Python survie

Version du 07/03/2023

1 Modules

1.1 Pour importer

Pour pouvoir utiliser par exemple $\sqrt{\pi}$:

```
import numpy
numpy.sqrt(numpy.pi)

import numpy as np
np.sqrt(np.pi)

from numpy import sqrt, pi
sqrt(pi)

from numpy import *   #déconseillé
sqrt(pi)
```

1.2 Modules importants

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

2 Opérations arithmetiques

```
+, -, *, /
x ** y, pow(x,y)    # puissance
x // y   # quotient de division
x % y    # reste de division
```

```
round(1.4) # -> 1 (arrondi)
```

3 Types

```
type(5) # int
type(5.3) # float
type(5.) # comme 5.0 : float
type("bonjour") # str
```

4 Variables

4.1 Affectation

```
a = 2.5
a, b = 2.4, 3 # affectation parallèle
a=b=c=0 # affectation multiple
x, y = y, x # échange de deux nombres
```

4.2 Incrémentation

```
x = x + 1
x += 1 # idem
x -= 1# marche aussi pour *, /
```

5 Print

5.1 Méthode basique

```
print(x)
print("x=", x)
print("x=", x, "et y=", y)
print("10/3~', round(10/3, 2)) # 10/3 ~ 3.33
```

5.2 f-string

```
print(f"x={x}")
print(f'x={x} et y={y}")
print(f"Le quotient de {x} par {y} est {x/y}")

# {var : .2f} affiche var avec deux chiffres après le
    point (virgule)
print(f"Le quotient de {x} par {y} est {x/y:.2f}")
print(f"0.3 corresponnd à {0.3:.2%}") # 0.3 correspond
    à 30.00%
```

5.3 end

```
print("...", end="; ") # remplace retour ligne par "; "
```

5.4 print sur plusieurs lignes

```
print("longue phrase que j'écris sur deux lignes \
     mais qui s'affiche sur une")
```

5.5 Retour à la ligne

```
print("\n") # retour à la ligne
print("ligne 1\nligne2")
# \n est collé à ligne2 pour éviter un espace
```

6 input

```
chaine = input("chaine qui s'affiche à l'écran") #
    donne un string
val = int(chaine) # transforme en entier
val = float(chaine) # transforme en float
```

7 Tests conditionnels

7.1 Opérateur booléens

```
True, False
```

7.2 Opérateur de comparaison

```
a == b # égal
a != b # différent
a <= b;  a < b; a>=b; a>b
```

7.3 Opérateur logique

```
or, and, not
```

7.4 Test condtionnel

```
if condition1:
    bloc d'instruction 1
elif condition2: # peut être omis ou répété
    bloc d'instruction 2
else: # peut être omis
   instruction 3
```

8 Listes

8.1 Création

```
L = [ 1, [2,4], "blue" ] # liste "mixte"
L = [] # liste vide
```

```
L = 4 * [0] # [0, 0, 0, 0]
L = [2*s for s in liste] # liste par compréhension
```

8.2 Liste d'entiers

```
list(range(1, 6)) # [1, 2, 3, 4, 5]
list(range(début, fin, pas)) # fin n'est pas inclus
```

8.3 Accès par élément

```
L[0] # premier élément
L[-1] # dernier élément
```

8.4 Accès par tranche

```
L[start, stop, pas] # stop est exclu
L[2:6] # L[2],...,L[5]
L[:5] # L[0],...,L[4]
L[0:5] # idem
L[2:] # L[2], L[3], ..
L[-3:] # trois dernier éléments: L[-3], L[-2], L[-1]

L[::2] # L[0], L[2], ....
```

8.5 Concaténation

```
liste1 + liste2 # concaténation
2*liste # liste + liste

liste.append(x)
liste = liste + [x] # idem
liste += [x] # idem
```

8.6 Longueur

```
len(liste)
```

8.7 Liste en compréhension

```
[x**2 for x in liste]
```

9 String

9.1 Création

```
"Rennes"
'Rennes'
```

```
"L'eau vive"
'L\'eau vive' # idem
'Il a dit "hello" et est parti'
"Il a dit \"hello\" et est parti" # idem
```

9.2 Accès par élément ou tranche

Comme pour les listes :

```
chaine[0]
chaine[-1]
chaine[debut, fin, pas]
chaine[:3] # premier 3 éléments
chaine[-3:] # dernier 3 éléments
```

9.3 Concaténation

Comme pour les listes, mais append ne marche pas :

```
chaine1 + chaine2 # concaténation
2*chaine1 # chaine1 + chaine1
```

9.4 Longueur

```
len(chaine)
```

10 Boucle for

```
for variable in objet itérable:
    instructions
# Exemples
for k in range(0, n): # k=0,...,n-1
for k in range(n) # idem
for k in range(m, n): # k=m, m+1, ...,n-1
for k in [2, 15, 4]: # 2, 15, 4
for k in "bonjour": # "b", "o", "n", ..., "r"
```

10.1 Parcourir une liste

```
for i in range(0, len(liste)):
    print(liste[i]) # élément indice i de la liste

for element in liste:
    print(element)
```

10.2 enumerate : couple indice, élément

```
for (indice, element) in enumerate(liste):
    print(indice, element)
```

Même chose sans `enumerate` :

```
for indice in range(0, len(liste))
    print(indice, liste[indice])
```

10.3 zip : parcourir deux éléments

```
for (element1, element2) in zip(liste1, liste2):
    print(elemen1, element2)
```

Même chose sans `zip` :

```
for indice in range(0, len(liste)):
    print(liste1[indice], liste2[indice])
```

11 Boucle while

```
while condition:
    instructions
```

12 numpy

```
import numpy as np
np.sin(np.pi)
```

```
from numpy import sin, pi
sin(pi)
```

12.1 Fonctions numériques prédéfinies

```
np.sin(), np.cos(), np.tan()
np.sqrt()
np.exp()
np.log(), np.log10(), np.log2()
np.abs()
np.degrees(), np.radians()
```

12.2 Linspace, arange

```
# array de 30 points équidistants entre 1 et 3 (inclus)
np.linspace(1, 3, 30)
np.linspace(1, 3) # comme np.linspace(1, 3, 50)
```

```
# array de points entre 1 et 3 (exclus) espacés de 0.1
np.arange(1, 3, 0.1)
```

12.3 Opérations sur les array

Les opérations sur les array se font élément par élément.

```
array1 + array2 # somme élément par élément
array1 * array2 # produit élément par élément
2*array # produit élément par élément
np.sin(array) # np.sin de chaque élément
```

13 Fonction Python

```
def ma_fonction(x, y, ....):
    ....
    return ....
```

14 Random

Choix d'un élément d'une liste suivant une probabilité uniforme.

```
import random

random.choice([a,b,c]) # choix d'un élément de [a,b,c]
random.choixe(liste) # choix d'un élément de liste
```

Choix d'un entier entre m et n (inclus) suivant une probabilité uniforme.

```
import random

random.randint(m, n) # choix d'un entier entre m et n
random.randint(-2, 5)#choix d'un entier de -2,-1,...,5
```

15 Matplotlib

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

plt.figure(
    figsize=(8, 4)) # option: largeur 8, hauteur 4
l_x = np.linspace(-5, 2, 100)
l_y1 = [1 + np.exp(x) for x in l_x]
l_y2 = [1 - np.exp(x) for x in l_x]
# options
# l_y1 = 1 + np.exp(l_x)
# l_y2 = 1 - np.exp(l_y)
plt.title("Graphe de .....", fontsize=12, color="blue")
plt.plot(l_x, l_y1, # commande minimale
    color="green", # couleur en option
    label="1+exp(x)", # label en option
    linestyle = ":", # courbe pointillée en option;
        aussi "--"
    linewidth = 3, # largeur de la courbe en option
    )
plt.plot(l_x, l_y2, color="...", label="...", ...)
plt.xlabel("x",
    fontsize=12, color="blue") # en option
plt.ylabel("f(x), g(x)")
plt.legend(fontsize=9) # pour afficher "label"
plt.grid() # en option: quadrillage
plt.show()
```

15.1 Autre commandes

```
plt.xticks([1,3,...]) # n'affiche que 1,3,.. sur l'axe
     des x
plt.xticks([1,3,...], ['a','b',...]) # affiche a,b.. en
     x=1,3..
plt.yticks(...)
plt.xlim(a, b) # restriction axe des x sur [a,b]
plt.ylim(c,d)
plt.axis('equal')  # repère orthonormé
```

15.2 Échelle logarithmique

On remplace `plt.plot()` par `plt.semilogy()` avec la même syntaxe, les mêmes options :

```
plt.semilogy(l_x, l_y, ....) # même options que plot
```

15.3 Représentation d'une suite de points

```
plt.plot(l_x, l_y,
    "o", # suite de points
    markersize = 4, # taille des points
    color="red", # couleur
    )
```

Variante : Suite de points reliées par des ségments

```
plt.plot(l_x, l_y,
    "-o", # suite de points reliés par des ségments
    markersize = 4, # taille des points
    color="red", # couleur
    linewidth = 3, # largeur de la courbe en option
    )
```

15.4 Droite horizontale/verticale

Suite de points

```
plt.axhline(y=4,
    color="red", # couleur
    linestyle = ":", # pointillée; aussi "--"
    linewidth = 3, # largeur de la droite
    )
plt.axvline(x=5,
     .... # même options que axhline
     )
```

16 Jupyter

— `print` n'est pas nécessaire pour la dernière ligne d'une cellule.

— les variables définies dans une cellule exécutée sont connues dans les autre.

16.1 Raccourcis

Maj-Ent	execute cellule + selectionne suivante
Opt(Alt)-Ent	executer cellule + insérer nouvelle
Cmd-Ent	execute et reste sur la cellule
Cmd z	undelete dernières modif dans cellule
Maj-l	affiche ou non les numéros de lignes
Esc m	cellule markdown
Esc y	cellule code
Esc a, Esc b	rajouter cellule avant, après
selection text + tab	indentation
selection text + Maj tab	desindentation
selection text + Cmd /	commenter, dé-commenter

16.2 Basthon

Travailler sur un notebook en ligne avec Basthon ne modifie pas l'original. L'enregistrer crée une nouvelle copie.

Différences

1. **matplotlib** Toujours utiliser `plt.figure()` ou `plt.figure(figsize=(6,4)`.
2. **ouvrir Fichier** Avant d'exécuter commande Python pour lire fichier, cliquer sur icone "dossier" et chercher fichier sur ordi.