PYTHON. **Conseils pratiques**

- Commencer par charger les bibliothèques (avec alias et en plus éventuellement sans)

```
from math import * ; import numpy as np
```

- Ne pas hésiter à utiliser des fonctions auxiliaires, en choisissant judicieusement les arguments.

Par exemple, pour étudier la suite d'intégrales $(J_n)_{n\in\mathbb{N}}$, avec $J_n = \int_0^{+\infty} \exp(-t^n)\ dt$, on peut écrire :

```
import scipy.integrate as integr
def f(n) :
     deg g :  return exp(-t**n)
     return g
def J(n) :  return integr.quad(f(n),0,inf)[0]
```

Commentaires : `f(n)` renvoie la fonction $t \longmapsto \exp(-t^n)$.
Ainsi, f est une fonction qui étant donné un paramètre n renvoie la fonction paramétrée associée.
La fonction `quad` renvoie le couple (valeur de l'intégrale, évaluation de l'erreur).
C'est pourquoi on renvoie uniquement le premier terme du couple.

Variante :

```
def J(n) :
     def f(t) :  return exp(-t**n)
     return integr.quad(f,0,inf)[0]
```

- Attention aux types des résultats renvoyés : par exemple, la fonction `fsolve` de résolution nulérique de l'équation $f(x) = 0$ renvoie la liste de solutions trouvées (même s'il y a une unique solution). De même, la fonciton `quad` renvoie un couple : utiliser `quad(f,a,b)[0]`pour évaluer $\int_a^b f$.

- Pour construire une matrice particulière, une solution fréquente consiste de partir de `zeros((n,p))`, puis de modifier les coefficients selon les valeurs souhaitées. Par exemple, pour définir une matrice aléatoire d'ordre n dont les coefficients sont des variables aléatoires indépendantes de loi de Bernoulli $\mathcal{B}(p)$.

```
import numpy as np ; import numpy.random as rd

def bernoulli(p) :
     return rd .binomial(1,p)
print(bernoulli(0.5)) # pour vérifier

def matrice(n,p) :
     A = np.zeros((n,n))
     for i in range(n) :
         for j in range(n) :
             A[i,j] = bernoulli(p)
     return A
print(matrice(5,0.5)) # pour vérifier
```

- Pour afficher un tableau X (array) avec seulement 2 chiffres significatifs (pour alléger), on utiliserait :

```
tronque = def(x) :   return round(x,2)
tronque = np.vectorize(tronque)
print(tronque(np.array([0.031875415,1.027331245])))
```

affiche `[0.03,1.03]`

PYTHON. Erreurs usuelles

Les erreurs de syntaxe sont détectées lors de la compilation. Même si une instruction ou une définition de fonction est syntaxiquement correcte, elle peut générer une erreur lors de son exécution (appelée souvent exception).

Lorsque le programme fonctionne mais ne renvoie pas le résultat attendu, il est conseillé de le tester dans des cas simples et/ou de faire afficher (via des instructions `print(...)`) les valeurs intermédiaires calculées.

Remarque : Un `"Warning"` n'est pas une erreur.

1) Erreurs de syntaxe :

" `invalid syntax` " , " `could not run code because it is incomplete` ".:
problème d'indentation, oubli de " `:` " ou erreur de parenthésage " `( ... )` ".

2) a) Erreurs d'indexations :

`x = [0,0,0] ; print(x[3])` produit le message d'erreur :
" `IndexError : list assignment index out of range` ".

b) Erreurs sur les opérations :

" `ZeroDivisionError : division by zero` ".

c) Erreurs de noms

" `NameError : name '...' is not defined` ".

C'est le cas notamment des fonctions non compilées ou des variables non affectées.

```
def essai() : return x
essai() # produit le message d'erreur
```
" `NameError: global name 'x' is not defined` ".

De même pour un appel à `ln(2)`, la fonciton logarithme étant notée `log`

Erreurs de nom de module : " `ImportError: No module named` '...' ".

3) a) Erreurs de types (on dit aussi de classe) :

`x = [0,1,0,1] ; x(3)` produit le message d'erreur :
" `TypeError : list object is not callable` " :
PYTHON n'attend pas une fonction (confusion par exemple entre $A[4]$ et $A(4)$).

`x = "2"+2` produit le message d'erreur
" `TypeError : can't convert 'int' object to str implicitly` ".

`1/input()` produit le message d'erreur
" `TypeError : unsupported operand type(s) for /: 'int' and 'str'` ".

b) Erreurs de type spécifiques à chaque module

`alg.det(2.5)` appelle la fonction déterminant avec un argument de type incorrect :
Cet appel produit un message d'erreur `LinAlgError` du module `numpy.linalg` :
" `numpy.linalg.linalg.LinAlgError: 0-dimensional array given. Array must be two-dimensional` ".

`integr.quad(2,0,1)` produit le message d'erreur :
" `quadpack.error: quad: first argument is not callable`