Python

Ce cours ne prétend pas être exhaustif, loin de là.
Son but est de faciliter l'accès aux débutants des fonctionnalités les plus utiles de Python et de répondre aux exigences de l'enseignement jusqu'en classe de terminale.
En ce qui concerne les applications pratiques qu'on peut faire de Python dans l'enseignement des sciences, on pourra consulter ce document officiel:
http://cache.media.eduscol.education.fr/file/Mathematiques/73/3/Algorithmique_et_programmation_787733.pdf
.. qui mériterait d'être mis à jour sans quoi certaines des méthodes qu'il préconise ne fonctionneront pas sur une version actuelle et officielle de Python.
Mais il donne malgré tout de précieuses indications sur les procédures et de nombreuses pistes à explorer.

Table des matières

Généralités sur Python

Sous Python: Un programme est une liste d'instructions réalisées de façon séquentielle dans leur ordre d'écriture

```
Instruction 1
Instruction 2
…
Instruction n
```

Les données et leurs conteneurs

Les instructions manipulent des **données** qui peuvent être des nombres ou des chaînes (lettres, mots, phrases).
Les données peuvent être stockées dans des **variables** grâce à des **instructions** d'affectation du type
Nom de la variable = donnée (On tape **x=10** on termine par la touche [entrée]. On a tapé une instruction.)
Le nom de la variable s'écrit en lettres éventuellement suivies de chiffres. Noms possibles: a , x, a1 , delta, age …
C'est la façon d'écrire la donnée qui permet de préciser si le contenu de la variable est de type numérique ou chaine.

Variable numérique	a = 101	a=10+4	a= –10	a=25.32	chiffres avec point décimal
Variable chaîne	a =" bonjour"	a="Bonjour "+ "Madame"	a='bonjour'	a="101"	expressions entre guillemets

A côté de ça, il existe des conteneurs spéciaux qui stockent des ensembles de valeurs

Type List **a=[1,3,5,8,102]** **a=[]** liste vide. **a=list("abc")** renvoie **a=['a','b','c']**
Type Range **range(1,5)** c'est un intervalle de valeurs entières. Ici contient 1,2,3,4. **range(,5)** contient 0,1,2,3,4.
range(1,12,3) = intervalle [1,11] par saut de 3 ={1,4,7,11}
Type Tuple (N-uplet en français) permet de stocker des données de plusieurs types, séparées par des virgules
tuple vide **()** tuple à un élément **a,** ou **(a,)** à 3 éléments **1,2.03,'c'** ou **(a,b,c)** ou **tuple('abc')=(a,b,c)**

Différents types d'instructions

Les instructions d'entrée/sortie

sont celles qui permettent à la machine de communiquer avec le monde extérieur dont l'opérateur fait partie.

L'instruction print(donnée**)** permet d'afficher à l'écran des données brutes ou stockées dans des variables
C'est le programmeur qui choisit la donnée à afficher.

print(101)	print("bonjour monsieur")	print(a) imprime le contenu de la variable a	print(a+b) si a et b variables de même type

L'instruction x=**input(**prompt**)** permet à l'opérateur de passer des données à une variable du programme.
Le prompt est la phrase qu'on lit à l'écran pour demander d'entrer une donnée. Il est choisi par le programmeur.
Le programmeur choisit aussi la variable de stockage qui ici est x.

x = input("donnez un chiffre:") → affiche donnez un chiffre: | attente | on tape 125 suivi de la touche [entrée]| suite
x= input("rentrez votre nom:") → affiche rentrez votre nom: | attente |on tape Duchnock suivi de [entrée] | suite

Dans les deux cas la variable x contient la donnée entrée et elle prend le type chaine. x="125" ou x="Duchnock"
Pour transformer x="125" en variable numérique il faut une instruction de conversion de type: x=**int(**x**)**
qui transforme x en nombre entier. Après quoi x = 125.

Print () et input() ne sont pas les seules instructions d'entrée/sortie. Il y en a d'autres qui permettent de dessiner à l'écran, de jouer de la musique ou une vidéo, d'émettre de signaux électriques à travers les ports (USB, VGA, COM , Ethernet, HDMI, ….) ce qui permet par exemple de communiquer par internet, d'imprimer, de lire ou d'écrire une clé USB, de commander une machine, de lire des capteurs de température , de pression ou autres…

Les instructions opératoires

sont celles qui permettent de manipuler les données (nombres ou chaînes) grâce à des **opérations** ou à des **fonctions**, pour les modifier ou les analyser au gré du programmeur.
Beaucoup de fonctions usuelles sont fournies avec Python natif (ou en important des modules spécialisés) mais le programmeur peut créer ses propres fonctions et les appeler en leur passant les variables de son choix.
Ces instructions donnent également accès à un **générateur aléatoire** qui permet d'obtenir des nombres au hasard.
Certains jeux d'instructions appelés "**structures**" permettent

- De tester une condition et de réaliser une tâche ou non (ou une autre) selon que la condition est vraie ou fausse.
- De réaliser une tâche pour toutes les valeurs d'une variable contenues dans un intervalle (un range) ou une liste.
- De réaliser une tâche aussi longtemps qu'une condition est vraie.

Le test des conditions a pour résultat **true** (vrai) ou **false**(faux) ce qui est considéré comme une donnée d'un type spécial différent d'un nombre ou d'une chaîne: le type "**Boléen**"
Par exemple print (5<8) → affiche true et print(8<5) → affiche false ce qui est la valeur de la condition testée.

Les types de variables

Les variables servent à mémoriser des données de tous types, à les soumettre à des opérations ou à des fonctions.
Leur nom est constitué de lettres, éventuellement suivies de chiffres ou du symbole souligné (Exemple nombre_8)
Les variables sont sensibles à la casse: M n'est pas la même variable que m.
Les principaux types de variables sont créés et différenciés lors de leur affectation:

Type	Contenu	Affectation
Int	Nombre entier	a = 8
Float	Nombre décimal	a = 3.56
Complex	Nombre complexe	a = 3 +2j (en France 3 + 2i, pas de signe de la multiplication entre le 2 et le j)
str	Chaine, lettres, caractères	a = "bonjour les amis" ou a = 'bonjour les amis'
Bool	Booléen, valeur vrai ou faux	a = true , a =false (permet de tester la valeur d'une proposition)

Opérations sur les nombres

Principales opérations sur les nombres	
x+y	somme de x et y
x – y	Différence x moins y
x * y	x multiplié par y
x / y	x divisé par y
x // y	division entière de x par y (donne la partie entière du quotient)
x % y	reste de la division entière de x par y (10 % 3 = 1 si x divisible par y 10 % 5 = 0)
– x	Opposé de x
+ x	x
abs(x)	valeur absolue de x
int(x)	conversion de x en entier ou partie entière de x si x est un décimal
Round(x,2)	Arrondi avec 2 décimales Round(x) arrondi à l'entier le plus proche
float(x)	Transforme x en décimal
complex(2,3)	crée un complexe 2+3j
c.conjugate()	conjugué du nombre complexe c
pow(x,y)	x à la puissance y
xy**	x à la puissance y

Opérations sur les chaines

Principales opérations sur les chaines	Caractères spéciaux
s1 + s2 Concaténation de ’s1’ et ’s2’ → 'toi'+ ' et ' +'moi' ='toi et moi' **len(s)** Longueur de la chaîne s **s.find('ab')** renvoie le plus petit index de la sous chaîne 'ab' dans s **s.replace('old','new')** remplace la 1ere occurrence de 'old' dans s par 'new' **s[n:m]** Extraction des caractères se trouvant entre le n-ième et le m-iè **s * n** Répétition ’n’ fois de la chaîne ’s’ **s in t** Teste si la chaîne ’s’ est présente dans ’t’. **s[n]** Extraction du n-ième caractère en partant du début de la chaîne **s[-n]** Extraction du n-ième caractère en partant de la fin de la chaîne **s[:n]** Extraction des ’n’ premiers caractères de la chaîne **s[n:]** Extraction des derniers caractères de la chaîne à partir du n-ème **s[-n:]** Extraction des caractères des ’n’ derniers caractères de la chaîne **chr(65)** : renvoie 'A' (conversion d'un code ASCII en caractère)	**** Antislash **\'** Guillemet simple **\"** Guillemet double **\n** Saut de ligne **\r** Retour chariot **\t** Tabulation horizontale Intégrés à une chaine ces caractères ont une action sur le print. Print ("bonjour") donne bonjour Print("bon\tjour") donne bon jour Print ("\"bonjour\"") donne "bonjour"

Listes, intervalles, n-uplets

Listes =[ ] ou list()

Définition et affectation	entiers=[1,2,3,4,5]
Nombre d'éléments d'une liste	len(entiers) = 5
Accès à un élément de la liste	entiers[0] = 1 entiers[1] = 2 etc .
à l'envers:	entiers[–1] =5 entiers[–2] =4 etc.
Modification	entiers[3]=29
Suppression d'un élément	del entiers[3]
Insertion d'un élément:	entiers[2:2] = [0] insère le 0 en position 2 , le 3 passe en pos 3.
tri de liste	entiers.sort() range les éléments dans l'ordre alphanumérique
Sous listes	
Plage de 1 à 3	entiers[1:4]
Indices > 3	entiers[3:]
Indices ≤ 2	entiers[:3]
Appartenance	if (4 in entiers) teste si 4 est dans la liste
Non appartenance	if (not 4 in entiers)
Fusion de listes	total = entiers + fractions crée la liste fusionnant les 2
Répétition	total =entiers*3 équivalent à entiers + entiers+ entiers
Copie	nombres=entiers (2 noms pour la même liste)
Comparaisons	if entiers == nombres ou if entiers ! = nombres

Intervalles = range()

Création	i=range(1,5)	→ i=[1,2,3,4]
	I =range(1,9,2)	→ [1;8] par pas de 2 = [1 ,3 , 5, 7]
	I = range (5)	→intervalle d'entiers de 0 à 4 =[0,1,2,3,4]
Print(i)	range(1,5)	
print(len(i))	5	
print(i[2])	3	
print(i[2:5])	range(3,5)	

N-Uplets = (, ,) = , , = tuple(, ,)

Permet de grouper des éléments de natures différentes par exemple t = (5 , 2.03 , 'abc')
t = 3, 'a', 4

Opérations communes aux Listes, intervalles, n-upplets

Opération	Résultat
`x in s`	`True` si un élément de *s* est égal à *x*, sinon `False`
`x not in s`	`False` si un élément de *s* est égal à *x*, sinon `True`
`s + t`	la concaténation de *s* et *t*
`s * n` or `n * s`	équivalent à ajouter *s* *n* fois à lui même
`s[i]`	*i*ème élément de *s* en commençant par 0
`s[i:j]`	tranche (*slice*) de *s* de *i* à *j*
`s[i:j:k]`	tranche (*slice*) de *s* de *i* à *j* avec un pas de *k*
`len(s)`	longueur de *s*
`min(s)`	plus petit élément de *s*
`max(s)`	plus grand élément de *s*
`s.index(x[, i[, j]])`	indice de la première occurrence de *x* dans *s* (à ou après l'indice *i* et avant indice *j*)
`s.count(x)`	nombre total d'occurrences de *x* dans *s*

Opérations sur les listes

Opération	Résultat
`s[i] = x`	element *i* de *s* est remplacé par *x*
`s[i:j] = t`	tranche de *s* de *i* à *j* est remplacée par le contenu de l'itérable *t*
`del s[i:j]`	identique à `s[i:j] = []`
`s[i:j:k] = t`	les éléments de `s[i:j:k]` sont remplacés par ceux de *t*
`del s[i:j:k]`	supprime les éléments de `s[i:j:k]` de la liste
`s.append(x)`	ajoute *x* à la fin de la séquence (identique à `s[len(s):len(s)] = [x]`)
`s.clear()`	supprime tous les éléments de `s` (identique à `del s[:]`)
`s.copy()`	crée une copie superficielle de `s` (identique à `s[:]`)
`s.extend(t)` or `s += t`	étend *s* avec le contenu de *t* (proche de `s[len(s):len(s)] = t`)
`s *= n`	met à jour *s* avec son contenu répété *n* fois
`s.insert(i, x)`	insère *x* dans *s* à l'index donné par *i* (identique à `s[i:i] = [x]`)
`s.pop([i])`	récupère l'élément à *i* et le supprime de *s*
`s.remove(x)`	supprime le premier élément de *s* pour qui `s[i] == x`
`s.reverse()`	inverse sur place les éléments de *s*

Conditions à valeur booléenne

Condition à tester	Opérateurs logiques de conditions	
x < y Inférieur strict	**not** (int(x)==x)	Négation
x <= y Inférieur ou égal	x==3 **or** x==4	Ou-logique
x > y Supérieur strict	x%2==0 **and** x> 7	Et-logique
x >= y Supérieur ou égal	**(**x >5 or x<6**)** and y>7	() Groupe plusieurs conditions en une seule
x == y Égal		
x != y Différent		

Notez: le "égale" permettant d'affecter une valeur à une variable s'écrit **=** (a = 3)
le égale figurant dans une condition s'écrit **==** (si x == y alors ….)

Instruction if

Permet de réaliser un ensemble d'instructions si une condition est réalisée

```
If x < 20 :
        Instruction 1
        Instruction 2
Elif  x < 25 :
        Instruction 3
Else :
        Instruction 4
Suite du programme
```

Si x < 20
        On réalise instructions 1 et 2

Sinon, si x est entre 20 et 25
        On réalise l'instruction 3
Sinon   (sous-entendu si x ≥ 25)
        On réalise instruction 4
La suite du programme n'est plus concernée par la condition précédente

Attention if, elif, else ne sont précédés d'aucun caractère, leur ligne doit se terminer par 2 points.
Les instructions à réaliser si la condition est vérifiée sont précédées de plusieurs espaces (indentation).

Instruction while

Permet de répéter un ensemble d'instructions tant qu'une condition est réalisée

```
n=1
While n <= 10000:
      Instructions
      n+=1
suite
```

Initialisation de n à 1
Tant que n ≤ 10000
      Réaliser les instructions (en retrait)
      Augmenter n de 1 (en retrait) Ne pas oublier sinon boucle sans fin.
On sort de la boucle quand n = 10001

```
n = 1
While n <= 10000:
      If n%17==0 and n%23==0:
            break
      n+=1
print(n, "est divisible par 17 et 23"
```

Ici on teste une condition (n est-il un multiple de 17 et de 23)
        Et si la condition est réalisée le break interrompt la boucle
        (on passe au print)

```
n = 0
While n <= 50:
      n+=1
      if n%2 != 0:
            continue
      print(n)
suite
```

Ici on initialise n à 0 car on l'augmente de 1 avant de tester

On teste la condition (si n n'est pas divisible par 2, si n est impair)
    Si elle est réalisée le continue nous rebranche au n+=1 sans rien faire
    Sinon on affiche n et on retourne au n+=1.
Résultat on imprime les nombre pairs compris entre 1 et 50

Instruction for

entiers = [1, 2, 3, 4, 5] **for n in entiers:** print(n)	Pour n dans entiers, parcourt la liste et affiche à l'écran 1 2 3 4 5
for i in range(len(entiers)): print(entiers[i])	Cette fois i varie de 0 à len(entiers) –1 c'est-à-dire de 0 à 4
for i in range(4): print (4*i +7)	i varie de 0 à 3

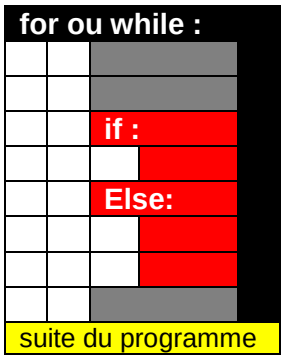
for i in range(10):
if (i+1)%7 =0:
break
else:
print(i)
	i varie de 0 à 9.
	Mais quand i +1 sera divisible par 7
	autrement dit quand i sera égal à 6
	la boucle for cessera à cause du break
	Tant que i +1 ne sera pas divisible par 7, elle imprimera i
	résultat : 0 , 1 , 2, 3, 4, 5 puisque quand i = 6 on sort de la boucle

for num in range(2,6): if num %2 ==0: print(num, "pair") **continue** print(num)	Sans le "continue" le programme afficherait 2 pair, 2 , 3, 4 pair , 4, 5. Mais l'instruction continue provoque une itération de num négligeant la suite du paquet d'instructions du for. Donc dans ce cas l'instruction print (num) n'est pas réalisée. L'affichage observé est donc: 2 pair, 3, 4 pair, 5.

Conventions d'écriture:

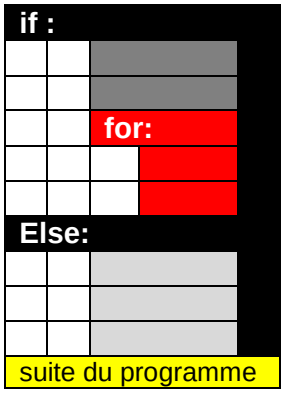
Les deux points sont nécessaires à la fin d'une instruction **if, elif, else, while, for**
C'est-à-dire chaque fois qu'on s'apprête à entrer des instructions soumises au test d'une condition ou devant être répétées un certain nombre de fois.
Le **if** , le **else** et le **elif** sont écrits au même niveau.
Mais ils subissent un retrait par rapport au **while** ou au **for** quand ils sont inclus dans ces structures.
Il est également envisageable qu'un **while** ou un **for** soit inclus dans un **if** et donc situés en retrait par rapport au **if**, au **else** ou au **elif**.

Règles d'indentation (niveau des instructions par rapport au début de ligne)



C'est le retrait qui est important, pas le nombre d'espaces qui le constitue.
Les instructions en noir ou en jaune ne doivent être précédées d'aucun espace.

Imbrication d'un if / else dans une structure for ou un while
Chaque ligne colorée est une instruction qui selon son retrait est exécutée
En gris à chaque tour de boucle
En rouge à chaque tour de boucle mais en fonction du résultat du test if/else
En jaune quand le for où le while se termine et que le programme reprend son cours.



Imbrication d'une boucle for (ou While) dans une structure if / else
Les instructions en gris foncé et en rouge sont réalisées si la condition testée par le if est vraie.
Mais les instructions en rouge bouclent en fonction de la plage d'indices du for.
Les instructions en gris clair sont réalisées si la condition testée par le if est fausse.
L'instruction en jaune est indépendante du if. Le programme reprend son cours.

Fonctions

Certaines fonctions sont incluses dans Python

On n'a vu que celles qui nous étaient les plus utiles. Il y en a beaucoup d'autres.
On trouvera une documentation sur les fonctions natives à cette adresse.
https://docs.python.org/fr/3/library/functions.html
En voici un aperçu:

		Fonctions natives		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Certaines fonctions sont dans des modules qui doivent être importés

Exemples
sqrt (n) =$\sqrt{n}$
sin(x) = sinus (x)

Import math (pour importer les fonctions du module math sur nombres réels)
Import cmath (pour importer les fonctions du module cmath les nombres complexes)

Sqrt() étant dans les deux modules pour éviter les conflits on écrit
Print ("racine carrée de –4 ", cmath.sqrt(–4))

Import Random (Pour importer le module générateur de nombres aléatoires)

Fonctions programmées par l'utilisateur

On peut définir une variable comme x, d ou n avant l'appel d'une fonction. Dans ce cas elle est globale et on peut l'utiliser dans la fonction si elle ne fait pas partie des arguments passés à la fonction.
Si on définit une variable dans la fonction elle est locale et ne peut être utilisée à l'extérieur qu'en tant que valeur renvoyée par la fonction dans le return (ou par un print).
Exemple: on définit f(x) avec un print(x). En dehors de la fonction on définit x = 7. Ensuite on appelle f(3). La fonction affiche 3. Ensuite on fait un print (x) ce qui affiche 7. La variable n'a pas été modifiée.
Si on appelle la fonction f(y) et qu'elle contient un print(x) quand on fait f(3) elle affiche 7. La valeur externe de x lui est accessible.

```
Def f(x):
    Return 3*x*x+2*x-7
```

Cette fonction reçoit le nombre stocké dans la variable x
et retourne la valeur de $3x^2+2x-7$ quand on remplace x par ce nombre

Donc si dans le corps du programme on écrit **a = f(1)** a sera égal à –2.
Si on teste **if f(1) > 0** le résultat sera false (faux)

```
Def   diviseur(d,n):
    Return n%d==0
Suite du programme
```

la fonction diviseur est définie comme devant recevoir les 2 nombres d,n diviseur(d,n)= true si d diviseur de n , sinon diviseur(d,n) retourne false.

```
def premier(n):
    d = 1
    nbDiviseurs = 0
    while d <= n:
        if diviseur(d, n):
            nbDiviseurs += 1
        d += 1
    return nbDiviseurs == 2
```

On passe le nombre n à la fonction premier()

La fonction passe tous les nombres d entre 1 et n à la fonction diviseur pour voir s'ils divisent n.
Si diviseur(d,n) = true on augmente le nombre de diviseurs

Si le nombre de diviseurs est 2 n est un nombre premier→
La fonction premier(n) retourne true (false si n n'est pas premier)

Fichiers

Actions élémentaires utiles au traitement des fichiers "texte"

Ouvrir un fichier

f=open("nom de fichier", "mode")

mode peut être égal à "w" (écriture seule) "r" (lecture seule, par défaut), "a" (ajout à la fin), "r+" (écriture /lecture)
mode binaire pour lire les fichiers autres que texte .(jpeg, .exe).On ajoute un b au mode "rb" "rb+"

Fermer le fichier

f.close()

Lire le fichier

a=f.read(taille) lit le nombre d'octets précisés par le nombre taille (résultat dans la variable a)

a=f.read() lit la totalité du fichier

a=f.readline() lit une ligne

Boucle de lecture

```
for line in f:
        print(line,end=" ")
```

Ecrire fichier

f.write("Ceci est une ligne \n") le \n signale la fin de ligne.

Import os facilite le travail sur les répertoires

Os.getcwd() permet de récupérer le chemin courant

Os.chdir("nouveau chemin") permet de changer de répertoire

Random

Import random

random.seed() initialise le générateur selon l'horloge système

random,random() retourne un nombre entre 0.0 et 1.0 (1.0 exclu)

random.randint(0,10) entier au hasard entre 0 et 10 inclus

random.sample(population,k) choisit au hasard k éléments de la population (exemple sample(1000000,60))

random.gauss(moyenne , écart type) réitéré donne des nombres obéissant à cette distribution

Matplotlib et Numpy

Installation

Sous Windows clic droit sur  cliquer sur **exécuter** tapez **cmd.exe**

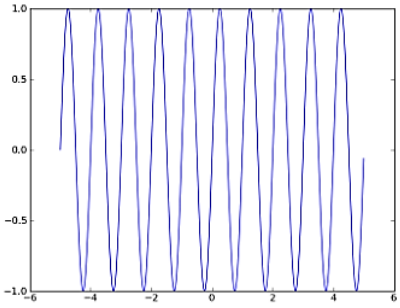
Toutes les commandes suivantes sont tapées dans la fenêtre noire qui apparaît, elles doivent se terminer par la touche [entrée] et leur exécution se termine quand on retrouve le prompt (invite de commande indiquant le chemin local suivi de >)
Tapez :
cd < Répertoire "scripts" sous le répertoire d'installation de python > pour vous rendre sous ce répertoire
Par exemple :
cd C:\Users\<votre nom>\AppData\Local\Programs\Python\Python36-32\Scripts
Tapez
python –mpip install –U pip
Attendez le retour de l'invite d'entrée >
Tapez
python –mpip install matplotlib
Attendez le retour de l'invite
Fermez la fenêtre noire. Lancez python.

Utilisation

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

>>> plt.figure()
>>> x = np.arange(-5,5, .01)
>>> y = np.sin(2*np.pi*x)
>>> plt.plot(x,y)
>>> plt.savefig('exemple_matplotlib.png')
```

Le fichier "exemple_matplotlib.png" doit alors apparaître dans votre répertoire courant.



Import nupy as np et **import matplotlib.pyplot as plt** : On importe les paquets utiles en leur donnant un alias plus court pour les invoquer. Celui de numpy sera "**np**", celui de matplotlib.pyplot sera" **plt**".
En gros, numpy fournit fonctions, plages et pas du repère, pyplot fournit le dessin.
plt.figure() signale le début de la construction d'une figure
x=np.arange(-5,5,.01) signale que sur le graphique la plage des x sera]-5 , +5[et le 0.01 signale qu'on divise en pas de 0.01 cette plage ce qui fait que la fonction de traçage va faire les calculs pour x=-4,99, -4,98, -4,97 etc ..
x=np.linspace(-5,5,1000) diviserait l'intervalle en 1000 intervalles égaux pour donner le pas de x
y=np.sin(2*np.pi*x) définit la fonction à tracer y=sin(2πx). np.pi = π donné par numpy. Ce qui fait 10 périodes de sin(x).
plt.plot(x,y) dessine la fonction quand x varie de –5 à +5 par pas de 0,01 selon les instructions du "arange".
plt.xlabel("légende axe des x") et **plt.ylabel("légende axe des y")** ajoutent des légendes à la figure
plt.show() montrerait la figure.
plt.savefig() sauvegarde la figure sous la forme d'un fichier image .png situé dans le répertoire courant de python.