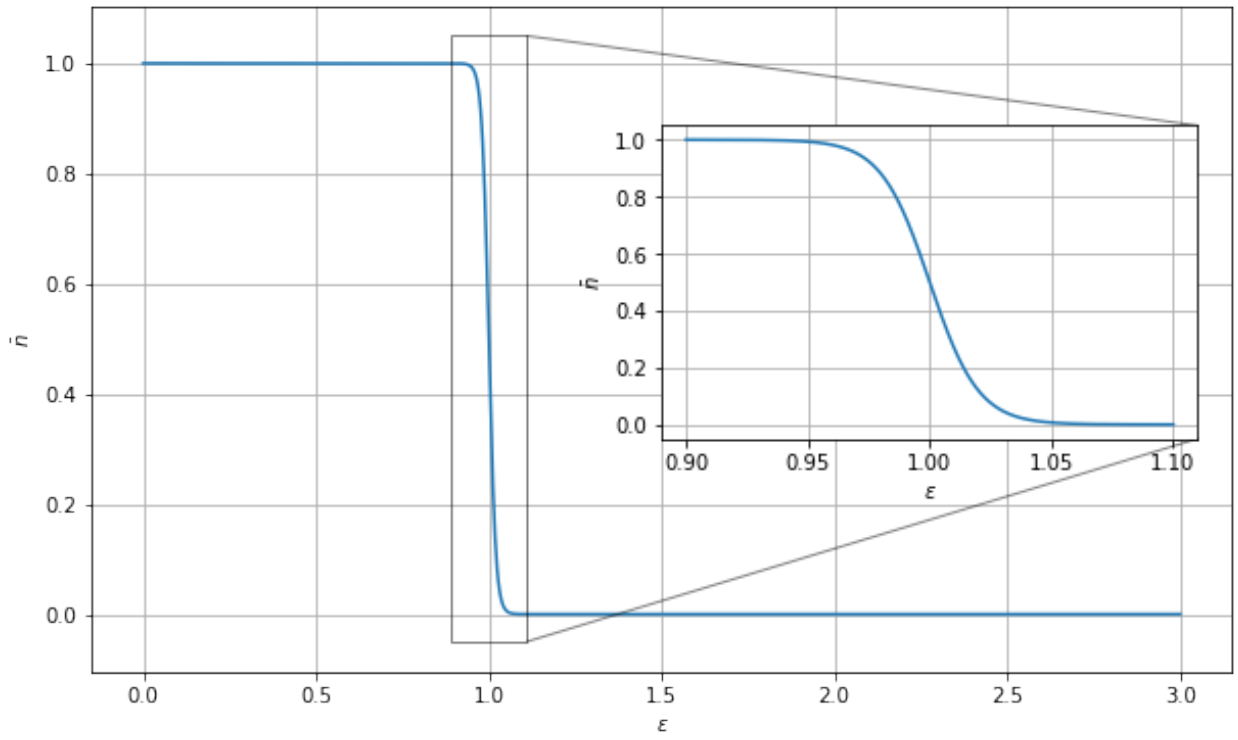
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 2, 3])
y = np.array([4, 5, 5.5])
erreurs_y = 0.1 * y

plt.errorbar(x, y, erreurs_y, fmt='.', label="données+barres")

plt.xlabel("l'axe des x")
plt.ylabel("l'axe des y")
plt.legend(loc=2)
plt.grid()
plt.title("Le titre")
```

```
Text(0.5, 1.0, 'Le titre')
```



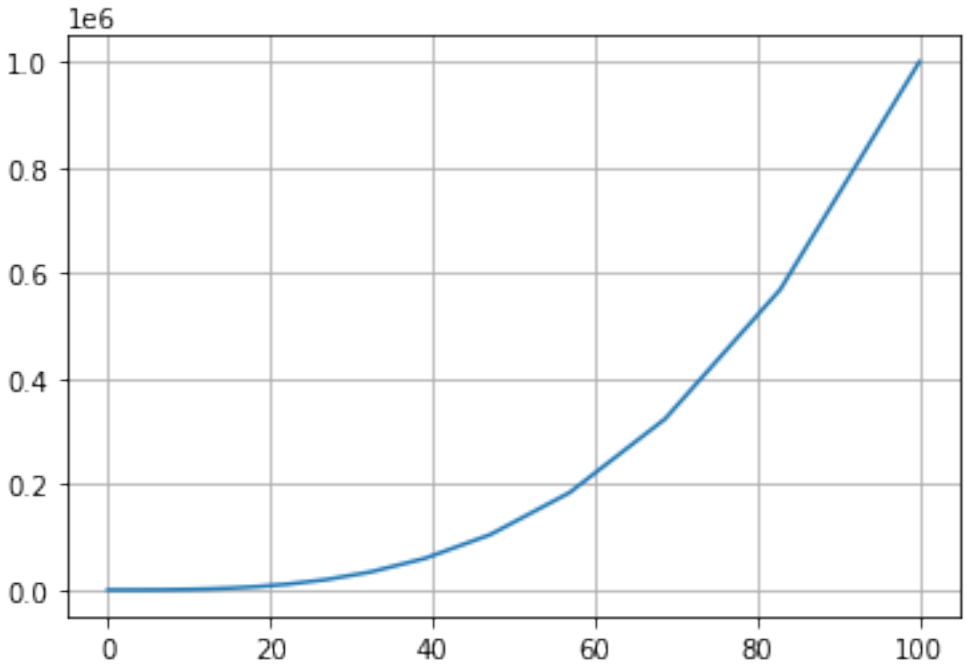
Inset

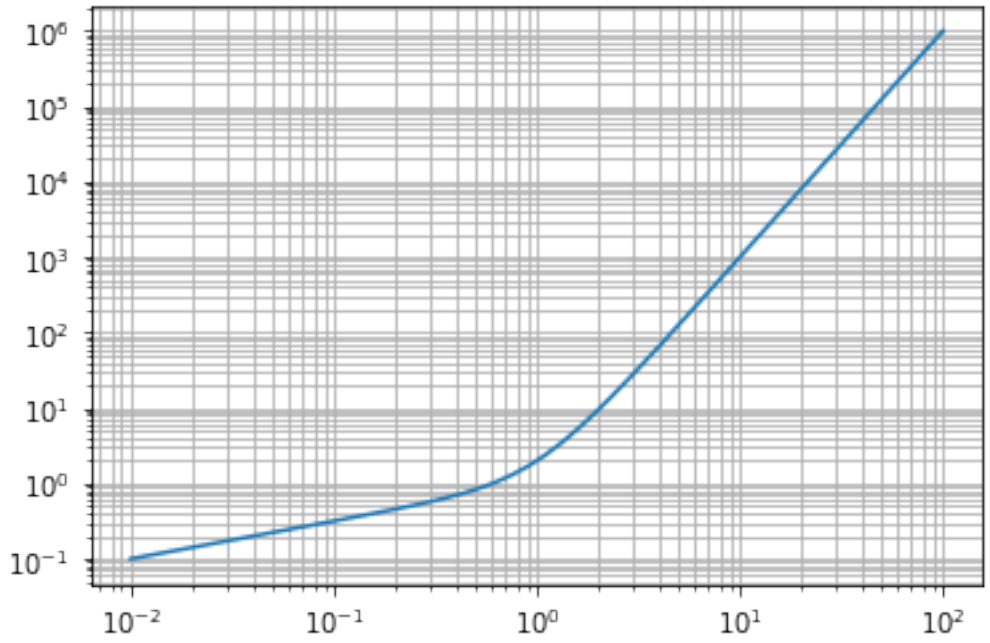
```
import numpy as np
import matplotlib.pyplot as plt

def fermi_dirac(epsilon, mu, beta):
    return 1/(np.exp(beta*(epsilon - mu))+1)

x = np.linspace(0, 3, num=1000)
x_zoom = np.linspace(0.9, 1.1, num=1000)

fig = plt.figure(figsize=(10, 6))
ax = fig.subplots(1, 1)

ax.plot(x, fermi_dirac(x, mu=1, beta=100))

axins = ax.inset_axes([0.5, 0.35, 0.47, 0.47])
axins.plot(x_zoom, fermi_dirac(x_zoom, mu=1, beta=100))
ax.indicate_inset_zoom(axins, edgecolor="black")

for a in [ax, axins]:
    a.set_xlabel(r'$\epsilon$')
    a.set_ylabel(r'$\bar n$')
    a.grid()
```



Echelle logarithmique

```
x = np.logspace(-2, 2)
y = x**3 + np.sqrt(x)

fig = figure()
ax = fig.subplots(1, 1)
ax.plot(x, y)
ax.grid()

fig = figure()
ax = fig.subplots(1, 1)
ax.loglog(x, y)
ax.grid(which="both")
```


1e6
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


10^{6}
10^{5}
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

Les commandes

Créer le graphique

- `plt.figure(figsize=(w, h))` : Permet de créer la figure. Largeur et hauteur en pouce (2.54 cm)
- `fig.subplots(n, m)` : découpe la figure en n (verticalement) et m (horizontalement) graphiques. Cette commander renvoie une matrice de n par m axes. Avec la méthode fonctionnel, la commande `plt.subplot(n, m, i)` créer le ième axe de la matrice et le sélectionne pour le prochain plot.

Tracer des données

Toutes ces fonctions sont les mêmes en mode fonctionnel ou objet.

- `plot(X, Y)` trace avec une ligne
- `plot(X, Y, fmt)` pour contrôler la forme et la couleur. Couleurs : blue (b), red (r), black (k), green (g). Forme : disque (o), diamand (d), ligne pointillée (:), …
- `plot(X, Y, color=..., linestyle=..., linewidth=...)`, syntaxe plus explicite. Les couleurs peuvent être en tout lettre (red, blue, orange, ..) suivre le cycle par défaut (C0, C1, C2, C3), ou être définie par l'intensité RGB. Le style des lignes : solid, dotted, dashed, dashdot. La linewidth est en point.
- `loglog`, `semilogy`, `semilogx` : pour tracer avec des échelles logarithmique. On peut aussi changer avec `plt.xscale('log')` ou `ax.set_xscale`
- `errorbar(x, y, yerr=..., xerr=...)`: pour tracer des points avec des barres d'erreur.
- `scatter(x, y, s=..., c=...)` pour tracer un nuage de points avec des tailles (s) ou des couleurs (c) sous forme d'un tableau.

Légendes,...

- Toutes les fonctions ci-dessus ont un argument optionel `label`. On peut alors utiler la fonction plt.legend (ou ax.legend) pour afficher la légende.
- `xlabel`, `ylabel` : ne jamais oublier les unités
- `grid` : Tracer une grille en arrière plan
- `text` : rajoute un texte

Remarques : pour les chaînes de caractère, il est possible d'utiliser des formules latex en utilisant des `$`. Il faut alors faire attention aux `\` : en effet il est possible qu'ils soient interprété comme des caractères spéciaux (par exemple `\n` est un retour à la ligne). Pour éviter ceci, on utilise des chaîne brutes (raw string), préfixées par un `r`.

2 Feuilles d'exercices

2.1 Exercices sur les fonctions

Que fait cette fonction ?

Répondre aux questions de cet exercice sans s'aider de l'ordinateur, puis vérifier.

1. On souhaite calculer $\sum \frac{1}{i}$. Quelle est la différence entre les différentes fonctions si dessous ? Laquelle est la “bonne” fonction, laquelle la “pire”

```
def serie_1(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
        print(output)

serie_1(5)

def serie_2(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
        return output

serie_2(5)

def serie_3(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
    print(output)

serie_3(5)


def serie_4(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
    return output

serie_4(5)
```

2. Parmi tous les appels de la fonction `f` ci dessous, lequels vont faire une erreur ? Quelle sera l’erreur ?

```
def f(a, b):
    return a*2 + b

p = [1, 2]
```

```
f(1, 3, 4)
f(1, 2)
f(Bonjour, Hello)
f(1, a=2)
f(b=1, a=2)
f("Bonjour", "Hello")
f[1, 2]
f("Bonjour, Hello")
f(1)
f(**p)
f(*p)
```

3. Qu’est il affiché ? Dans quels fonction `x` est une variable globale ?

```
def f1(x, y):
    print(x + y)

def f2(y):
    x = 15
    print(x + y)

def f3(x, y):
    print(x + y)
    x = 15

def f4(y):
    print(x + y)
    x = 15

x = 10
f1(1, 2)
x = 5
f1(1, 2)
f2(1)
print(x)
f3(1, 2)
f4(1) # Attention, il y a un piège ici.
```

4. Qu'est il affiché ?

```
from math import sin

pi = 3.141592653589793

def g():
    return sin(pi/2)

def f():
    pi = 0
    return sin(pi/2)

print(f())
print(g())
pi = 0
print(f())
print(g())
```

4. Qu'est il affiché ?

```
def f(a, b, c):
    print(100*a + 10*b + c)

a = 1
b = 2
c = 3
f(a, b, c)
f(c, b, a)
f(a, b=c, c=b)
f(a=a, b=a, c=a)
```

Fonction cos_deg

Ecrire une fonction qui renvoie le cosinus d'un angle exprimé en degré

Volume d'un cône

1. Ecrire une fonction qui renoit le volume d'un cône de rayon r et hauteur h.
2. Ecrire une fonction qui envoit le volume d'un tronc de cône de rayon r_1 et r_2.
3. Ecrire une seule fonction pour laquelle le tronc de cône est par défaut un cône (i.e. $r_2 = 1$)

Fonction datetime

Importer la fonction `datetime` du module `datetime` et regarder se documentation.

1. Utiliser cette fonction pour entrer votre date de naissance (et l'heure si vous la connaissez) en nomant explicitement les arguments.
2. Même question mais après avoir mis les arguments dans une liste
3. Dans un dictionnaire

Fonction date en français

1. Ecrire une fonction `date_en_francais` qui renvoie un objet `date` mais dont les arguments sont en français (`annee`, `mois`, `jour`)
2. Idem avec la fonction `datetime` du module `datetime` (il faudra rajouter les arguments optionels `heure`, `minute`, `seconde`)

Polynômes

On considère un polynôme de degrés n : $a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$.

1. Ecrire une fonction `eval_polynome_troisieme_degres(x, a_0, a_1, a_2, a_3)` qui évalue un tel polynôme en x.
2. Ecrire une fonction qui permettra d'évaluer un polynôme de degré inférieur à 4 est que l'on pourra utiliser de la façon suivante :

```
eval_polynome(x, 3, 4) # 3x + 4
eval_polynome(x, a_0=2, a_2=4) # 4x^2 + 4
```

3. Faire en sorte que cette fonction marche pour n'importe quel degré.

Equation du second degré

On souhaite trouver les solutions de l'équation : $ax^2 + bx + c = 0$. On rappelle que pour cela on calcule le discriminant $\Delta = b^2 - 4 * a * c$.

- Si $\Delta > 0$, les solutions sont $\frac{-b \pm \sqrt{Delta}}{2a}$
- Si $\Delta = 0$, il y a une solution (double) $\frac{-b}{2a}$
- Si $\Delta < 0$, les solutions sont $\frac{-b \pm i\sqrt{Delta}}{2a}$

1. Ecrire une fonction equation_second_degre qui donne les solutions.
2. Tester l'équation $x^2 - 3x + 2$
3. Ecrire une fonction qui résout une équation du premier degré.
4. Faire en sorte que cette fonction soit appelée lorsque $a = 0$.

Coordonnée polaire d'un nombre complexe

On considère une nombre complexe z et sa représentation polaire : $z = re^{i\theta}$

1. Ecrire une fonction qui à partir de r et θ renvoie z

Nombres premiers

Ecrire une fonction qui renvoie True si un nombre est premier et False sinon. Pour cela, on testera si le nombre est divisible par les entiers successifs à partir de 2.

On admet que si n n'est pas premier, alors il existe un entier $p \geq 2$ tel que $p^2 \leq n$ qui est un diviseur de n.

1. Ecrire la fonction à l'aide d'une boucle.
2. Combien y a t-il d'années au XXIème siècle qui sont des nombres premiers ?
3. Si vous n'êtes pas convaincu que c'est mieux avec des fonctions, faites le sans…

Mention

Ecrire une fonction qui à partir de la note sur 20 donne la mention

2.2 Exercices sur les nombres

Fonctions mathématiques

- Est ce que la fonction log est le logarithme décimal ou népérien ?
- Calculer $x = \sqrt{2}$ puis calculer x^2. Que se passe-t-il ?
- Calculer arccos $\frac{\sqrt{2}}{2}$ et comparer à sa valeur théorique.

Constante de structure fine

La constante de structure fine est définie en physique comme étant égale à

$$\alpha = \frac{e^2}{2\epsilon_0 hc}$$

où

- e est la charge de l'électron et vaut $1.602176634 \times 10^{-19} C$
- h est la constante de Planck et vaut $6.626\,070\,15 \times 10^{-34} Js$
- ϵ_0 la permitivité du vide et vaut $8.8541878128 \times 10^{-12} F/m$
- c la célérité de la lumière dans le vide, $c = 299792458 m/s$

Définissez en Python les variables `e`, `hbar`, `epsilon_0` et `c`. Calculez α et $1/\alpha$

Précision des nombres

- Soit $x = 1$ et $\epsilon = 10^{-15}$. Calculez $y = x + \epsilon$ et ensuite $y - x$.
- Pourquoi le résultat est différent de 10^{-15}.
- Que vaut cette valeur ?

Calcul d'une dérivée

On considère une fonction $f(x)$. On rappelle que la dérivée peut se définir comme

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Pour calculer numériquement une dérivée, il faut évaluer la limite en prenant une valeur 'petite' de ϵ.

On prendra comme exemple $f(x) = \sin(x)$.

- Calculer numériquement la dérivée de f en $\pi/4$ en utilisant la formule pour $\epsilon = 10^{-6}$.
- Comparer à la valeur théorique $\cos(x)$ pour différentes valeurs de ϵ que l'on prendra comme puissance de 10 ($\epsilon = 10^{-n}$). Que se passe-t-il si ϵ est trop petit ? trop grand ?
- Ecrire la fonction `sin_prime(x, epsilon)` qui calcule la dérivée de sin en x
- Ecrire une fonction qui prend une fonction quelconque et renvoie la fonction dérivée.

Nombre complexe

- Ecrire une fonction qui calcule le module d'un nombre complexe z
- Ecrire une fonction qui à partir de r et θ renvoie le nombre $z = re^{i\theta} = r\cos(\theta) + ir\sin(\theta)$

2.3 Exercices sur les conteneurs

Manipulation des listes

On considère la liste [1, 5, 3, 5, 6, 2]

1. Écrire une fonction 'somme' qui renvoie la somme des éléments d'une liste de nombres. On fera explicitement la boucle for.
2. Écrire une fonction 'maximum' qui renvoie le maximum des éléments d'une liste de nombres. On fera explicitement la boucle for.
3. Écrire une fonction 'arg_maximum' qui renvoie l'indice du maximum d'une liste de nombres. On fera explicitement la boucle for.
4. Écrire une fonction 'trouve' qui renvoie l'indice correspondant à l'argument. On fera explicitement la boucle for.
5. Comment répondre aux questions 1, 2, 3, 4 en utilisant des fonctions déjà existantes ?

Liste comprehension

1. Créer une liste nomée `nombres` contenant les entiers de 0 à 9 inclus
2. Créer une liste contenant la racine carré des éléments de `nombres` (on utilisera une comprehension de liste)
3. Créer une liste contenant tous les nombres pairs de la listes `nombres` (on utilisera une comprehension de liste)
4. Toujours en utilisant un comprehension de liste, considérant deux listes `l1` et `l2`, créer une nouvelle liste contenant les couples pris deux à deux de l1 et l2. On supposera que les deux liste ont la même longueur. Quelle fonction python fait la même chose ?
5. En utilisant la fonction de la question 4 et la liste de la question 2 vérifier que l'on a bien $y = x^2$ pour chaque élément.

Exercice de base sur les dictionnaires

1. Tout d'abord, nous allons créer un petit dictionnaire qui contient des informations sur un étudiant. Utilisons les clés et valeurs suivantes en exemple :
   - 'nom': 'Jean Dupont'
   - 'âge': 20
   - 'filière': 'Informatique'
2. Modifier l'age pour qu'il soit égal à 21
3. Afficher le genre de l'étudiant si il possède une telle clé sinon afficher un message inquant que l'on ne connait pas son genre.

Exercice sur les ensembles

La fonction chr permet de convertir un code ASCII en un caractère. La liste des lettres majuscules peut être obtenue à partir de la commande suivante :

```
liste_majuscules = [chr(65+i) for i in range(26)]
```

On souhaite vérifier qu'un mot de passe entré par un utilisateur est sufisament compliqué. Voici les règles :

- Il doit contenir 12 caractères différents
- Il doit contenir au moins 2 majuscules différentes
- Il doit contenir au moins un caractère de ponctuation `.,;:!?`
- Il ne doit pas contenir d'espace

Ecrire une fonction qui renvoie True si toutes les conditions sont vérifiées et False sinon

2.4 Chaînes de caractères

Liste de prix

Voici un liste de prix sous forme d'un dictionnaire. Afficher la liste de prix de la façon suivante :

```
price_liste = {'tomates':3.4,
     'pommes':2.49,
     'oignons':1.45}
```

Unicode

Afficher toutes les lettres greques de α à ω. (On pourra copier coller ces lettres pour avoir leurs unicodes)

Combien La Fontaine a t-il utilisé de mots différents dans ses fables ?

Les lignes suivantes permettent de télécharger l'ensemble des fables de la Fontaine.

- Combien y a-t-il de mots différents ? On pourra d'abord remplacer toutes les ponctuations par des espaces, puis créer une liste de mots que l'on mettra en minuscule. On créera ensuite un ensemble dont on regardera la taille.
- Quelle est le mot le plus long ?

2.5 Objets

Vecteur

Créer une classe Vecteur3D. Chaque vecteur aura 3 attributs : x, y, z

- Écrire une méthode norme qui renvoie la norme.
- Écrire la méthode `__add__` pour faire la somme entre deux veteurs
- Écrire la méthode `__mul__` pour faire soit le produit par un scalaire ($2\vec{u}$) ou le produit scalaire ($\vec{u} \cdot \vec{v}$).

Bibliographie

Un livre est décrit par son titre, auteur et année de publication (pour faire les choses simplements). Écrire une classe `Livre` qui enregistre ces informations. Ecrire la méthode `__repr__` et `__str__`.

Une bibliographe est une liste de livre. Écrire la classe `Bibliographie` qui enregistre une liste de livre (on stockera la liste de livre sous forme d'une liste qui sera un attribut de la bibliographie).

L'objectif final est de pouvoir faire ceci ::

```
livre1 = Livre("A very nice book", "F. Dupont", 2014)
livre2 = Livre("A very smart book", "A. Einstein", 1923)
livre3 = Livre("A very stupid comic", "D. Duck", 1937)

bibliographie = Bibliographie([book1, book2, book3])
```

Maintenant que tout est fait sous forme d'objet, on peut imaginer écrire plusieurs méthode :

- Écrire une méthode `filter_by_year` qui fait une nouvelle bibliographie ne contenant que les livres d'une année donée.
- Écrire une méthode `to_html` qui formate correctement la bibliographie. La méthode de la classe Bibliographie devra appeler une méthode pour chaque Livre.

Et en HTML ::

```
<table>
    <thead>
        <tr> <th>Auteur</th><th>Titre</th><th>Année</th></tr>
    </thead>
    <tbody>
       <tr><td>F. Dupont</td><td>2014</td><td>A very nice book</td></tr>
       <tr><td>A. Einstein</td><td>1923</td><td>A very smart book</td></tr>
       <tr><td>D. Duck</td><td>1937</td><td>A very stupid comic</td></tr>
    </tbody>
</table>
```

Remarque : si un objet possède une méthode `_repr_html_`, alors le jupyter notebook utilsera automatiquement la représentation en HTML. Rajouter cette méthode (qui appelera to_html).

Système de calcul formel

Cet exercice est à but purement pédagogique. Pour utiliser un système de calcul formel sous Python, la librairie `sympy` existe et fonctionnera bien mieux que ce que l'on va faire !

L'objectif de ce TD est de réaliser un système de calcul formel qui permettra de manipuler des expressions algébriques simples et de réaliser des opérations simples. Par exemple, on souhaite pouvoir effectuer ::

```
x = Symbol('x')
y = Symbol('y')

s = 2*x*y + sin(x)*y

print(s.diff(x)) # Dérivée par rapport à x
```

Chaque expression sera représentée par un arbre. Les feuilles de l'arbre seront soit les symboles soit les constantes numériques. Les noeuds seront des fonctions à un ou plusieurs argument (sinus, somme, opposé, …). Le nom de la classe

du noeud désignera la fonction. Les “enfants” du noeud seront les arguments de la fonction. Par exemple l'expression ci dessus correspondra à l'objet suivant ::

```
# sA : 2*x*y
sA = Prod(Prod(Number(2), Symbol('x')), Symbol('y'))
# sB : sin(x)*y
sB = Prod(Sin(Symbol('x')), Symbol('y'))

s = Sum(sA, sB)
```

Structure du programme

Voici la structure de base ::

```
class Expr(object):
    pass

class Node(Expr):
    pass

class Leave(Expr):
    pass
```

Pour les feuilles ::

```
class Symbol(Leave):
    pass

class Number(Leave):
    pass
```

Ensuite on définit les fonctions ::

```
class Function(Node):
    """ Function with an arbitrary number of arguments """
    pass
```

Les opérateurs sont des fonctions comme les autres, mais elle seront simplement affichées différemment ::

```
class BinaryOperator(Function):
    pass

class Sum(BinaryOperator):
    pass
# Idem pour Sub, Div, Prod, Pow

class UnitaryOperator(Function):
    pass

class Neg(UnitaryOperateor):
    pass
```

Les fonction mathématiques, qui prennent un seul argument ::

```
class MathFunction(Function):
    pass
```

(continues on next page)

(continued from previous page)

```
class Sin(MathFunction):
    pass
```

Questions

On va procéder étape par étape. Il sera plus facile de commencer par les feuilles avant d'écrire la structure globale.

1. Ecrire le `__init__` de la classe Symbol et Number
2. Ecrire une méthode `display` sur ces classes afin de renvoyer une chaîne de caractère contenant le symbole ou le nombre
3. Ecrire le `__init__` de la class Sin ainsi que le display. Le display devra appeler le display de l'argument. Par exemple ceci devra fonctionner ::

```
>>> x = Symbol('x')
>>> Sin(x).display()
sin(x)
>>> Sin(Sin(x)).display()
sin(sin(x))
```

4. Généraliser le **init** et le display de `Sin` afin de le mettre dans la class MathFunction. On rajoutera un attribut de classe à chaque sous classe de MathFunction ::

```
class Sin(MathFunction):
    funtion_name = 'sin'
```

5. Faire de même pour les opérateurs binaires. On pourra commencer par simplement le faire pour Sum, puis généraliser avec un attribut de classe ::

```
class Sum(BinaryOperator):
    operator_name = '+'
```

6. A ce stade quelque chose comme ceci devrait fonctionner ::

```
x = Symbol('x')
y = Symbol('y')
Sum(x, Sin(Prod(x, y)))
```

Rajouter les méthodes `__add__`, `__mul__`, etc à la classe Expr afin de pouvoir écrire :

```
>>> x + Sin(x*y)
```

7. Ecrire les méthodes `evaluate` afin de calculer la valeur numérique d'une expression. Cette méthode fonctionnera de la sorte :

```
>>> expr = x + Sin(x*y)
>>> expr.evaluate(x=1, y=3)
```

On aura donc le protocole suivant ::

```
def evaluate(self, **kwd):
    pass
```

Le dictionnaire kwd sera passé récursivement jusqu'aux feuilles et sera utilisé pour évaluer les symboles.

Les opérateurs binaires numériques sont définis dans le module `operator` et les fonctions dans le module `math`. Afin de factoriser le code, on rajoutera donc simplement un attribut de classe du type `operator_function = operator.add` pour les opérateurs binaires et `math_function = math.sin` pour les fonctions.

8. Maintenant que vous avez compris le principe, il devrait être facile d'écrire une méthode `diff` qui effectue la dérivée par rapport à une variable !

9. Reste à simplifier les expressions. Une technique consiste à créer des règles de simplifications sous forme de méthode que l'on regroupe ensuite dans une liste ::

```
class Sum(BinaryOperator):
    operator_name = '+'
    operator_function = operator.add

    def simplication_de_deux_nombres(self):
        if isinstance(self.arg1, Number) and
                        isinstance(self.arg2, Number):
            return Number(self.arg1.value + self.arg2.value)

    def simplication_addition_avec_zero(self):
        pass

    liste_simplication = ['simplication_de_deux_nombres',
                    'simplication_addition_avec_zero']
```

Ensuite, il faut réussir à appeler correctement et de façon recursive ces méthodes…

10. Pour l'affichage des opérateurs binaires, les règles de priorité peuvent être utilisées pour éviter de mettre trop de parenthèses. Par exemple, dans le cas `a*(b+c)`, la multiplication appelle le display de l'addition. Comme elle est prioritaire, l'addition va renvoyer le résulat avec des parenthèses. Dans le cas inverse `a + b*c`, c'est inutile. Il faut donc que le display d'un opérateur passe sa priorité à ses enfants lors de l'appel de display. Implémenter ce principe.

2.6 Utilisation de numpy

Calcul Monte-Carlo de la surface d'un disque

On rappelle que la fonction np.random.rand permet de créer une distribution uniforme de points entre 0 et 1.

1. Créer des tableaux X et Y de variables uniforméments réparties entre -1 et 1
2. Evaluer la probabilité que le point de coordonnées (x, y) soit dans le cercle unité ?
3. En déduire une estimation ed π
4. De la même façon, évaluer le volume d'une sphère en dimension 13

Calcul de la moyenne

Un tableau numpy contient les notes d'une classe. Chaque ligne correspond à un élève et chaque colone à un examen.

```
import numpy as np

N_eleves = 35
N_examens = 3

notes = np.random.rand(N_eleves, N_examens)*20
```

1. Calculer la moyenne de chaque élève. Calculer la moyenne pour chaque examen.
2. Calculer la moyenne de chaque élève, sachant que les coeficients pour les 3 examens sont de 1, 3 et 2.

2.7 Exercices sur les graphiques

Population

À partir de la page [https://fr.wikipedia.org/wiki/D%C3%A9mographie_de_l%27Europe}(https://fr.wikipedia.org/wiki/D%C3%A9mogra et de la page https://en.wikipedia.org/wiki/Demographics_of_France tracer sur un même graph la population du monde, de l'europe et de la france depuis 1800.

```
europe_annees = [1800, 1850, 1900, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2019]
europe_population = [187000000 , 266000000 , 420000000 , 549329000 , 605407000 ,␣
↪656919000 , 693567000 , 720858000 , 725558000 , 736413000 , 747183000]
monde_population = [813000000, 1128000000, 1550000000, 2536431000 , 3034950000 ,␣
↪3700437000 , 4458004000 , 5327231000 , 6143494000 , 6956824000 , 7713468000 , ]

france_annees = [1806, 1821, 1831, 1841, 1851, 1872, 1881, 1891, 1901, 1911, 1921,␣
↪1931, 1946, 1950, 1960, 1970, 1980, 1990, 2000, 2010, 2020, 2022]
france_population = [29107000, 30462000, 32569000, 34230000, 35783000, 36103000,␣
↪37672000, 38343000, 38962000, 39605000, 39108000, 41524000, 40125230, 41647258,␣
↪45464797, 50528219, 53731387, 56577000, 58858198, 62765235, 67287241, 67813396]
```

Statistiques sur le COVID-19

Le fichier `data_covid.dat` contient trois colonnes : la première est la date, la seconde le nombre cumulé de décès en hôpital liés au COVID-19 et la troisième le nombre cumulé de décès en EHPAD liés au COVID-19.

Le fichier commence le lundi 2 mars 2020. C'est l'origine de nos dates (jour 0). Dans la suite, on entend par date ou numéro du jour le nombre de jours écoulés depuis le 2 mars 2020. Ce fichier a été extrait de la base de donnée en janvier 2021 (il ne comprend donc que la première et début de la deuxième vague).

1. Lire ce fichier à l'aide de la commande loadtxt, et extraire les trois colonnes dans trois variables.
2. Tracer le graphe du nombre de décès cumulé en hôpital et en EHPAD en fonction de la date.
3. Ajouter sur le même graph le nombre total de décés cumulé
4. Au cours des 30 premiers jours, le nombre cumulé de décès en hôpital suit une loi proche d'une exponentielle. Tracer cette courbe en échelle semi-logarithmique.
5. Au cours de ces 30 premiers jours, le nombre de décès peut s'écrire $N(j) = N_0(1+a)^{j-j_0}$. Tracer cette courbe avec les paramètres pour lesquels elle s'ajuste bien à l'œil. Quelle est la valeur de a? (Nous ne demandons pas dans cette question de faire un ajustement, mais simplement de superposer les deux courbes).

6. Calculer le tableau du nombre quotidien de décès en hôpital et en EHPAD en fonction de la date. Et tracer la courbe pour l'hôpital. (Remarque, il est possible de faire cette opération sans boucle)
7. Quel jour a connu le plus grand nombre de décès en hôpital ? (on utilisera la fonction `argmax`)
8. Combien de jours ont connu plus de 50 décés en EHPAD ?
9. Comment extraire en une ligne le nombre de décés pour un jour de la semaine donné ?
10. Calculer et tracer le nombre total de décès en hôpital par jour de la semaine. Pour quel jour de la semaine enregistre-t-on le moins de décès ?

3 Correction

3.1 Exercices sur les fonctions

Que fait cette fonction ?

Répondre aux questions de cet exercice sans s'aider de l'ordinateur, puis vérifier.

1. On souhaite calculer $\sum \frac{1}{i}$. Quelle est la différence entre les différentes fonctions si dessous ? Laquelle est la "bonne" fonction, laquelle la "pire"

```
def serie_1(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
        print(output)

serie_1(5)

def serie_2(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
        return output

serie_2(5)

def serie_3(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
    print(output)

serie_3(5)


def serie_4(n):
    output = 0
    for i in range(1, n+1):
        output = output + 1/i
    return output

serie_4(5)
```

```
1.0
1.5
1.8333333333333333
2.083333333333333
2.283333333333333
2.283333333333333
```

```
2.283333333333333
```

2. Parmi tous les appels de la fonction `f` ci dessous, lequels vont faire une erreur ? Quelle sera l'erreur ?

```
def f(a, b):
    return a*2 + b

p = [1, 2]
```

```
f(1, 3, 4)
f(1, 2)
f(Bonjour, Hello)
f(1, a=2)
f(b=1, a=2)
f("Bonjour", "Hello")
f[1, 2]
f("Bonjour, Hello")
f(1)
f(**p)
f(*p)
```

3. Qu'est il affiché ? Dans quels fonction `x` est une variable globale ?

```
def f1(x, y):
    print(x + y)

def f2(y):
    x = 15
    print(x + y)

def f3(x, y):
    print(x + y)
    x = 15

def f4(y):
    print(x + y)
    x = 15

x = 10
f1(1, 2)
x = 5
f1(1, 2)
f2(1)
print(x)
f3(1, 2)
f4(1) # Attention, il y a un piège ici.
```

```
3
3
```

(continues on next page)

(continued from previous page)

```
16
5
3
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-3-bf7078096a9a> in <module>
     21 print(x)
     22 f3(1, 2)
---> 23 f4(1)

<ipython-input-3-bf7078096a9a> in f4(y)
     11
     12 def f4(y):
---> 13     print(x + y)
     14     x = 15
     15

UnboundLocalError: local variable 'x' referenced before assignment
```

4. Qu'est il affiché ?

```
from math import sin

pi = 3.141592653589793

def g():
    return sin(pi/2)

def f():
    pi = 0
    return sin(pi/2)

print(f())
print(g())
pi = 0
print(f())
print(g())
```

```
0.0
1.0
0.0
0.0
```

4. Qu'est il affiché ?

```
def f(a, b, c):
    print(100*a + 10*b + c)

a = 1
b = 2
c = 3
f(a, b, c)
f(c, b, a)
f(a, b=c, c=b)
f(a=a, b=a, c=a)
```

```
123
321
132
111
```

Fonction cos_deg

Ecrire une fonction qui renvoie le cosinus d'un angle exprimé en degré

```
from math import cos, pi

def cos_deg(angle):
    return cos(angle/180*pi)
```

Volume d'un cône

1. Ecrire une fonction qui renoit le volume d'un cône de rayon r et hauteur h.
2. Ecrire une fonction qui envoit le volume d'un tronc de cône de rayon r_1 et r_2.
3. Ecrire une seule fonction pour laquelle le tronc de cône est par défaut un cône (i.e. $r_2 = 1$)

```
from math import pi

def volume_cone(h, r):
    return pi*h*r**2/3

def volume_tronc_cone(h, r1, r2):
    return pi*h*(r1**2 - r2**2)/3

def volume_tronc_cone(h, r1, r2=0):
    return pi*h*(r1**2 - r2**2)/3
```

Fonction datetime

Importer la fonction `datetime` du module `datetime` et regarder se documentation.

1. Utiliser cette fonction pour entrer votre date de naissance (et l'heure si vous la connaissez) en nomant explicitement les arguments.
2. Même question mais après avoir mis les arguments dans une liste
3. Dans un dictionnaire

```
from datetime import date, datetime

print(date(year=2011, month=6, day=2))

liste = [2011, 6, 2]

print(date(*liste))

parametre = {'year':2011, 'month':6, 'day':2}

print(date(**parametre))
```

```
2011-06-02
2011-06-02
2011-06-02
```

Fonction date en français

1. Ecrire une fonction `date_en_francais` qui renvoie un objet `date` mais dont les arguments sont en français (`annee`, `mois`, `jour`)
2. Idem avec la fonction `datetime` du module `datetime` (il faudra rajouter les arguments optionels `heure`, `minute`, `seconde`)

```
def date_fr(annee, mois, jour):
    return date(annee, mois, jour)

print(date_fr(annee=2011, mois=6, jour=2))

def datetime_fr(annee, mois, jour, heure=0, minute=0, seconde=0):
    return datetime(annee, mois, jour, heure, minute, seconde)

print(datetime_fr(annee=2011, mois=6, jour=2, heure=15))
```

```
2011-06-02
2011-06-02 15:00:00
```

Polynômes

On considère un polynôme de degrés n : $a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$.

1. Ecrire une fonction `eval_polynome_troisieme_degres(x, a_0, a_1, a_2, a_3)` qui évalue un tel polynôme en x.
2. Ecrire une fonction qui permettra d'évaluer un polynôme de degré inférieur à 4 est que l'on pourra utiliser de la façon suivante :

   ```
   eval_polynome(x, 3, 4) # 3x + 4
   eval_polynome(x, a_0=2, a_2=4) # 4x^2 + 4
   ```

3. Faire en sorte que cette fonction marche pour n'importe quel degré.

```
def eval_polynome_troisieme_degres(x, a_0, a_1, a_2, a_3):
    return a_0 + a_1*x + a_2*x**2 + a_3*x**3

def eval_polynome(x, a_0=0, a_1=0, a_2=0, a_3=0, a_4=0):
    return a_0 + a_1*x + a_2*x**2 + a_3*x**3 + a_4**4

def eval_polynome(x, *a):
    output = 0
    for i, val_a in enumerate(a):
        output += val_a*x**i
    return output

x = 10
eval_polynome(x, 3, 4)
```

```
43
```

Equation du second degré

On souhaite trouver les solutions de l'équation : $ax^2 + bx + c = 0$. On rappelle que pour cela on calcule le discriminant $\Delta = b^2 - 4 * a * c$.

- Si $\Delta > 0$, les solutions sont $\frac{-b \pm \sqrt{Delta}}{2a}$
- Si $\Delta = 0$, il y a une solution (double) $\frac{-b}{2a}$
- Si $\Delta < 0$, les solutions sont $\frac{-b \pm i\sqrt{Delta}}{2a}$

1. Ecrire une fonction equation_second_degre qui donne les solutions.
2. Tester l'équation $x^2 - 3x + 2$
3. Ecrire une fonction qui résout une équation du premier degré.
4. Faire en sorte que cette fonction soit appelée lorsque $a = 0$.

```
from math import sqrt

def equation_seconde_degre(a, b, c):
    Delta = b**2 - 4*a*c
    if Delta>0:
        return (-b + sqrt(Delta))/(2*a), (-b - sqrt(Delta))/(2*a)
    if Delta==0:
        return -b/(2*a)
    if Delta<0:
        return (-b + sqrt(-Delta)*1J)/(2*a), (-b - sqrt(-Delta)*1J)/(2*a)

print(equation_seconde_degre(1, -3, 2))

def equation_premier_degre(a, b):
    return -b/a

def equation_seconde_degre(a, b, c):
    if a==0:
        return equation_premier_degra(b, c)
    Delta = b**2 - 4*a*c
    if Delta>0:
        return (-b + sqrt(Delta))/(2*a), (-b - sqrt(Delta))/(2*a)
    if Delta==0:
        return -b/(2*a)
    if Delta<0:
        return (-b + sqrt(-Delta)*1J)/(2*a), (-b - sqrt(-Delta)*1J)/(2*a)

# Dans cet exercice, les else ou elif sont inutiles.
```

```
(2.0, 1.0)
```

Coordonnée polaire d'un nombre complexe

On considère une nombre complexe z et sa représentation polaire : $z = re^{i\theta}$

1. Ecrire une fonction qui à partir de r et θ renvoie z

```
from math import cos, sin, pi

def from_polar(r, theta):
    return r*cos(theta) + 1J* r*sin(theta)

from_polar(1, pi/2)
```

```
(6.123233995736766e-17+1j)
```

Nombres premiers

Ecrire une fonction qui renvoie True si un nombre est premier et False sinon. Pour cela, on testera si le nombre est divisible par les entiers successifs à partir de 2.

On admet que si n n'est pas premier, alors il existe un entier $p \geq 2$ tel que $p^2 \leq n$ qui est un diviseur de n.

1. Ecrire la fonction à l'aide d'une boucle.
2. Combien y a t-il d'années au XXIème siècle qui sont des nombres premiers ?
3. Si vous n'êtes pas convaincu que c'est mieux avec des fonctions, faites le sans…

```
def est_premier(n):
    p = 2
    while p**2<=n:
        if n%p==0:
            return False
        p += 1
    return True

est_premier(7)

count_premier = 0
for annee in range(2001, 2101):
    if est_premier(annee):
        count_premier += 1
print(f'Il y a {count_premier} années premières aux XXI siècle')
```

Mention

Ecrire une fonction qui à partir de la note sur 20 donne la mention

```
def mention(note):
    if note>=18:
        return 'Felicitation'
    if note>=16:
        return 'Très bien'
    if note>=14:
        return "Bien"
    if note>=12:
```

(continues on next page)

(continued from previous page)

```
        return "Assez bien"
    return 'Pas de mention'
```

3.2 Exercices sur les nombres

Fonctions mathématiques

- Est ce que la fonction log est le logarithme décimal ou népérien ?
- Calculer $x = \sqrt{2}$ puis calculer x^2. Que se passe-t-il ?
- Calculer arccos $\frac{\sqrt{2}}{2}$ et comparer à sa valeur théorique.

```
from math import log

print(log(10))
```

```
2.302585092994046
```

```
from math import sqrt
sqrt(2)**2
```

```
2.0000000000000004
```

```
from math import acos, pi

print(acos(sqrt(2)/2))
print(pi/4)
```

```
0.7853981633974483
0.7853981633974483
```

Constante de structure fine

La constante de structure fine est définie en physique comme étant égale à

$$\alpha = \frac{e^2}{2\epsilon_0 hc}$$

où

- e est la charge de l'électron et vaut $1.602176634 \times 10^{-19} C$
- h est la constante de Planck et vaut $6.626\,070\,15 \times 10^{-34} Js$
- ϵ_0 la permitivité du vide et vaut $8.8541878128 \times 10^{-12} F/m$
- c la célérité de la lumière dans le vide, $c = 299792458 m/s$

Définissez en Python les variables `e`, `hbar`, `epsilon_0` et `c`. Calculez α et $1/\alpha$

```
from math import pi

e = 1.602176634E-19
h = 6.62607015E-34
epsilon_0 = 8.8541878128E-12
c = 299792458


alpha = e**2/(2*epsilon_0*h*c)
print(1/alpha)
```

```
137.0359990841083
```

Précision des nombres

- Soit $x = 1$ et $\epsilon = 10^{-15}$. Calculez $y = x + \epsilon$ et ensuite $y - x$.
- Pourquoi le résultat est différent de 10^{-15}.
- Que vaut cette valeur ?

```
x = 1
epsilon = 1E-15
y = x + epsilon
print(y - x)
```

```
1.1102230246251565e-15
```

```
print(5*2**-52)
```

```
1.1102230246251565e-15
```

Calcul d'une dérivée

On considère une fonction $f(x)$. On rappelle que la dérivée peut se définir comme

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Pour calculer numériquement une dérivée, il faut évaluer la limite en prenant une valeur 'petite' de ϵ.

On prendra comme exemple $f(x) = \sin(x)$.

- Calculer numériquement la dérivée de f en $\pi/4$ en utilisant la formule pour $\epsilon = 10^{-6}$.
- Comparer à la valeur théorique $\cos(x)$ pour différentes valeurs de ϵ que l'on prendra comme puissance de 10 ($\epsilon = 10^{-n}$). Que se passe-t-il si ϵ est trop petit ? trop grand ?
- Ecrire la fonction `sin_prime(x, epsilon)` qui calcule la dérivée de sin en x
- Ecrire une fonction qui prend une fonction quelconque et renvoie la fonction dérivée.

```
from math import sin, pi, cos

x = pi/4
epsilon = 1E-11

d = (sin(x + epsilon) - sin(x))/epsilon - cos(x)
print(d)

for n in range(1, 15):
    epsilon = 10**(-n)
    d = (sin(x + epsilon) - sin(x))/epsilon - cos(x)
    print('n=', n, d)

# Lorsque epsilon est trop petit, il y a des erreurs d'arrondi. Lorsque epsilon est␣
↪trop grand, nous
# sommes loin de la limite
```

```
5.365427460879424e-06
n= 1 -0.03650380828255784
n= 2 -0.0035472894973379576
n= 3 -0.0003536712121802177
n= 4 -3.535651724428934e-05
n= 5 -3.5355413900983734e-06
n= 6 -3.5344236126721995e-07
n= 7 -3.5807553921962665e-08
n= 8 3.050251939917814e-09
n= 9 3.635694267867251e-08
n= 10 9.245353623787977e-07
n= 11 5.365427460879424e-06
n= 12 -5.7368027853721415e-06
n= 13 0.00010528549967714351
n= 14 0.003435954573552613
```

```
def sin_prime(x, epsilon):
    return (sin(x + epsilon) - sin(x))/epsilon

def derivee(f, epsilon):
    def f_prime(x):
        return (f(x+epsilon) - f(x))/epsilon
    return f_prime

derivee(sin, epsilon=1E-8)(x) - cos(x)
```

```
3.050251939917814e-09
```

Nombre complexe

- Ecrire une fonction qui calcule le module d'un nombre complexe z
- Ecrire une fonction qui à partir de r et θ renvoie le nombre $z = re^{i\theta} = r\cos(\theta) + ir\sin(\theta)$

3.3 Exercices sur les conteneurs

Manipulation des listes

On considère la liste [1, 5, 3, 5, 6, 2]

1. Écrire une fonction 'somme' qui renvoie la somme des éléments d'une liste de nombres. On fera explicitement la boucle for.
2. Écrire une fonction 'maximum' qui renvoie le maximum des éléments d'une liste de nombres. On fera explicitement la boucle for.
3. Écrire une fonction 'arg_maximum' qui renvoie l'indice du maximum d'une liste de nombres. On fera explicitement la boucle for.
4. Écrire une fonction 'trouve' qui renvoie l'indice correspondant à l'argument. On fera explicitement la boucle for.
5. Comment répondre aux questions 1, 2, 3, 4 en utilisant des fonctions déjà existantes ?

```
l = [1, 5, 3, 5, 6, 2]

def somme(l):
    output = 0
    for val in l:
        output += val
    return output

def maximum(l):
    if len(l)==0:
        return None
    output = l[0]
    for val in l[1:]:
        if val>output:
            output = val
    return output

def arg_maximum(l):
    if len(l)==0:
        return None
    output = l[0]
    i_max = 0
    for i, val in enumerate(l):
        if val>output:
            output = val
            i_max = i
    return i_max

def trouve(l, valeur_a_trouver):
    for i, val in enumerate(l):
        if valeur_a_trouver==val:
            return i
    else:
```

(continues on next page)

(continued from previous page)

```
        return None

# On peut utiliser sum, max et l.index
```

Liste comprehension

1. Créer une liste nomée `nombres` contenant les entiers de 0 à 9 inclus
2. Créer une liste contenant la racine carré des éléments de `nombres` (on utilisera une comprehension de liste)
3. Créer une liste contenant tous les nombres pairs de la listes `nombres` (on utilisera une comprehension de liste)
4. Toujours en utilisant un comprehension de liste, considérant deux listes `l1` et `l2`, créer une nouvelle liste contenant les couples pris deux à deux de l1 et l2. On supposera que les deux liste ont la même longueur. Quelle fonction python fait la même chose ?
5. En utilisant la fonction de la question 4 et la liste de la question 2 vérifier que l'on a bien $y = x^2$ pour chaque élément.

```
import math
nombres = list(range(10))
racine_nombres = [math.sqrt(nb) for nb in nombres]
nb_pairs = [nb for nb in nombres if nb%2==0]
```

```
liste1 = ['A', 'B', 'C']
liste2 = [10, 4, 24]
def f(l1, l2):
    return [(l1[i], l2[i]) for i in range(len(l1))]

f(liste1, liste2)
```

```
[('A', 10), ('B', 4), ('C', 24)]
```

```
for a, racine_a in f(nombres, racine_nombres):
    if not math.isclose(a, racine_a**2):
        print(f'Problème avec {a} et {racine_a}')
```

Exercice de base sur les dictionnaires

1. Tout d'abord, nous allons créer un petit dictionnaire qui contient des informations sur un étudiant. Utilisons les clés et valeurs suivantes en exemple :
   - 'nom': 'Jean Dupont'
   - 'âge': 20
   - 'filière': 'Informatique'
2. Modifier l'age pour qu'il soit égal à 21
3. Afficher le genre de l'étudiant si il possède une telle clé sinon afficher un message inquant que l'on ne connait pas son genre.

```
etudiant = {"nom": "Jean Dupont",
            "age": 20,
            "filière": "Informatique"}

etudiant['age'] = 21

if 'genre' in etudiant:
    print("Le gende de l'étudiant est", etudiant['genre'])
else:
    print("L'étudiant n'a pas de genre spécifié")
```

```
L'étudiant n'a pas de genre spécifié
```

Exercice sur les ensembles

La fonction chr permet de convertir un code ASCII en un caractère. La liste des lettres majuscules peut être obtenue à partir de la commande suivante :

```
liste_majuscules = [chr(65+i) for i in range(26)]
```

On souhaite vérifier qu'un mot de passe entré par un utilisateur est sufisament compliqué. Voici les règles :

- Il doit contenir 12 caractères différents
- Il doit contenir au moins 2 majuscules différentes
- Il doit contenir au moins un caractère de ponctuation `.,;:!?`
- Il ne doit pas contenir d'espace

Ecrire une fonction qui renvoie True si toutes les conditions sont vérifiées et False sinon

```
majuscules = set(majuscules)
ponctuations = set(',.;:!?')

def verifie_mot_de_passe(mdp):
    mdp = set(mdp)
    if len(mdp)<12:
        return False
    if len(mdp&majuscules)<2:
        return False
    if not ponctuations&mdp:
        return False
    if ' ' in mdp:
        return False
    return True
```

```
verifie_mot_de_passe('edsfJe:;ZErb_4')
```

```
True
```

3.4 Chaînes de caractères

Liste de prix

Voici un liste de prix sous forme d'un dictionnaire. Afficher la liste de prix de la façon suivante :

```
price_liste = {'tomates':3.4,
     'pommes':2.49,
     'oignons':1.45}
```

```
for nom, prix in price_liste.items():
    prix = f'{prix:5.2f}'
    prix = prix.replace('.', '€')
    print(f'{nom:>10s}: {prix}')
```

```
   tomates:  3€40
    pommes:  2€49
   oignons:  1€45
```

Unicode

Afficher toutes les lettres greques de α à ω. (On pourra copier coller ces lettres pour avoir leurs unicodes)

```
debut = ord('α')
fin = ord('ω')
print(' '.join([chr(i) for i in range(debut, fin+1)]))
```

```
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ χ ψ ω
```

Combien La Fontaine a t-il utilisé de mots différents dans ses fables ?

Les lignes suivantes permettent de télécharger l'ensemble des fables de la Fontaine.

- Combien y a-t-il de mots différents ? On pourra d'abord remplacer toutes les ponctuations par des espaces, puis créer une liste de mots que l'on mettra en minuscule. On créera ensuite un ensemble dont on regardera la taille.
- Quelle est le mot le plus long ?

```
import requests

s = requests.get('https://www.gutenberg.org/cache/epub/56327/pg56327.txt').text

ponctutation_list = '?,.;:!\'\"-[]()/«»'
```

```
for ponctuation in ponctutation_list:
    s = s.replace(ponctuation, ' ')

liste_mots = s.split()
liste_mots = [item.strip().lower() for item in liste_mots]
ensemble_mots = set(liste_mots)

len(ensemble_mots)
```

```
10496
```

```
ensemble_mots = sorted(list(ensemble_mots))
tailles = [len(mot) for mot in ensemble_mots]
```

```
taille_max = max(tailles)
[mot for mot, taille in zip(ensemble_mots, tailles) if taille==taille_max]
```

```
['batrachomyomachie']
```

3.5 Objets

Vecteur

Créer une classe Vecteur3D. Chaque vecteur aura 3 attributs : x, y, z

- Écrire une méthode norme qui renvoie la norme.
- Écrire la méthode `__add__` pour faire la somme entre deux veteurs
- Écrire la méthode `__mul__` pour faire soit le produit par un scalaire ($2\vec{u}$) ou le produit scalaire ($\vec{u} \cdot \vec{v}$).

Bibliographie

Un livre est décrit par son titre, auteur et année de publication (pour faire les choses simplements). Écrire une classe `Livre` qui enregistre ces informations. Ecrire la méthode `__repr__` et `__str__`.

Une bibliographe est une liste de livre. Écrire la classe `Bibliographie` qui enregistre une liste de livre (on stockera la liste de livre sous forme d'une liste qui sera un attribut de la bibliographie).

L'objectif final est de pouvoir faire ceci ::

```
livre1 = Livre("A very nice book", "F. Dupont", 2014)
livre2 = Livre("A very smart book", "A. Einstein", 1923)
livre3 = Livre("A very stupid comic", "D. Duck", 1937)

bibliographie = Bibliographie([book1, book2, book3])
```

Maintenant que tout est fait sous forme d'objet, on peut imaginer écrire plusieurs méthode :

- Écrire une méthode `filter_by_year` qui fait une nouvelle bibliographie ne contenant que les livres d'une année donée.
- Écrire une méthode `to_html` qui formate correctement la bibliographie. La méthode de la classe Bibliographie devra appeler une méthode pour chaque Livre.

Et en HTML ::

```
<table>
    <thead>
        <tr> <th>Auteur</th><th>Titre</th><th>Année</th></tr>
    </thead>
    <tbody>
       <tr><td>F. Dupont</td><td>2014</td><td>A very nice book</td></tr>
       <tr><td>A. Einstein</td><td>1923</td><td>A very smart book</td></tr>
```

(continues on next page)

(continued from previous page)

```
        <tr><td>D. Duck</td><td>1937</td><td>A very stupid comic</td></tr>
    </tbody>
</table>
```

Remarque : si un objet possède une méthode `_repr_html_`, alors le jupyter notebook utilsera automatiquement la représentation en HTML. Rajouter cette méthode (qui appelera to_html).

```
class Livre():
    def __init__(self, titre, auteur, annee):
        self.titre = titre
        self.auteur = auteur
        self.annee = annee

    def __repr__(self):
        return f"Livre({self.titre!r}, {self.auteur!r}, {self.annee!r})"

    def to_html_table_line(self):
        return f"<tr><td>{self.auteur}</td><td>{self.titre}</td><td>{self.annee}</td>
↪</tr>"


bibio_html_template="""<table>
<thead>
<tr><th>Auteur</th><th>Titre</th><th>Année</th></tr>
</thead>
<tbody>
{content}
</tbody>
</table>"""


class Bibliographie():
    def __init__(self, liste_des_livres):
        self._liste_livres = liste_des_livres

    def __getitem__(self, key):
        return self._liste_livres[key]

    def __repr__(self):
        return f"Bibliographie({self._liste_livres!r})"

    def to_html_table(self):
        content = '\n'.join([livre.to_html_table_line() for livre in self._liste_
↪livres])
        return bibio_html_template.format(content=content)

    _repr_html_ = to_html_table
```

```
livre1 = Livre("A very nice book", "F. Dupont", 2014)
livre2 = Livre("A very smart book", "A. Einstein", 1923)
livre3 = Livre("A very stupid comic", "D. Duck", 1937)

bibliographie = Bibliographie([livre1, livre2, livre3])
bibliographie
```

```
Bibliographie([Livre('A very nice book', 'F. Dupont', 2014), Livre('A very smart book
↪', 'A. Einstein', 1923), Livre('A very stupid comic', 'D. Duck', 1937)])
```

Système de calcul formel

Cet exercice est à but purement pédagogique. Pour utiliser un système de calcul formel sous Python, la librairie `sympy` *existe et fonctionnera bien mieux que ce que l'on va faire !*

L'objectif de ce TD est de réaliser un système de calcul formel qui permettra de manipuler des expressions algébriques simples et de réaliser des opérations simples. Par exemple, on souhaite pouvoir effectuer ::

```
x = Symbol('x')
y = Symbol('y')

s = 2*x*y + sin(x)*y

print(s.diff(x)) # Dérivée par rapport à x
```

Chaque expression sera représentée par un arbre. Les feuilles de l'arbre seront soit les symboles soit les constantes numériques. Les noeuds seront des fonctions à un ou plusieurs argument (sinus, somme, opposé, …). Le nom de la classe du noeud désignera la fonction. Les "enfants" du noeud seront les arguments de la fonction. Par exemple l'expression ci dessus correspondra à l'objet suivant ::

```
# sA : 2*x*y
sA = Prod(Prod(Number(2), Symbol('x')), Symbol('y'))
# sB : sin(x)*y
sB = Prod(Sin(Symbol('x')), Symbol('y'))

s = Sum(sA, sB)
```

Structure du programme

Voici la structure de base ::

```
class Expr(object):
    pass

class Node(Expr):
    pass

class Leave(Expr):
    pass
```

Pour les feuilles ::

```
class Symbol(Leave):
    pass

class Number(Leave):
    pass
```

Ensuite on définit les fonctions ::

```
class Function(Node):
    """ Function with an arbitrary number of arguments """
    pass
```

Les opérateurs sont des fonctions comme les autres, mais elle seront simplement affichées différemment ::

```
class BinaryOperator(Function):
    pass

class Sum(BinaryOperator):
    pass
# Idem pour Sub, Div, Prod, Pow

class UnitaryOperator(Function):
    pass

class Neg(UnitaryOperateor):
    pass
```

Les fonction mathématiques, qui prennent un seul argument ::

```
class MathFunction(Function):
    pass

class Sin(MathFunction):
    pass
```

Questions

On va procéder étape par étape. Il sera plus facile de commencer par les feuilles avant d'écrire la structure globale.

1. Ecrire le __init__ de la classe Symbol et Number

```
class Expr(object):
    def __add__(self, other):
        return Sum(self, other)

    def __mul__(self, other):
        return Prod(self, other)

    def __repr__(self):
        return self.display()

class Node(Expr):
    pass

class Leave(Expr):
    pass

class Symbol(Leave):
    pass

class Number(Leave):
    pass

class Function(Node):
```

(continues on next page)

(continued from previous page)

```
    """ Function with an arbitrary number of arguments """
    pass

class BinaryOperator(Function):
    pass

class Sum(BinaryOperator):
    pass
# Idem pour Sub, Div, Prod, Pow

class UnitaryOperator(Function):
    pass

class Neg(UnitaryOperator):
    pass

class MathFunction(Function):
    pass

class Sin(MathFunction):
    pass
```

```
import numbers

class Symbol(Leave):
    def __init__(self, symbole):
        self.symb = symbole

class Number(Leave):
    def __init__(self, nombre):
        if not isinstance(nombre, numbers.Number):
            raise ValueError
        self.nbre = nombre
```

2. Ecrire une méthode `display` sur ces classes afin de renvoyer une chaîne de caractère contenant le symbole ou le nombre

```
class Symbol(Leave):
    def __init__(self, symbole):
        self.symb = symbole

    def display(self, *args):
        return self.symb

class Number(Leave):
    def __init__(self, nombre):
        if not isinstance(nombre, numbers.Number):
            raise ValueError
        self.nbre = nombre

    def display(self):
        return str(self.nbre)
```

3. Ecrire le `__init__` de la class Sin ainsi que le display. Le display devra appeler le display de l'argument. Par exemple ceci devra fonctionner ::

```
>>> x = Symbol('x')
>>> Sin(x).display()
sin(x)
>>> Sin(Sin(x)).display()
sin(sin(x))
```

```
class Sin(MathFunction):
    def __init__(self, arg):
        self.arg = arg

    def display(self):
        return f'Sin({self.arg.display()})'

x = Symbol('x')
Sin(Sin(x)).display()
```

```
'Sin(Sin(x))'
```

4. Généraliser le **init** et le display de `Sin` afin de le mettre dans la class MathFunction. On rajoutera un attribut de classe à chaque sous classe de MathFunction ::

```
class Sin(MathFunction):
    funtion_name = 'sin'
```

```
class MathFunction(Function):
    def display(self):
        return f'{self.function_name}({self.arg.display()})'


class Sin(MathFunction):
    function_name = 'sin'
    def __init__(self, arg):
        self.arg = arg


x = Symbol('x')
Sin(Sin(x)).display()
```

```
'sin(sin(x))'
```

5. Faire de même pour les opérateurs binaires. On pourra commencer par simplement le faire pour Sum, puis généraliser avec un attribut de classe ::

```
class Sum(BinaryOperator):
    operator_name = '+'
```

```
class Function(Node):
    """ Function with an arbitrary number of arguments """
    def __init__(self, *args):
        self.args = args

class BinaryOperator(Function):
    def __init__(self, arg1, arg2):
        self.args = (arg1, arg2)
```

(continues on next page)

(continued from previous page)

```
    def display(self):
        return f'({self.args[0].display()}) {self.operator_name} ({self.args[1].
↪display()})'

class Prod(BinaryOperator):
    operator_name = '*'


class Sum(BinaryOperator):
    operator_name = '+'

x = Symbol('x')
y = Symbol('y')
Sum(x, Sin(Prod(x, y))).display()
```

```
'(x) + (sin((x) * (y)))'
```

6. A ce stade quelque chose comme ceci devrait fonctionner ::

```
x = Symbol('x')
y = Symbol('y')
Sum(x, Sin(Prod(x, y)))
```

Rajouter les méthodes `__add__`, `__mul__`, etc à la classe Expr afin de pouvoir écrire :

```
>>> x + Sin(x*y)
```

```
x + Sin(x*y)
```

```
(x) + (sin((x) * (y)))
```

7. Ecrire les méthodes `evaluate` afin de calculer la valeur numérique d'une expression. Cette méthode fonctionnera de la sorte :

```
>>> expr = x + Sin(x*y)
>>> expr.evaluate(x=1, y=3)
```

On aura donc le protocole suivant ::

```
def evaluate(self, **kwd):
    pass
```

Le dictionnaire kwd sera passé récursivement jusqu'aux feuilles et sera utilisé pour évaluer les symboles.

Les opérateurs binaires numériques sont définis dans le module `operator` et les fonctions dans le module `math`. Afin de factoriser le code, on rajoutera donc simplement un attribut de classe du type `operator_function = operator.add` pour les opérateurs binaires et `math_function = math.sin` pour les fonctions.

```
import math
import numbers
import operator

class Expr(object):
    def __add__(self, other):
        return Sum(self, other)
```

(continues on next page)

(continued from previous page)

```
    def __mul__(self, other):
        return Prod(self, other)

    def __repr__(self):
        return self.display()

class Node(Expr):
    pass

class Leave(Expr):
    pass


class UnitaryOperator(Function):
    pass

class Neg(UnitaryOperator):
    pass


class Symbol(Leave):
    def __init__(self, symbole):
        self.symb = symbole

    def display(self, *args):
        return self.symb

    def evaluate(self, **kwd):
        try:
            return kwd[self.symb]
        except KeyError:
            raise Exception("La valeur de {} n'est pas définie".format(self.symb))

class Number(Leave):
    def __init__(self, nombre):
        if not isinstance(nombre, numbers.Number):
            raise ValueError
        self.nbre = nombre

    def display(self):
        return str(self.nbre)

    def evaluate(self, **kwd):
        return self.nbre

class Function(Node):
    """ Function with an arbitrary number of arguments """
    def __init__(self, *args):
        self.args = args

    def evaluate(self, **kwd):
        evaluated_args = [elm.evaluate(**kwd) for elm in self.args]
        return self.math_function(*evaluated_args)

class BinaryOperator(Function):
    def __init__(self, arg1, arg2):
```

(continues on next page)

(continued from previous page)

```
        self.args = (arg1, arg2)

    def display(self):
        return f'({self.args[0].display()}) {self.operator_name} ({self.args[1].
↪display()})'

class Prod(BinaryOperator):
    operator_name = '*'
    math_function = operator.mul

class Sum(BinaryOperator):
    operator_name = '+'
    math_function = operator.add

class MathFunction(Function):
    def display(self):
        return f'{self.function_name}({self.args[0].display()})'


class Sin(MathFunction):
    function_name = 'sin'
    math_function = math.sin
    def __init__(self, arg):
        self.args = (arg,)
```

```
x = Symbol('x')
y = Symbol('y')
expr = x + Sin(x*y)
print(expr)
expr.evaluate(x=2, y=4.5)
```

```
(x) + (sin((x) * (y)))
```

```
2.4121184852417565
```

8. Maintenant que vous avez compris le principe, il devrait être facile d'écrire une méthode `diff` qui effectue la dérivée par rapport à une variable !

9. Reste à simplifier les expressions. Une technique consiste à créer des règles de simplifications sous forme de méthode que l'on regroupe ensuite dans une liste ::

```
class Sum(BinaryOperator):
    operator_name = '+'
    operator_function = operator.add

    def simplication_de_deux_nombres(self):
        if isinstance(self.arg1, Number) and
                        isinstance(self.arg2, Number):
            return Number(self.arg1.value + self.arg2.value)

    def simplication_addition_avec_zero(self):
        pass

    liste_simplication = ['simplication_de_deux_nombres',
                          'simplication_addition_avec_zero']
```

Ensuite, il faut réussir à appeler correctement et de façon recursive ces méthodes…

10. Pour l'affichage des opérateurs binaires, les règles de priorité peuvent être utilisées pour éviter de mettre trop de parenthèses. Par exemple, dans le cas `a*(b+c)`, la multiplication appelle le display de l'addition. Comme elle est prioritaire, l'addition va renvoyer le résulat avec des parenthèses. Dans le cas inverse `a + b*c`, c'est inutile. Il faut donc que le display d'un opérateur passe sa priorité à ses enfants lors de l'appel de display. Implémenter ce principe.

```
import numbers
import operator
import math

class Expr(object):

    def binary_operator(self, other, operator):
        if isinstance(other, numbers.Number):
            other = Number(other)
        if isinstance(other, Expr):
            return operator(self, other)
        return NotImplemented

    def reversed_binary_operator(self, other, operator):
        if isinstance(other, numbers.Number):
            other = Number(other)
        if isinstance(other, Expr):
            return operator(other, self)
        return NotImplemented

    def __add__(self, other):
        return self.binary_operator(other, Sum)

    def __mul__(self, other):
        return self.binary_operator(other, Prod)

    def __truediv__(self, other):
        return self.binary_operator(other, Div)

    def __sub__(self, other):
        return self.binary_operator(other, Sub)

    def __radd__(self, other):
        return self.reversed_binary_operator(other, Sum)

    def __rmul__(self, other):
        return self.reversed_binary_operator(other, Prod)

    def __rtruediv__(self, other):
        return self.reversed_binary_operator(other, Div)

    def __rsub__(self, other):
        return self.reversed_binary_operator(other, Sub)

    def __neg__(self):
        return Neg(self)

    def __repr__(self):
        return self.display()
```

(continues on next page)

(continued from previous page)

```
    def diff(self, var):
        out = self._diff(var)
        return out.simplify()

class Node(Expr):
    pass

class Leave(Expr):
    def simplify(self):
        return self

class Symbol(Leave):
    def __init__(self, symbole):
        self.symb = symbole

    def display(self, *args):
        return self.symb

    def evaluate(self, **kwd):
        try:
            return kwd[self.symb]
        except KeyError:
            raise Exception("La valeur de {} n'est pas définie".format(self.symb))

    def __eq__(self, other):
        if not type(self)==type(other):
            return False
        return self.symb==other.symb

    def _diff(self, var):
        if self==var:
            return Number(1)
        return Number(0)

class Number(Leave):
    def __init__(self, nombre):
        if not isinstance(nombre, numbers.Number):
            raise ValueError
        self.nbre = nombre

    def display(self, *args):
        return str(self.nbre)

    def evaluate(self, **kwd):
        return self.nbre

    def __eq__(self, other):
        if isinstance(other, numbers.Number):
            other = Number(other)
        if isinstance(other, Number):
            return other.nbre==self.nbre
        return False

    def _diff(self, var):
        return Number(0)
```

(continues on next page)

(continued from previous page)

```
class Function(Node):
    """ Function with an arbitrary number of arguments """
    def __init__(self, *args):
        self.args = args

    def evaluate(self, **kwd):
        evaluated_args = [elm.evaluate(**kwd) for elm in self.args]
        return self.math_function(*evaluated_args)

    def __eq__(self, other):
        if not type(self)==type(other):
            return False
        return self.args==other.args

    def _diff(self, var):
        partial_derivative = getattr(self, 'partial_derivative', None)
        if partial_derivative is None:
            raise NotImplementedError('Cannot derivate function {self.__class__}.'.
↪format(self=self))
        if len(self.args)==0:
            return Number(0)
        out = self.args[0].diff(var)*partial_derivative[0](*self.args)
        for deriv, arg in zip(partial_derivative[1:], self.args[1:]):
            out = out + arg.diff(var)*deriv(*self.args)
        return out

    liste_simplification = []
    def simplify(self):
        out = type(self)(*[elm.simplify() for elm in self.args])
        for elm in self.liste_simplification:
            tmp = getattr(out, elm)()
            if tmp is not None:
                return tmp.simplify()
        return out


class BinaryOperator(Function):
    commutative=False
    def display(self, parent_priority=0):
        if parent_priority>self.priority:
            fmt_str = '({} {} {})'
        else:
            fmt_str = '{} {} {}'
        return fmt_str.format(self.args[0].display(self.priority),
                self.operator_name, self.args[1].display(self.priority))

    def __eq__(self, other):
        if not type(self)==type(other):
            return False
        if self.args==other.args:
            return True
        if self.commutative:
            return self.args==other.args[::-1]
        return False

class Sum(BinaryOperator):
    priority = 0
```

(continues on next page)

(continued from previous page)

```
    operator_name = '+'
    math_function=operator.add
    commutative=True
    partial_derivative = (lambda x, y:1, lambda x, y:1)


    def simplication_de_deux_nombres(self):
        if isinstance(self.args[0], Number) and isinstance(self.args[1], Number):
            return Number(self.args[0].nbre + self.args[1].nbre)

    def simplication_addition_avec_zero(self):
        if self.args[0]==0:
            return self.args[1]
        if self.args[1]==0:
            return self.args[0]

    def simplification_identique(self):
        if self.args[1]==self.args[0]:
            return 2*self.args[0]

    liste_simplification = ['simplication_de_deux_nombres',
        'simplication_addition_avec_zero', 'simplification_identique']

class Prod(BinaryOperator):
    priority = 1
    operator_name = '*'
    math_function=operator.mul
    commutative=True
    partial_derivative = (lambda x, y:y, lambda x, y:x)

    def simplication_de_deux_nombres(self):
        if isinstance(self.args[0], Number) and isinstance(self.args[1], Number):
            return Number(self.args[0].nbre * self.args[1].nbre)


    def simplication_multiplication_par_un(self):
        if self.args[0]==1:
            return self.args[1]
        if self.args[1]==1:
            return self.args[0]

    def simplication_multiplication_par_zero(self):
        if self.args[0]==0 or self.args[1]==0:
            return Number(0)

    liste_simplification = ['simplication_de_deux_nombres',
        'simplication_multiplication_par_un', 'simplication_multiplication_par_zero']


class Div(BinaryOperator):
    priority = 2
    operator_name = '/'
    math_function=operator.truediv
    partial_derivative = (lambda x, y:1/y, lambda x, y:-x/(y*y))

class Sub(BinaryOperator):
    priority = 0.5
```

(continues on next page)

(continued from previous page)

```
    operator_name = '-'
    math_function=operator.sub
    partial_derivative = (lambda x, y:1, lambda x, y:-1)


class UnitaryOperator(Function):
    def display(self, parent_priority=0):
        if parent_priority>self.priority:
            fmt_str = '({}{})'
        else:
            fmt_str = '{}{}'
        return fmt_str.format(self.unitary_symbol,
                              self.args[0].display(self.priority))

class Neg(UnitaryOperator):
    priority = 0.5
    unitary_symbol = "-"
    math_function = operator.neg
    partial_derivative = (lambda x:-1,)

class MathFunction(Function):
    def display(self, *args):
        return '{}({})'.format(self.function_name, self.args[0].display())


class Sin(MathFunction):
    function_name = 'sin'
    math_function = math.sin
    partial_derivative = (lambda x:cos(x),)


class Cos(MathFunction):
    function_name = 'cos'
    math_function = math.cos
    partial_derivative = (lambda x:-sin(x),)


sin = Sin
cos = Cos
```

```
x = Symbol('x')
sin(2*x).diff(x).simplify()
```

```
2 * cos(2 * x)
```

3.6 Utilisation de numpy

Calcul Monte-Carlo de la surface d'un disque

On rappelle que la fonction np.random.rand permet de créer une distribution uniforme de points entre 0 et 1.

1. Créer des tableaux X et Y de variables uniforméments réparties entre -1 et 1
2. Evaluer la probabilité que le point de coordonnées (x, y) soit dans le cercle unité ?
3. En déduire une estimation ed π
4. De la même façon, évaluer le volume d'une sphère en dimension 13

```
N = 1000000

X = 2*(np.random.rand(N)-.5)
Y = 2*(np.random.rand(N)-.5)

R = np.sqrt(X**2 + Y**2)
print('Estimation de pi', np.mean(R<1)*4)
```

```
Estimation de pi 3.143348
```

```
dim = 13

X = 2*(np.random.rand(dim, N)-.5)

R = np.sqrt(np.sum(X**2, axis=0))
print(np.mean(R<1))
```

```
0.00011
```

Calcul de la moyenne

Un tableau numpy contient les notes d'une classe. Chaque ligne correspond à un élève et chaque colone à un examen.

```
import numpy as np

N_eleves = 35
N_examens = 3

notes = np.random.rand(N_eleves, N_examens)*20
```

1. Calculer la moyenne de chaque élève. Calculer la moyenne pour chaque examen.
2. Calculer la moyenne de chaque élève, sachant que les coeficients pour les 3 examens sont de 1, 3 et 2.

```
moyenne_eleve = notes.mean(axis=1)
moyenne_exams = notes.mean(axis=0)

print(f'Moyenne eleves : {moyenne_eleve}')
print(f'Moyenne exams : {moyenne_exams}')
```

```
Moyenne eleves : [ 4.86475764 13.87157763 16.63133782 12.54131626  7.78354896 14.
↪91060888
 11.49341153  7.40491898  7.7408721  12.83294063  5.19142779  7.16174387
 11.72812901 16.37947513 13.78278386  9.49417344 14.78770611 10.32141726
  7.73516296 12.93635595 12.0172946   6.32059816 14.24920113  7.9846933
  9.34793038  4.97840815  0.93303272 11.61755392  4.74755408  6.75401493
  7.58238456 12.39258674 16.24376983  8.61322578  9.22967243]
Moyenne exams : [10.51405794 10.77392961  8.93534844]
```

```
moyenne_ponderee = (notes*np.array([1, 3, 2])[np.newaxis, :]).sum(axis=1)/6
print(f'Moyenne eleves : {moyenne_ponderee}')
```

```
Moyenne eleves : [ 4.8469425  14.99684316 17.5957476  13.28084804  7.38128184 14.
↪16213389
 10.37123986  5.34245749  7.19795841 14.43345158  6.1073791   5.54566383
 10.46290838 15.54873099 12.71400994  7.05525354 13.60650382 12.16244485
 10.58650973 14.71984854 13.36255598  4.45639921 14.35210974  9.04184847
 10.33807833  3.32534396  0.65691568 10.95163956  3.27687672  7.73253324
  8.73697947 12.65795077 16.75820541  8.44130762 11.91460335]
```