

Python_et_numpy_avance

October 15, 2023

1 Python

1.1 Les ensembles

- Comme en mathématiques : un ensemble ne contient pas deux fois le même élément
- Union $|$, intersection $\&$
- Avec les tableaux numpy, il existe la fonction `np.unique`

```
[45]: s1 = {1, 3, 4}
      s2 = {1, 6, 8}
      print(s1|s2)
      print(s1&s2)
```

```
{1, 3, 4, 6, 8}
{1}
```

```
[46]: set([1, 2, 3, 4, 3, 2, 1])
```

```
[46]: {1, 2, 3, 4}
```

```
[52]: data = "Comme en mathématiques : un ensemble ne contient pas deux fois le même_
      ↪élément"

      s = set(data)
      s
```

```
[52]: {' ',
      ':',
      'C',
      'a',
      'b',
      'c',
      'd',
      'e',
      'f',
      'h',
      'i',
      'l',
```

```
'm',
'n',
'o',
'p',
'q',
's',
't',
'u',
'x',
'é',
'ê'}
```

```
[57]: a = np.array([2, 1, 3, 3, 4, 5, 2, 3])
      np.unique(a)
```

```
[57]: array([1, 2, 3, 4, 5])
```

```
[ ]:
```

1.2 Les n-uplets (tuples)

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

```
[ ]: a = (1, 2, 3)
      print(a[1])
```

```
[62]: def ma_fonction():
      return 1, 2, 3

      a, b, c = ma_fonction()

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

```
<class 'tuple'>
2
```

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

```
[ ]: # Tuple de taille 0
      ()
      # Tuple de taille 1
```

```
(1,)
```

1.3 Les dictionnaires

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

```
[64]: parametres = {"N":100, 'l':30, "h":50, "g":9.81, 'm':0.1}
      print(parametres['N'])
```

```
100
```

```
[65]: # Boucle for sur un dictionnaire
      for key, val in parametres.items():
          print(f'La valeur de {key} est {val}')
```

```
La valeur de N est 100
La valeur de l est 30
La valeur de h est 50
La valeur de g est 9.81
La valeur de m est 0.1
```

```
[66]: # Liste de dictionnaire
      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.")
          #print('{} a {} ans'.format(personne['nom'], personne['age']))
```

```
Dupont a 13 ans.
Dubois a 34 ans.
```

```
[ ]:
```

1.4 Les fonctions

Nombre arbitraire d'arguments (tuple et dictionnaire)

```
[1]: def f(a, b, c):
      print(a, b, c)

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

1 3 2

```
[2]: g = f
     g.__name__
```

```
[2]: 'f'
```

```
[ ]: [f]
```

```
[70]: p = (2, 3)
     f(1, *p)
```

1 2 3

```
[79]: d = {'a':1, 'c':2}
     f(b=3, **d)
```

1 3 2

```
[87]: def f(a, *args, **kwd):
     print(a, args, kwd)

     f(1, 2, 3, 4)
     f(1, 2, 3, m=4, l=5)
     f(a=3, m=4, l=5)
```

```
1 (2, 3, 4) {}
1 (2, 3) {'m': 4, 'l': 5}
3 () {'m': 4, 'l': 5}
```

```
[88]: parametres = {'a':1, 'c':2}
     f(**parametres)
```

```
1 () {'c': 2}
```

```
[89]: import numpy as np

     parametres_pendule = {"m":100, "l":30, "g":9.81}
     def periode(l, g, **kwd):
         return 2*np.pi*np.sqrt(l/g)

     periode(**parametres_pendule)
```

```
[89]: 10.987679728847352
```

```
[ ]:
```

1.4.1 Fonctions anonymes

lambda arg1, arg2 : expr

```
[3]: f = lambda a, b, c: a+b+c  
f(1, 2, 3)
```

[3]: 6

```
[4]: (lambda a, b, c: a+b+c)(1, 2, 3)
```

[4]: 6

```
[ ]: # integrale(f, a, b, N_step)  
# integrale(lambda x: exp(-x**2), 0, 1, 100)
```

2 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'autres modules
- Modules standards (ex: math)

```
[5]: from math import exp  
exp(2.1)
```

[5]: 8.166169912567652

```
[6]: import math  
math.exp(1)
```

[6]: 2.718281828459045

```
[8]: # Syntaxe à éviter  
from math import *  
from numpy import *  
sin(pi/4)
```

[8]: 0.7071067811865476

```
[14]: np.acos(np.array([1, 2]))
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-14-e2b177d7b2db> in <module>  
----> 1 np.acos(np.array([1, 2]))  
  
~/.local/lib/python3.8/site-packages/numpy/__init__.py in __getattr__(attr)
```

```

309         return Tester
310
--> 311         raise AttributeError("module {!r} has no attribute "
312                                "{!r}".format(__name__, attr))
313

```

```

AttributeError: module 'numpy' has no attribute 'acos'

```

```

[7]: # Donner un nom plus simple
     # Eviter sauf si tout le monde le fait
     import numpy as np

```

3 Langage orienté objet

- Tout en Python est objet
- Un objet contient des données
- La classe d'un objet définit ce qu'est l'objet
- La classe définit 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

```

[19]: l = [5, 4, 1, 2]
      l.insert(1, 'coucou') # modification de l'objet
      print(l)
      # l.index('coucou')

```

```

[5, 'coucou', 4, 1, 2]

```

```

[20]: z = 1 + 2J
      z.real # attribut de l'objet
      z.conjugate() # methode qui renvoie un nouvel objet

```

```

[20]: (1-2j)

```

```

[21]: z

```

```

[21]: (1+2j)

```

```

[23]: print(l.insert(2, 'bonjour'))

```

None

```

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

```

```
print(b[0])
```

45

45

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

B

1

```
[28]: from math import pi
      def f(la_liste):
          la_liste[0] = 13 + pi

      l = [1, 2]
      f(l)
      l
```

[28]: [13, 2]

3.1 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'instruction `global` ne doit **jamais** être utilisée (sauf cas exceptionnels)

```
[29]: x = 1

      def f1():
          print(x)

      f1()
      x = 3
      f1()
      #def print(x):
      #     None
```

1

3

```
[32]: from math import sin, cos
      def f(x):
          print(sin(x))
```

```
f(1)
sin = cos
f(1)
```

0.8414709848078965
0.5403023058681398

```
[33]: def f2(x):
      print(x)
      print(x)
      f2(4)
```

3
4

```
[35]: x = 3
      def f3():
          x = 4
          print(x)
      f3()
      print(x)
```

4
3

```
[37]: x = 2
      def f4():
          print(x)
          x = 4
          print(x)
      f4()
```

```
-----
UnboundLocalError                                Traceback (most recent call last)
<ipython-input-37-1bfa6a49a868> in <module>
      4     x = 4
      5     print(x)
----> 6 f4()

<ipython-input-37-1bfa6a49a868> in f4()
      1 x = 2
      2 def f4():
----> 3     print(x)
      4     x = 4
      5     print(x)

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


Les variables globales doivent être constantes : * Constantes numériques * Autres objets : fonctions, modules, ...

```
[38]: from math import sin, pi

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

```
[ ]:
```

4 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"

```
[39]: from math import sqrt

sqrt(-1)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-39-eb80c803d6ae> in <module>
      1 from math import sqrt
      2
----> 3 sqrt(-1)

ValueError: math domain error
```

```
[40]: l = [1, 2, 4]
      l(1)
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-40-0642d5456545> in <module>
      1 l = [1, 2, 4]
----> 2 l(1)

TypeError: 'list' object is not callable
```

```
[41]: def f(x):
      print(x)
```

```
f[1]
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-41-eaf288da1121> in <module>  
      2     print(x)  
      3  
----> 4 f[1]  
  
TypeError: 'function' object is not subscriptable
```

```
[42]: def f(x):  
      y = 2*x  
      print(x)
```

```
File "<ipython-input-42-ae826c11a736>", line 3  
    print(x)  
    ^  
IndentationError: unexpected indent
```

```
[46]: def f(x, y, z):  
      return (x + y*(sin(x+z))*34  
  
      #           + 45 )  
  
      b = 2
```

```
File "<ipython-input-46-64b203b3365b>", line 6  
    b = 2  
    ^  
SyntaxError: invalid syntax
```

4.1 Créer sa propre erreur

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

```
[47]: # 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(h=1, v_0=10))
print(temps_arrivee(h=1, v_0=1))

```

0.10545470031833197

```

-----
Exception                                Traceback (most recent call last)
<ipython-input-47-d41ab9e67048> in <module>
      9
     10 print(temps_arrivee(h=1, v_0=10))
----> 11 print(temps_arrivee(h=1, v_0=1))

<ipython-input-47-d41ab9e67048> in temps_arrivee(h, v_0, g)
      5     Delta = v_0**2 - 2*g*h
      6     if Delta<0:
----> 7         raise Exception("La balle ne touche pas le plafond")
      8     return (v_0 - sqrt(Delta))/g
      9

Exception: La balle ne touche pas le plafond

```

```

[48]: # 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:
              print("La balle ne touche pas le plafond")
          #     raise Exception("La balle ne touche pas le plafond")
          else:
              return (v_0 - sqrt(Delta))/g

      t = temps_arrivee(h=1, v_0=10)
      print(2*t + 1)
      #print(temps_arrivee(h=1, v_0=1))
      t = temps_arrivee(h=1, v_0=1)
      print(2*t + 1)

```

1.2109094006366639

La balle ne touche pas le plafond

```

-----
TypeError                                Traceback (most recent call last)

```

```
<ipython-input-48-f2880f6bcfc3> in <module>
```

```
14 #print(temps_arrivee(h=1, v_0=1))
15 t = temps_arrivee(h=1, v_0=1)
---> 16 print(2*t + 1)
```

TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'

```
[49]: 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))
```

0.10545470031833197

ValueError Traceback (most recent call last)

```
<ipython-input-49-3a03f4968e75> in temps_arrivee(h, v_0, g)
```

```
4     try:
----> 5         return (v_0 - sqrt(Delta))/g
6     except ValueError:
```

ValueError: math domain error

During handling of the above exception, another exception occurred:

Exception Traceback (most recent call last)

```
<ipython-input-49-3a03f4968e75> in <module>
```

```
7     raise Exception("La balle ne touche pas le plafond")
8 print(temps_arrivee(1, 10))
----> 9 print(temps_arrivee(1, 1))
```

```
<ipython-input-49-3a03f4968e75> in temps_arrivee(h, v_0, g)
```

```
5     return (v_0 - sqrt(Delta))/g
6     except ValueError:
----> 7         raise Exception("La balle ne touche pas le plafond")
8 print(temps_arrivee(1, 10))
9 print(temps_arrivee(1, 1))
```

Exception: La balle ne touche pas le plafond

```
[ ]:
```

5 Les fichiers

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

```
[50]: !pwd
      # Sous windows !cd => c:\Users\login\
```

```
/home/pierre/Enseignement/2023/Python1/cours/cours2
```

```
[51]: filename = "/home/pierre/Enseignement/2023/Python1/cours/cours2/
      ↪Python_et_numpy_avance.ipynb"
      filename = "Python_et_numpy_avance.ipynb"
```

```
[52]: current_dir = "/home/pierre/Enseignement/2022/PythonENS/cours/cours2"
```

```
[53]: # Chemin relatif
      import os
      print(os.listdir(current_dir))
      #print(os.listdir('data/'))
      print(os.listdir('../cours1'))
```

```
['Python_et_numpy_avance.ipynb', 'untitled.txt', 'test.json', 'data',
'.ipynb_checkpoints', 'test_python.txt']
['Bases de python et utilisation de numpy.ipynb', 'ma_figure.pdf',
'ma_figure.svg']
```

```
[ ]:
```

```
[ ]:
```

```
[54]: file = 'data/test_python.txt'
      fd = open(file, 'w')
      fd.write('Bonjour\n')
      fd.close()
```

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

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

```
Bonjour
Au revoir
```

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

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

```
[ ]:
```

5.1 JSON

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

```
[61]: import json

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

annuaire = [personne_1, personne_2]

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

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

```
[{'nom': 'Dupont', 'age': 13}, {'nom': 'Dubois', 'age': 34}]
```

```
[ ]:
```

6 Numpy

```
[1]: import numpy as np
      import matplotlib.pyplot as plt
```

7 Tableaux nD

```
[6]: a = np.array([[1,2], [3, 4]])
      # l'index est un tuple
      a
      a[0, 1]
```

```
#a[(0, 1)]
```

```
[6]: 2
```

```
[9]: x = np.random.rand(5, 5)
      print(x)
```

```
[[0.44896538 0.66974977 0.53821711 0.57615746 0.08194611]
 [0.78903641 0.02560489 0.26720062 0.50950385 0.78355268]
 [0.42192165 0.17556142 0.40080314 0.2183728 0.03265151]
 [0.93833002 0.4835424 0.96206389 0.43382712 0.99247962]
 [0.08243995 0.99159993 0.54426793 0.37827489 0.61231135]]
```

```
[14]: np.zeros([2, 4])
```

```
[14]: array([[0., 0., 0., 0.],
            [0., 0., 0., 0.]])
```

```
[11]: # Récupérer une colonne
      x[:,0]
      x[1:3, :-1]
```

```
[11]: array([[0.78903641, 0.02560489, 0.26720062, 0.50950385],
            [0.42192165, 0.17556142, 0.40080314, 0.2183728 ]])
```

```
[12]: x.shape
```

```
[12]: (5, 5)
```

```
[ ]: # méthode reshape
      x = np.random.rand(25)
      x.reshape((5, 5))
```

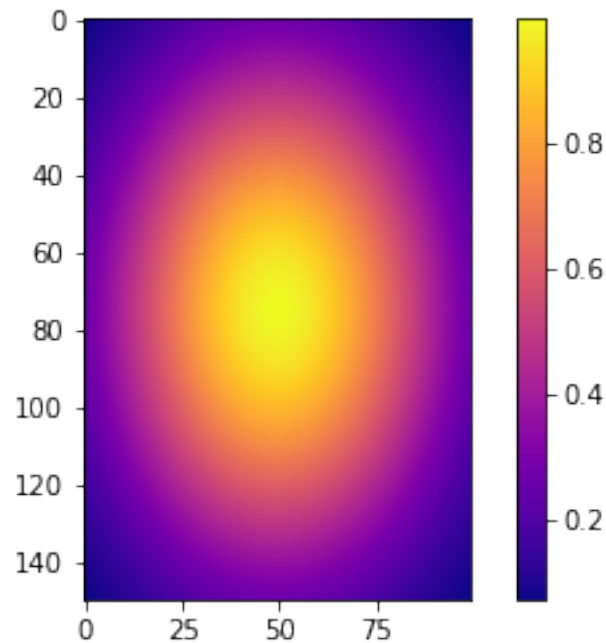
```
[16]: # meshgrid
      a, b = np.meshgrid([1, 2, 3], [7, 8, 9, 10])
      print(a)
      print(b)
```

```
[[1 2 3]
 [1 2 3]
 [1 2 3]
 [1 2 3]]
[[ 7  7  7]
 [ 8  8  8]
 [ 9  9  9]
 [10 10 10]]
```

```
[22]: # Création d'une image 2D avec meshgrid

x = np.linspace(-.5, .5, 100)*4*np.pi
y = np.linspace(-.5, .5, 150)*4*np.pi
X, Y = np.meshgrid(x, y)
R = np.sqrt(X**2 + Y**2)
#plt.imshow((np.sin(Y)+np.cos(X))*np.exp(-R**2/30))
plt.imshow(np.exp(-R**2/30), cmap='plasma')
plt.colorbar()
```

[22]: <matplotlib.colorbar.Colorbar at 0x7f89207834c0>



```
[ ]: a = np.random.rand(3, 4, 2)
a
```

```
[ ]: a.sum(axis=1)
```

```
[ ]: a.sum(axis=(1, 2))
```

```
[ ]:
```

```
[27]: x = np.random.rand(5, 3)
x
```



```
[27]: array([[0.14373463, 0.23439261, 0.25924488],
            [0.19477976, 0.2866457 , 0.9013089 ],
            [0.02264623, 0.90387362, 0.46635957],
            [0.54645181, 0.75662366, 0.22234325],
            [0.12929755, 0.88166716, 0.62862056]])
```

```
[29]: x.mean(axis=1)
```

```
[29]: array([0.21245737, 0.46091145, 0.46429314, 0.50847291, 0.54652842])
```

8 Tableau dans la mémoire

- strides
- from numpy.lib.stride_tricks import as_strided

```
[1]: a = []
     b = a
     b.append(1)
     a
```

```
[1]: [1]
```

```
[ ]: a = np.arange(10)
     b = a[:3]
     b[0] = 10
     a
```

```
[ ]: a.shape
```

```
[ ]: a = np.linspace(0, 1, 11)
     #a.dtype = np.int64
     a
```

```
[ ]: a[2]
```

```
[ ]: b = a[::2]
     b
```

```
[ ]:
```

```
[ ]: print(a.strides)
     print(b.strides)
```

```
[ ]: a = np.zeros((4, 3))
     a.strides
```

```
[ ]: a = np.arange(10)
a.reshape((5, 2))
```

```
[ ]: from numpy.lib.stride_tricks import as_strided
a = np.arange(10)
b = as_strided(a, shape=((6, 5)), strides=(8, 8))
print(b)
#b.mean(axis=1)
```

9 Broadcast

```
[30]: import numpy as np
from numpy import *
```

```
[32]: x = np.random.rand(5)
x
```

```
[32]: array([0.40514252, 0.13478749, 0.22673899, 0.98649996, 0.35557377])
```

```
[38]: x[2:4] + 2
```

```
[38]: array([3., 4.])
```

```
[ ]:
```

```
[42]: # Si sur une dimension la taille du tableau vaut 1, alors numpy peut l'étendre
# pour qu'elle ait la même valeur que celle de l'autre tableau
x = random.rand(5, 5)
a = array([ [1, 2, 3, 4, 5] ])
print(x.shape)
print(a.shape)
x + a
x[2:4, :] + a
```

```
(5, 5)
```

```
(1, 5)
```

```
[42]: array([[1.02509885, 2.74115212, 3.29267669, 4.72117002, 5.59556385],
          [1.52200889, 2.33703806, 3.70425874, 4.56971256, 5.15240082]])
```

```
[ ]: x = np.arange(10)
x + 1
```

```
[ ]: # Il y a une syntaxe simple pour rajouter une dimension de taille 1
# C'est newaxis
a = arange(5)
```

```
b = a[np.newaxis, :]
x = random.rand(5, 5)
x[2:4, :] = b
x
```

```
[43]: # Exemple : calculer une moyenne pondérée
# Chaque ligne est un élève, chaque colonne un examen
notes = random.rand(10, 5)*20
print(notes)
coef = array([1, 4, 2, 5, 8])
```

```
[[ 8.36849253 11.92181278  9.00265016 18.86317485 18.7849147 ]
 [14.13264325 16.54411792 11.07599628  3.55013717 15.55692167]
 [12.22955338  0.72140408  1.01774637  9.32702606  7.06777787]
 [ 8.1618087  17.02043415  5.36869047 15.26328654 17.53625932]
 [ 0.27927201  7.36561785  1.15922399 18.97472348 12.69871536]
 [12.56920069 16.97387632 12.16501643 16.49076701  5.35795125]
 [ 1.91633744 19.12602645 10.92793234 18.09573856 19.31894248]
 [ 9.47044479  2.09979974  0.56171049  8.13554496 12.48921352]
 [17.18138621  6.12986777  6.298114    2.06346925 15.41860937]
 [ 0.57529256 12.10533197 14.31454181  6.87408222  1.72796792]]
```

```
[44]: # Il est inutile de faire des boucles
(notes*coef[np.newaxis,:]).sum(axis=1)/np.sum(coef)
```

```
[44]: array([15.93281179, 12.23335834,  6.01640079, 15.17937168, 11.42617658,
          11.50560919, 17.26532703,  7.97922488,  9.39816532,  6.29099293])
```

10 Au delà de numpy : numba

- Calculer π (avec une formule très très lente!!!)

$$\frac{\pi}{4} = \sum_i \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

- numba.vectorize

```
[51]: def pi_python(N):
    res = 0
    coef = 1
    for i in range(N):
        res += coef/(2*i+1)
        coef = -coef
    return 4*res

%timeit pi_python(1000000) # 77 ms
```

121 ms \pm 1.32 ms per loop (mean \pm std. dev. of 7 runs, 10 loops each)

```
[52]: import numpy as np
def pi_np(N):
    Ti = np.arange(N)
    moins_un_puissance_i = ( 1-2*(Ti%2) )
    return 4*np.sum( moins_un_puissance_i/(2*Ti+1))

%timeit pi_np(1000000) # 11.5ms
```

16.2 ms \pm 506 μ s per loop (mean \pm std. dev. of 7 runs, 100 loops each)

```
[54]: from numba import jit, int64, float64
import numba
# Just in time compiler

numba_pi = jit(float64(int64))(pi_python)

%timeit numba_pi(1000000)
```

1.67 ms \pm 1.59 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

```
[55]: numba_pi(1000000)
```

```
[55]: 3.1415916535897743
```

```
[69]: # Décorateur @
@jit( float64(int64) )
def numba_pi(N):
    res = 0
    coef = 1
    for i in range(N):
        res += coef/(2*i+1)
        coef = -coef
    return 4*res
```

```
[70]: #@jit(float64[:])(float64[:], float64[:]), parallel=True)
@jit(parallel=True)
def somme(a, b):
    N = len(a)
    c = np.zeros(N)
    for i in numba.prange(N):
        c[i] = a[i] + b[i]
    return c

a = np.linspace(1, 2, 1000000)
b = np.logspace(1, 2, 1000000)
```

```
somme(a, b)
```

```
%timeit somme(a, b)
```

1.3 ms \pm 61.9 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

```
[71]: %timeit a + b
```

1.37 ms \pm 110 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

```
[ ]: # Numpy utilise des variables intermédiaires -> vitesse limitée par la mémoire
(a + b)*c + a
tmp1 = a + b
tmp2 = tmp1*c
res = tmp2 + a
```

```
[ ]:
```

```
[42]: def volume(dimension, M):
      pts = np.random.rand(M, dimension)*2 - 1
      carre_norme = np.sum(pts**2, axis=1)
      return np.mean(carre_norme<1) * 2**dimension

volume(5, 1000000)
```

```
[42]: 5.277408
```

```
[49]: %timeit volume(10, 10000000)
```

1.89 s \pm 96.3 ms per loop (mean \pm std. dev. of 7 runs, 1 loop each)

```
[47]: @jit(float64(float64, int64))
def volume_numba(dimension, M):
    output = 0
    for i in range(M):
        point = np.random.rand(dimension)*2 -1
        carre_norme = np.sum(point**2)
        if carre_norme<1:
            output += 1
    return output/M * 2**dimension

volume_numba(5, 1000000)
```

```
[47]: 5.273312
```

```
[50]: %timeit volume_numba(10, 10000000)
```

3.29 s ± 7.58 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

11 Les décorateurs

Utilisé pour modifier automatiquement une fonction

Par exemple :

- Gérer les erreurs d'une façon spécifique
- Loguer les appels d'une fonction
- Mettre en cache
- Pour numpy : np.vectorize peut être utilisé comme un décorateur

```
[60]: def dit_bonjour(f):
      def nouvelle_fonction(*args, **kwd):
          print('Bonjour', args)
          return f(*args, **kwd)
      return nouvelle_fonction

      @dit_bonjour
      def f(x):
          return x**2

      f(2)
```

Bonjour (2,)

```
[60]: 4
```

```
[ ]: f(2)
```

```
[62]: def dit_qqc(mot):
      def mon_decorateur(f):
          def nouvelle_fonction(*args, **kwd):
              print(mot)
              return f(*args, **kwd)
          return nouvelle_fonction
      return mon_decorateur

      @dit_bonjour
      @dit_qqc('Hello!')
      def f(x):
          return 2*x

      f(2)
```

```
Bonjour (2,)
Hello!
```

```
[62]: 4
```

```
[68]: import time

def cache(f):
    the_cache = {}
    def nouvelle_fonction(x):
        if x not in the_cache.keys():
            the_cache[x] = f(x)
        return the_cache[x]
    return nouvelle_fonction

@cache
def f(x):
    time.sleep(2)
    print('COUCOU')
    return x**2

@cache
def g(x):
    time.sleep(2)
    print('BONJOUR')
    return x**3

print(f(2))
print(g(2))
```

```
COUCOU
4
BONJOUR
8
```

```
[64]: f(2)
```

```
COUCOU
```

```
[64]: 4
```

```
[65]: f(2)
```

```
[65]: 4
```

```
[ ]:
```