Le langage Python

Pierre Cladé

Mar 06, 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
```

(continues on next page)

(continued from previous page)

```
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 :
        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)
```

(continues on next page)

(continued from previous page)

```
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.

```python
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

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

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

```python
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.

```python
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 $0.xxxx$ 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.

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:

```python
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:

```python
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]
```

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 :

```python
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

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

Cette methode fonctionne aussi pour les dictionnaires ou les ensembles

```python
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}

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

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)
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
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 : $a_0 + a_1x + a_2x^2 + ... + a_nx^n$.

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 question 1, 2, 4 en utilisant des fonctions déjà existantes ?

Liste comprehension

1. Créer une liste nomée `nombres` contenant les entier de 0 à 9 inclus
2. Créer une liste contenant la racince carré des éléments de `nombres` (on utilisera une comprehension de liste)
3. Créer une lsite contenant tous les nombres pairs de la listes `nombres` (on utilisera une comprehension de liste)
4. Toujours en utilisant un comprehension de lsite, 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

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 un telle clé sinon afficher un message

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

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)
```

(continues on next page)

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
    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
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 ?

```python
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

```python
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. $r_2 = 1$)

```python
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 : $a_0 + a_1x + a_2x^2 + ... + a_nx^n$.

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:
```

(continues on next page)

(continued from previous page)

```
        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:
        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
```

```python
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 $z = re^{i\theta} = r\cos(\theta) + ir\sin(\theta)$