Objectif du cours

- Python 1 : Outils de bases pour les scientifiques
- Python 2 : Programmation orientée objet. Projet informatique

Python 1 :

- Quelques bases de Python
- Les tableaux numpy
- Statistiques, ajustement de courbes
- Transformée de Fourier
- Equations différentielles

Installation de Python

- Il existe plusieurs interpréteurs open source de Python. Le principal est CPython
- Il est fortement conseillé d'installer Anaconda et Python 3.8

Comment exécuter Python

- Jupyter notebook : très pratique (compte rendu de TP, cours). Pas pour des projets
- Spyder : éditeur de texte adapté à Python et l'environnement scientifique (à la matlab)
- IPython (inclus dans spyder) : terminal interactif. Faire des essais avant de copier dans un programme
- Console : interface graphique, test automatique, ...

Plan du cours

Aujourd'hui : Bases de python.

- Les types de bases de python
- Structures de contrôle
- Les fichiers
- Les exceptions

Types de données

Nombres

- entiers : pas de limite de tailles
- réels : flottant (float). Par défaut 64bits, précision relative de environ 10^{-15}
- Complexe : deux réels ; `a = 1 + 3J`
- +, -, *, /, ** (puissance)
- Modulo, division entière

In [ ]:

```
print(2**145)

x = 1
print((x + 1E-16) - x)

z = 1 + 3J
print(z.imag)
```

In [ ]:

```
print(5%2)
print(5//2)
```

Booléens

- `True` et `False`
- Comparaison : `>` , `>=` , `==` , `<=` , `<` , `!=`
- Opérations : and, or et not (attention aux priorités)
- "évaluation paresseuse"

In [ ]:

```
x = -1

x>0 and sqrt(x)>2
#if x>0:
#    return sqrt(x)
# else:
#    return False
```

Chaînes de caractères (str)

- Plusieurs façon d'écrire une chaine : '', '', """, ''' .
- Caractère spéciaux : \n (retour à la ligne), \t (tabulation)
- Unicode
- Concaténation
- Quelques méthodes sur les chaînes
- Formatage de chaîne de caractère

In [ ]:

```
s0 = 'Bonjour'
s1 = "Pierre"
s2 = "Aujourd'hui"
print(s1[2])
print(s1[2:5])
print(s0 + ' ' + s1) # Concaténation
```

In [ ]:

```
s3 = """Une chaine
sur plusieurs
lignes"""

s3
```

In [ ]:

```
print("Rayon \u03B3")
print("Rayon γ")
```

In [ ]:

```
# Quelques méthodes
s = 'Bonjour'
print(s.replace('o', 'r'))
s = "1;2;4;3"
print(s.split(';'))

s = 'monfichier.txt'
print(s.endswith('.txt'))
```

Formatage

Mettre une variable dans une chaîne de caractère

On utilise la méthode .format (ancienne syntaxe avec le %). On utilise des accolades.

On peut préciser:

- le nom de l'argument (keyword argument)
- le type d'écriture : entier (d), virgule fixe (f), notation scientifique
- la précision

Exemple {name:8.3f} :

- argument : name
- virgule fixe
- taille totale 8
- 3 chiffres après la virgule

In [ ]:

```
# Formatage
from math import pi
print('La valeur de pi est {:.3f}'.format(pi))
c = 299792458

print('La vitesse de la lumière est c={c:.3e} m/s'.format(c=c))

# Format string
print(f'La valeur de pi est {pi:.3f}')
```

Les listes

- Elles peuvent contenir n'importe quel type de donnée
- Les indices commencent par 0
- Index négatif : par la fin (modulo la taille de la liste)
- iterateur : par exemple `range` .

In [ ]:

```
l = ['Pierre', 34, 3.1415]
l[1]
l.append(25)
l
```

La méthode .append() modifie la liste. C'est toujours la même liste (comme on rajoute une page dans un classeur).

In [ ]:

```

```

Créer une liste à partir d'une autre liste

- Boucle `for`
- Liste comprehension []
- `list.append` : méthode de l'objet liste

In [ ]:

```
liste_initiale = [1, 34, 23, 2.]

liste_finale = []
for elm in liste_initiale:
    liste_finale.append(elm**2)
liste_finale

[elm**2 for elm in liste_initiale]
```

Parcourir une liste

- `for`
- enumerate
- zip

In [ ]:

```
ma_liste = ['Dupont', 'Martin', 'Dubois']
for name in ma_liste:
    print(name)

for i, name in enumerate(ma_liste):
    print(f'Le nom numéro {i} est {name}')
```

In [ ]:

```
liste_age = [12, 35, 23]
for age, name in zip(liste_age, ma_liste):
    print(f"{name} a {age} ans.")
```

Les n-uplets (tuples)

- Comme les listes sauf que l'on ne peut pas les modifier
- Défini avec des ()
- Utilisé pour regroupé une nombre connu de données : (x, y, z)
- Utilisé lorsqu'une fonction renvoie plusieurs valeurs

In [ ]:

```
def ma_fonction():
    return 1, 2, 3

a = ma_fonction()
print(type(a))
print(a[1])
```

In [ ]:

```
a, b, c = (1, 2, 5)
print(b)
```

In [ ]:

```

```

Les dictionnaires

- Conteneur (comme les listes ou tuple)
- Le contenu est indéxé par une clé qui est en général un nombre ou une chaine de caractère
- Explicit is better than implicit (paramètre ou résultat d'une expérience)

In [ ]:

```
parametres = {"N":100, 'l':30, "h":50, "g":9.81}
print(parametres['N'])

for key, val in parametres.items():
    print(f'La valeur de {key} est {val}')
```

In [ ]:

```
personne_1 = {"nom":"Dupont", "age":13}
personne_2 = {"nom":"Dubois", "age":34}

annuaire = [personne_1, personne_2]

for personne in annuaire :
    print(f"{personne['nom']} a {personne['age']} ans.")
```

Les ensembles

- Comme en mathématiques : ne contient pas deux fois le même élément
- Union |, intersection &

In [ ]:

```
import random
liste_aleatoire = [random.randint(1, 6) + random.randint(1, 6)
                   for _ in range(100)]
for val in set(liste_aleatoire):
    count = liste_aleatoire.count(val)
    print(f"Le nombre {val} est présent {count} fois")
```

Structures de contrôle

Boucles

- while
- for (voir les listes)

Tests

- if, elif, else

Fonctions

```
def nom_fonction(arg1, arg2, ...):
        ...
        return out1, out2
```

- Argument optionel, argument nommé
- Il peut y avoir plusieurs return au sein d'une fonction
- Documentation ≠ commentaire
- Une fonction qui ne renvoie rien renvoie None
- Utilsation d'un nombre arbitraire d'arguments (*args, **kwd)
- Utiliser plusieurs arguments dans un même conteneur

In [ ]:

```
def exponentielle(x, precision=1E-9):
    """ Calcule e**x

    Utilise le dl sum(x**n/n!)

    x : nombre dont on calcule exp
    precision : precision du calcul
    """
    result = 0
    n = 1
    term = 1 # Initial value
    while abs(term)>precision :
        result = result + term
        term = term * x/n
        n = n+1
    return result

print(2*exponentielle(2.1, 1E-6))
print(exponentielle(2.1))
print(exponentielle(2.1, precision=1E-6))
exponentielle(1)
```

In [ ]:

```
def afficher(x):
    print('x')

a = afficher(10)
print(a)
```

In [ ]:

```
def f(a, b, *args, **kwd):
    print(a)
    print(b)
    print(args)
    print(kwd)

f(1, 4, 5, 2, coucou='bonjour', alpha=14)
```

In [ ]:

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

params = (1, 4, 6)
print(f(1, *params))

params = {"a":1, "b":4, "c":6} #explicit is better than implicit!
print(f(1, **params))
```

Fonctions anonymes

```
lambda arg1, arg2 : expr
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Modules en Python

- Les modules sont des fichiers dont on peut importer les objets (en général des fonctions) qui y sont définis.
- Un module peut contenir d'autre modules
- Modules standards (ex: math)

In [ ]:

```
from math import exp
exp(2.1)
```

In [ ]:

```
import math
math.exp(1)
```

In [ ]:

```
# Syntaxe à éviter
from math import *
sin(pi/4)
```

In [ ]:

```
# Donner un nom plus simple
# Eviter sauf si tout le monde le fait
import numpy as np
```

Langage orienté objet

- Tout en Python est object
- Un objet contient des données
- La classe d'un objet défini ce qu'est l'objet
- La classe défini des fonctions qui permettent d'utiliser les données de l'objet ou de le modifier. Ce sont des méthodes

Exemples :

- liste, nombre complexe
- Tableaux numpy
- Oscilloscope

In [ ]:

```
l = [5,4,1,2]
l.insert(1, 'coucou') # modification de l'objet
print(l.index(2))
z = 1 + 2J
z.real # attribut de l'objet
z.conjugate() # methode qui renvoie un nouvel objet
```

In [ ]:

```
a = [1, 2, 3]
b = a.copy()
a.insert(0, 45)
print(b[0])
```

In [ ]:

```
a = [1, 2, 4]
b = a
a = 'Bonjour'
print(b[0])
```

In [ ]:

```

```

Variables globales/locales

- Dans une fonction, une variable est soit globale soit locale
- Si on assigne un objet à une variable dans une fonction, elle est automatiquement locale
- Modifier une liste ne rend pas la liste locale
- L'instrution `global` ne doit **jamais** être utilisée (sauf cas exceptionels)

In [ ]:

```
x = 1
def f1():
    print(x)
f1()
x = 3
f1()

def f2(x):
    print(x)
f2(3)

def f3():
    x = 4
    print(x)
f3()

def f4():
    print(x)
    x = 4
f4()
```

Les variables globales doivent être constantes :

- Constantes numériques
- Autres objets : fonctions, modules, ...

In [2]:

```
from math import sin, pi
def ma_fonction(x):
    if x==0:
        return 1
    return sin(pi*x)/(pi*x)
```

In [ ]:

```

```

In [ ]:

```

```

Les exceptions (erreurs)

- Il faut toujours lire et comprendre les erreurs
- En général python indique toujours l'endroit où se trouve l'erreur
- Sauf pour les "syntax error"

In [ ]:

```

```

Créer sa propre erreur

- raise Exception('Un message')
- Ne pas utiliser print(message)
- try: except

In [ ]:

```
# On lance une balle à la vitesse v_0 vers le haut
# Calculer le temps pour arriver au plafond de hauteur h
from math import sqrt
def temps_arrivee(h, v_0, g=9.81):
    Delta = v_0**2 - 2*g*h
    if Delta<0:
        raise Exception("La balle ne touche pas le plafond")
    return (v_0 - sqrt(Delta))/g

print(temps_arrivee(1, 10))
print(temps_arrivee(1, 1))
```

In [ ]:

```
from math import sqrt
def temps_arrivee(h, v_0, g=9.81):
    Delta = v_0**2 - 2*g*h
    try:
        return (v_0 - sqrt(Delta))/g
    except ValueError:
        raise Exception("La balle ne touche pas le plafond")
print(temps_arrivee(1, 10))
print(temps_arrivee(1, 1))
```

In [ ]:

```

```

Les fichiers

- Répertoire absolu et relatif
- Chemin d'accès
- Fichier texte (c'est une chaine de caractère)
- Lecture/écriture
- Format JSON

In [ ]:

```
!pwd
# Sous windows : c:\Users\login\
```

In [ ]:

```
import os
print(os.listdir('.'))
```

In [ ]:

```
file = '/tmp/test_python.txt'
fd = open(file, 'w')
fd.write('Bonjour\n')
fd.close()

with open(file, 'a') as fd:
    fd.write('Au revoir')
```

In [ ]:

```
with open(file) as fd:
    print(fd.read())
```

In [ ]:

```
with open(file) as fd:
    lines = fd.readlines()

for line in lines:
    print(line.strip()) # strip pour enlever le \n
```

JSON

- Permet d'enregistrer une liste ou un dictionnaire
- Fichier lisible par une humain

In [ ]:

```
import json

personne_1 = {"nom":"Dupont", "age":13}
personne_2 = {"nom":"Dubois", "age":34}

annuaire = [personne_1, personne_2]

file = '/tmp/test.json'
with open(file, "w") as f:
    json.dump(annuaire, f, indent=2)
```

In [ ]:

```
with open(file) as f:
    annuaire = json.load(f)
print(annuaire)
```

In [ ]:

```

```