1. Langage Python

1.1 Introduction 5
1.2 Installation de Python 7
1.3 Syntaxe de base 9
1.4 Listes et tuples 13
1.5 Chaîne de caractères 15
1.6 Structures de contrôle 17
1.7 Fonctions 20
1.8 Modules 22

1.1 Introduction

Tout le monde utilise un ordinateur mais comment nous en servir pour approfondir nos connaissances en mathématiques? Pour cela dans ce livre nous avons choisi le **langage de programmation Python**. Il existe en effet des centaines de langages de programmation alors pourquoi choisir Python ? avant de répondre à cette question on va d'abord vous expliquer ce qu'est un langage de programmation. C'est un langage qui sert à décrire les actions qu'un ordinateur doit réaliser. Pour cela, l'ordinateur utilise des programmes. Au début les programmeurs écrivaient des programmes en binaire.
Il constitués de suites de 0 et 1 ordonnées de façon précise, afin que le processeur sache quoi faire.
Par exemple 0100001001101111011011100110101001101111011101010111 0010 signifie **Bonjour**.
Alors discuter en binaire avec un ordinateur peut être très très long. Heureusement pour nous, aujourd'hui les développeurs écrivent leurs programmes sous forme de textes, avec une syntaxe particulière à respecter, mais toujours convertis en langage machine pour exécution.

Python est un langage de programmation créé par **Guido van Rossum**, dont la première version est sortie en 1991. Ce langage est désormais géré par la Python Software Foundation `https://www.python.org/psf/`. Python n'est pas une référence au serpent python, il tire son nom de la série télévisée les **Monthy Python** dont van Rossum est fan. En juillet 2018, Van Rossum a démissionné en tant que leader de la communauté après 30 ans.

O.henaoui

Le langage Python est un très bon choix pour débuter en programmation car c'est

- un langage de haut niveau, il demande peu de connaissance sur le fonctionnement de l'ordinateur.

- un langage open-source Python est gratuit et son environnement est riche en librairies.

- un langage multiplateforme, Python fonctionne sur toutes les plateformes les plus courantes, Windows, Linux et Mac Os, ainsi que sur des plateformes mobiles telles que Maemo ou Android.

- le langage de programmation le plus facile à apprendre.

La syntaxe du code utilise l'indentation et elle obéit à moins de règles par rapport à d'autres langages (pas d'accolades, crochets, points-virgules, etc...), ce qui facilite la lecture et la compréhension. Python a une syntaxe qui permet aux développeurs d'écrire des programmes avec moins de lignes.

```
1 print("Hello, World!")
2
3
4
5
```

programme en Python

```
1 int main()
2 {
3     printf("Hello, World! \n");
4     return 0;
5 }
```

programme en C

- un langage à typage dynamique, le programmeur n'a pas besoin de déclarer le type des variables.

- un langage utilisé dans de nombreux domaines

* *L'administration système*: Python est utilisé pour écrire des scripts qui peuvent automatiser des tâches, telles que la recherche dans votre système de fichiers, l'accès à Internet, l'analyse des types de fichiers.

* *Le prototypage rapide d'applications*: Le prototypage consiste à concevoir des versions intermédiaires et donc incomplètes des gros projets conçues pour tester l'utilisation avant la phase finale.

* *La recherche scientifique*: Python dispose de nombreuses bibliothèques (scipy, numpy et matplotlib...) puissantes en calcul numérique permettant d'aborder efficacement un problème scientifique.

* *Les applications de gestion*: Il existe des logiciels en Python qui traitent la gestion de stocks, etc.

* *Les sites et les applications web*: Python est utilisé dans de nombreuses applications et sites web comme YouTube, Microsoft, Instagram, Pinterest, Reddit et Spotify, et est l'un des langages officiels de Google (Guido van Rossum a travaillé pour Google de 2005 à 2012).

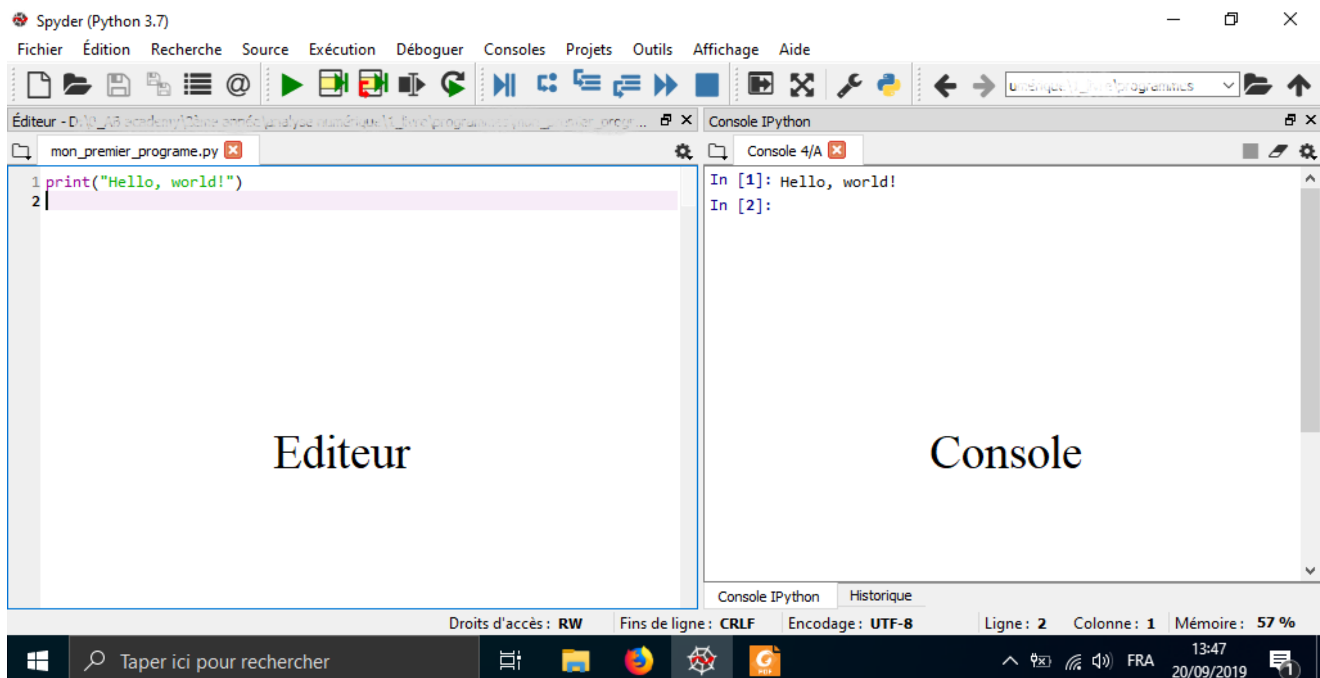
Dans le monde industriel NASA, EDF, CEA, AirBus utilisent tous Python dans des domaines très variés.

* *Jeux vidéos*: Il existe des bibliothèques (pyGame...) facilitant la création de jeux vidéo en 2D et 3D.

- le langage de programmation le plus populaire, le 6 septembre 2019, Le magazine IEEE Spectrum (Institute of Electrical and Electronics Engineers), la plus grande association mondiale de professionnels techniques, a publié la sixième édition de son classement annuel des meilleurs langages de programmation, qui place encore Python au sommet pour la troisième année consécutive.

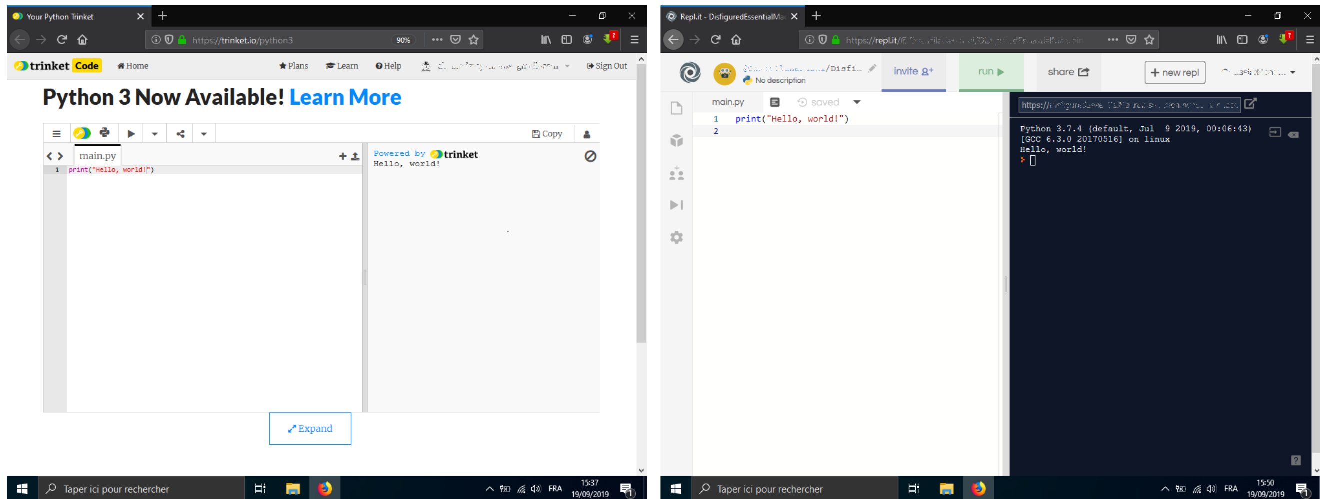
1.2 Installation de Python

L'installation de Python est assez simple, il suffit de télécharger gratuitement la version qui correspond à votre système d'exploitation à partir du site web officiel à l'adresse: http://www.Python.org. La dernière version de Python est la version3. Il existe quelques différences importantes entre la version2 et la version3, c'est pour quoi la version2 cessera d'être maintenue après le 1er janvier 2020. L'installation comporte le langage en lui-même ainsi qu'un environnement de développement (IDLE). Comme tout environnement Python comporte une console et un éditeur.
* **La console** (avec l'invite de commande) permet d'exécuter des instructions, des calculs etc.
* **L'éditeur** de fichiers permet de taper le texte d'un programme.
L'exécution du programme (commande Run) dans l'éditeur affiche le résultat dans la console.
Si certains modules complémentaires s'avèrent nécessaires, ils sont disponibles en téléchargement gratuit à l'adresse https://docs.python.org/fr/3/installing.
L'installation d'un environnement Python complet peut-être une vraie galère car il faut télécharger les modules dont on a besoin un à un. Pour cela nous vous conseillons d'installer une distribution contenant des librairies de modules scientifiques, par exemple la distribution **Anaconda** disponible à l'adresse https://store.continuum.io. Elle contient un interpréteur Python, un IDE environnement de développement **Spyder** ainsi qu'un gestionnaire de paquets puissant appelé **Conda** permettra d'installer et mettre à jour des modules qui seront utilisés dans ce livre comme **NumPy** pour la manipulation de matrices, **SciPy** pour le calcul scientifique, **Matplotlib** pour la création de figures.



Différents sites proposent d'exécuter un programme en Python directement depuis un navigateur. Ils permettent de disposer dans un simple navigateur web d'un éditeur et d'un interpréteur Python sans aucune installation. Cela permet notamment aux étudiants de réaliser leurs programmes chez eux sans avoir à installer quoi que ce soit.

Par exemple `https://trinket.io/python3` et `https://repl.it/languages/python3`

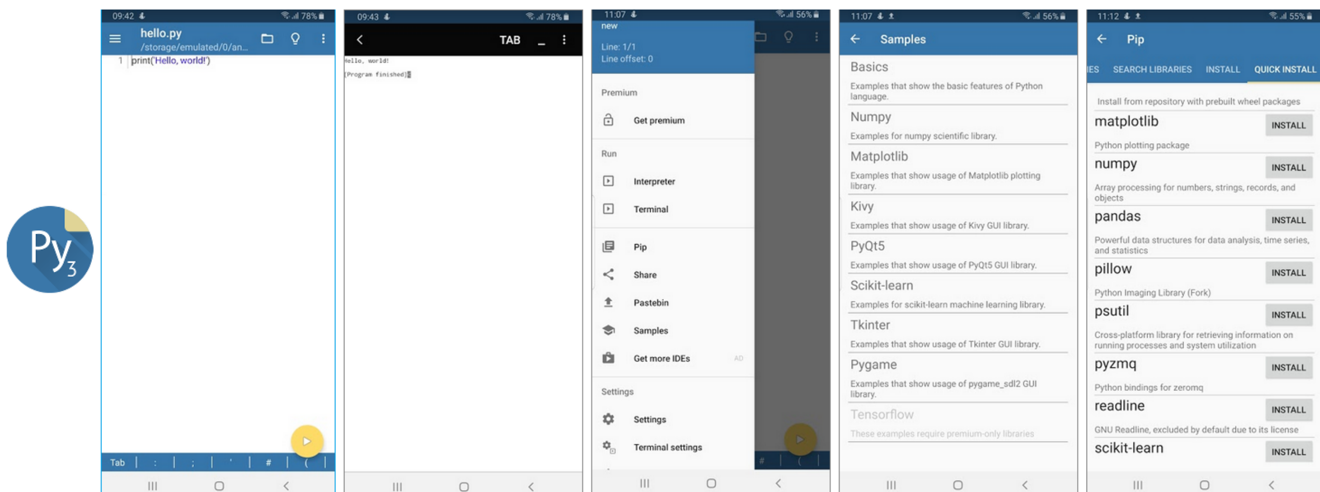


L'interface de programmation est divisée en deux parties :

- à gauche, la fenêtre de script pour écrire des programmes complets qui pourront être sauvegardés;
- à droite, la console dans laquelle le code est exécuté.

L'outil **Python Tutor** en ligne via la page `http://pythontutor.com` va permettre aux étudiants de détailler comment un code Python s'exécute ligne par ligne.

L'application **Pydroid 3** est le moyen éducatif le plus simple et le plus puissant à utiliser pour Android.



O.henaoui

1.3 Syntaxe de base

Un langage de programmation est donc un langage qui sert à écrire un script pour donner l'ordre à un ordinateur de faire ce que vous voulez. Nous allons commencer par donner des ordres très simples.

1.3.1 Commandes de base

print() permet d'afficher, lors de l'exécution d'un programme, des phrases, des valeurs, des calculs ou tout élément interprétable. Dans la version 3 de Python, cette commande est devenue une fonction. On peut avoir une instruction sur plusieurs lignes en utilisant le caractère back-slash \ en fin de lignes. Il existe des caractères spéciaux qui ne seront pas affichés: $\backslash t$ tabulation; $\backslash n$ saut à la ligne;...

```
1 print("Python3:\n\n Hello,\t world!")
2
```

```
In [1]: Python3:

Hello,  world!
In [2]:
```

help() permet d'obtenir de l'aide sur des fonctions prédéfnies ou sur des modules tout entiers.

```
1
```

```
In [1]: help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
In [2]:
```

dir() affiche la liste de toutes les fonctions et variables disponibles dans un module.

1.3.2 Commentaire

Pour bien expliquer un script (ou un programme) et faciliter sa lecture, il doit être bien commenté. Tout ce qui suit le caractère dièse # est ignoré par Python jusqu'à la fin de la ligne et est considéré comme un commentaire. Les commentaire multi-lignes sont précédé et suivi de trois apostrophes ' ' ' ou trois guillemets " " ".

```
1 # Mon premier script
2 print("Hello, world!") # Bonjour le monde
3
```

```
In [1]: Hello, world!

In [2]:
```

1.3.3 Données et variables

Il est pratique d'enregistrer les **données** saisies par le programmeur dans la mémoire de l'ordinateur pour pouvoir les réutiliser. Une **variable** est un espace mémoire dans lequelle une donnée est stockée. Elle peut avoir un nom court (a,b,x,y,...) ou un nom plus descriptif (âge, mesure, volume,...).
Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :
* doit commencer par une lettre, aucun caractère spécial ou chiffre n'est permis, sauf underscore «_»;
* ne peut contenir que des lettres (aux majuscules et aux minuscules), des chiffres et underscore «_».
Il faut faire attention par exemple X et x sont deux variables différentes.
Il existe des **mots réservés** par Python que nous ne pouvons pas créer des variables portant ces noms.

and	as	assert	break	class	continue	def	del	elif	else	except
False	finally	for	from	global	import	if	in	is	lambda	none
nonlocal	not	or	pass	raise	return	True	try	while	with	yield

1.3.4 Types de données

En Python, il n'est pas nécessaire de déclarer les variables en précisant leur type. Lors des opérations, le langage associe à la variable le type de la donnée et on l'obtient à l'aide de la commande **type()**.
* Un nombre entier (**int**) est un nombre positif ou négatif, sans décimales.
* Un nombre à virgule flottante (**float**) il est mémorisé avec une précision fixée (15 chiffres significatifs). Exemple 2.55 ou 255e-2 avec un "e" pour indiquer la puissance de 10 (notation scientifique).
* Un nombre complexe (**complex**) s'écrit donc a + bj, avec a et b des variables de type float.
* Une chaîne de caractères (**str**) est écrite entre simples ou doubles-quotes. Exemple: "Python3".
* Une expression booléenne (**bool**) a deux valeurs possibles True ou False.
* Une liste (**list**) est une collection ordonnée de variables séparé par des virgules et modifiable.
* Un tuple (**tuple**) est une liste qui ne peut plus être modifiée.

```
1 print(2,"\t\t",type(2))                    # entier
2 print(2.5,"\t\t",type(2.5))                # réel
3 print(1 + 3j,"\t\t",type(1 + 3j))          # complexe
4 print("Python3","\t",type("Python3"))      # chaîne de caractères
5 print(2>5,"\t\t",type(2>5))                # expression booléenne
6 print([1,2,3],"\t",type([1,2,3]))          # liste
7 print((1,2,3),"\t",type((1,2,3)))          # tuple
8
```

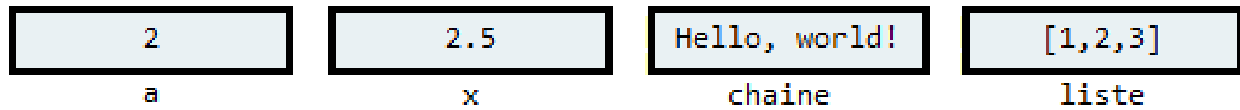
```
In [1]:
2          <class 'int'>
2.5        <class 'float'>
(1+3j)     <class 'complex'>
Python3    <class 'str'>
False      <class 'bool'>
[1, 2, 3]  <class 'list'>
(1, 2, 3)  <class 'tuple'>
In [2]:
```

On peut changer le type d'une donnée en l'indiquant entre parenthèses, précédée du nom du nouveau type souhaité. Par exemple: float(3) c'est-à-dire l'entier 3 devient un réel 3.0.

1.3.5 Opérateurs

Un opérateur est un symbole (ou mot réservé) utilisé pour effectuer une opération entre des variables.

1) Opérateur d'affectation "=": il faut bien comprendre qu'il ne s'agit en aucune façon d'une égalité. Une variable est une sorte de boite virtuelle dans laquelle on peut mettre une donnée.




```
1 a=2                 # affecter la valeur 2 à la variable a
2 print(a)
3 x=2.5               # affecter la valeur 2.5 à la variable x
4 print(x)
5 chaine="Hello, world!" # affecter "Hello, world!" à la variable chaine
6 print(chaine)
7 liste=[1,2,3]       # affecter [1,2,3] à la variable liste
8 print(liste)
9
```

```
In [1]:
2
2.5
Hello, world!
[1, 2, 3]

In [2]:
```

On peut affecter une valeur à plusieurs variables simultanément, par exemple $a = b = 2$. On peut aussi faire des affectations multiples en utilisant une seule fois l'opérateur par exemple $a, b, c = 1, 5, 0.3$, c'est-à-dire les variables a, b et c prennent simultanément les valeurs 1, 5 et 0.3.

Il est possible de faire l'affectation multiple précédente en utilisant les tuples: $(a, b, c) = (1, 5, 0.3)$.

```
1 a1,b1,c1=1,5,0.3      # affectation multiple
2 (a2,b2,c2)=(1,5,0.3) # affectation multiple en utilisant les tuples
3 print(a1,b1,c1,"\t",(a1,b1,c1))
4 print(a2,b2,c2,"\t",(a2,b2,c2))
5
```

```
In [1]:
1 5 0.3   (1, 5, 0.3)
1 5 0.3   (1, 5, 0.3)

In [2]:
```

Commande input permet de demander à entrer une donnée au clavier et à terminer avec <Enter>. Elle renvoie toujours une chaîne de caractères, que l'on doit convertir en entier ou en flottant si on souhaite entrer une valeur numérique.

```
1 # 1ère méthode
2 a=input("entrer une valeur:") # input renvoie une chaîne de caractères
3 print(a,"\t",type(a))        # a est de type str
4 b=float(a)                   # conversion de la chaîne en un réel
5 print(b,"\t",type(b))        # b est de type float
6 # 2ème méthode
7 c=float(input("entrer une valeur:"))# c est de type float
8 print(c,"\t",type(c))
9
```

```
In [1]:
entrer une valeur:5
5   <class 'str'>
5.0 <class 'float'>

entrer une valeur:5
5.0 <class 'float'>

In [2]:
```

2) Opérateurs mathématiques

```
1 print("2+3=",2+3)    # + addition
2 print("2-3=",2-3)    # - soustraction
3 print("2*3=",2*3)    # * multiplication
4 print("2**3=",2**3)  # ** puissance (Exposants)
5 print("2/3=",2/3)    # /  division
6 print("2//3=",2//3)  # // partie entière de la division
7 print("2%3=",2%3)    # %  reste de la division
8
```

```
In [1]:
2+3= 5
2-3= -1
2*3= 6
2**3= 8
2/3= 0.6666666666666666
2//3= 0
2%3= 2
In [2]:
```

Ordre de traitement des opérations Lorsqu'il existe plusieurs opérateurs dans une expression, l'interpréteur utilise l'ordre dit « PEDMAS » (Parenthèses, Exposants, Division, Multiplication, Addition, Soustraction) qui reprend les lois associatives et commutatives de l'algèbre.

```
1 # Ordre de traitement des opérations PEDMAS
2 # (5-3*7**2)=(5-3*49)=(5-147)=(-142)
3 # 6/2+4*(5-3*7**2)=6/2+4*(-142)=6/2+-568=3.0+-568=-565.0
4 print("6/2+4*(5-3*7**2)=",6/2+4*(5-3*7**2))
5
```

```
In [1]:
6/2+4*(5-3*7**2)= -565.0
In [2]:
```

On peut appliquer aux chaînes de caractères et aux listes les opérateurs + et * :

```
1 print("Langage "+"Python")
2 print(10*"a")
3 name=input("Entrer votre nom: ")
4 print(3*"*","Bienvenue "+name,3*"*")
5 print([1,2,"a"]+["b","c"],3*[1,2])
6
```

```
In [1]:
Langage Python
aaaaaaaaaa
Entrer votre nom: mustapha
*** Bienvenue mustapha ***
[1, 2, 'a', 'b', 'c'] [1, 2, 1, 2, 1, 2]
In [2]:
```

3) Opérateurs d'assignation Soit une variable à laquelle on a assigné (affecté) une valeur. Si on lui assigne ensuite une autre valeur, Python supprimera alors automatiquement l'anciènne variable et garde la nouvelle donnée.

```
1 x,y,z,t,p,q,r=2,3,5,6,4,11,7
2 print(x,y,z,t,p,q,r)
3 x+=2    # x=x+2
4 y-=2    # y=y-2
5 z*=2    # z=z*2
6 t**=2   # t=t**2
7 p/=2    # p=p/2
8 q//=2   # q=q//2
9 r%=2    # r=r%2
10 print(x,y,z,t,p,q,r)
11
```

```
In [1]:

2 3 5 6 4 11 7
4 1 10 36 2.0 5 1

In [2]:
```

O.henaoui

4) Opérateurs de comparaison

```
1 x,y=2,5
2 print(x == y,"\t",x,"est égal à",y)
3 print(x != y,"\t",x,"est différent de",y)
4 print(x > y ,"\t",x,"est plus grand que",y)
5 print(x < y ,"\t",x,"est plus petit que",y)
6 print(x >= y,"\t",x,"est plus grand ou égal à",y)
7 print(x <= y,"\t",x,"est plus petit ou égal à",y)
8
```

```
In [1]:
False  2 est égal à 5
True   2 est différent de 5
False  2 est plus grand que 5
True   2 est plus petit que 5
False  2 est plus grand ou égal à 5
True   2 est plus petit ou égal à 5
In [2]:
```

5) Opérateurs logiques Les expressions avec un opérateur logique sont évaluées à "True" ou "False".

p	q	not p	p and q	p or q
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

```
1 print(1>2 or 5>2)   # |p (False)|q (True)|p or q (True)
2 print(1>2 and 5>2)  # |p (False)|q (True)|p and q (False)
3 print(not 1>2)      # |p (False)         |not p (True)
4
```

```
In [1]:
True
False
True
In [2]:
```

1.4 Listes et tuples

Une liste est une collection d’éléments séparés par des virgules et l’ensemble est entre des crochets.

```
L= [1500, 'ab', 1.25, 'py', 1.25]
    L[0]  L[1]  L[2]  L[3]  L[4]
```

```
1 L=[] # liste vide
2 L=[1500,"ab",1.25,"py",1.25] # affectation
3 print("L=",L)
4 print("len(L)=",len(L))# nombre d'éléments de la liste
5 print("L[1]=",L[1])  # afficher le 2ème élément de la liste
6 print("L[2]=",L[2])  # afficher le 3ème élément de la liste
7 print("L[-2]=",L[-2])  # L[-2]=L[len(L)-2]=L[3]
8 print("L[1:4]=",L[1:4])# L[1:4]=[L[1],L[2],L[3]]
9 print("L[2: ]=",L[2:]) # L[2:]=[L[2],L[3],L[4]]
10 print("ab dans L:","ab" in L)  #"ab" appartient à la liste?
11 print(" 3 dans L:",3 in L)     #  3  appartient à la liste?
12 print("i(1.25) =",L.index(1.25)) # index du premier 1.25
13 print("count(1.25)=",L.count(1.25)) # combien d'élément
14
```

```
In [1]:
L= [1500, 'ab', 1.25, 'py', 1.25]
len(L)= 5
L[1]= ab
L[2]= 1.25
L[-2]= py
L[1:4]= ['ab', 1.25, 'py']
L[2: ]= [1.25, 'py', 1.25]
ab dans L: True
 3 dans L: False
i(1.25) = 2
count(1.25)= 2
In [2]:
```

O.henaoui

1) Modifier une liste pour jouter ou suprimer une valeur à une liste python on utilise:

append(e) Permet d'ajouter un élément e en fin de liste.

insert(i, e) Permet d'insérer un élément e à la position i.

remove(e) Permet de suprimer le premier élément de la liste qui a la même valeur que e.

pop(i) Retire et renvoie l'élément de la liste d'index i. Si i n'est pas fourni, il traite le dernier élément.

reverse() Inverser l'ordre des éléments

```
1 L=[1500,"ab",1.25,"py",1.25]
2 print("L=",L)
3 L[1]=36          # Modifier le 2ème élément
4 print("L=",L)
5 L[2]=L[2]+1      # ajouter 1 au 3ème élément
6 print("L=",L)
7 del L[3]         # supprime 4ème élément
8 print("L=",L)
9 L.append("vie") # ajouter "vie" à la fin
10 print("L=",L)
11 L.insert(2,"cd") # ajouter l'élément "cd" à L[2]
12 print("L=",L)
13 L.remove(2.25)   # suprimer l'élément 2.25
14 print("L=",L)
15 L.pop()          # suprimer le dernier élément
16 print("L=",L)
17 L.pop(1)         # suprimer l'élément d'index 1
18 print("L=",L)
19 L.reverse()      # inverser l'ordre des éléments
20 print("L=",L)
21
```

```
In [1]:
L= [1500, 'ab', 1.25, 'py', 1.25]
L= [1500, 36, 1.25, 'py', 1.25]
L= [1500, 36, 2.25, 'py', 1.25]
L= [1500, 36, 2.25, 1.25]
L= [1500, 36, 2.25, 1.25, 'vie']
L= [1500, 36, 'cd', 2.25, 1.25, 'vie']
L= [1500, 36, 'cd', 1.25, 'vie']
L= [1500, 36, 'cd', 1.25]
L= [1500, 'cd', 1.25]
L= [1.25, 'cd', 1500]
In [2]:
```

2) Copier une liste Attention! la commande $S = L$ ne crée pas une nouvelle variable S mais simplement une référence vers L. Ainsi, tout changement effectué sur S sera répercuté sur L aussi !

```
1 L=[1500,"ab",1.25,"py",1.25]
2 S=L
3 S[2]="vie"
4 print("L=",L)
5 print("S=",S)
6
```

```
In [1]:
L= [1500, 'ab', 'vie', 'py', 1.25]
S= [1500, 'ab', 'vie', 'py', 1.25]
In [2]:
```

Alors comment copier une liste qui sera indépendante?

```
1 L=[1500,"ab",1.25,"py",1.25]
2 S=L[:]
3 S[2]="vie"
4 print("L=",L)
5 print("S=",S)
6
```

```
In [1]:
L= [1500, 'ab', 1.25, 'py', 1.25]
S= [1500, 'ab', 'vie', 'py', 1.25]
In [2]:
```

O.henaoui

3) Tuple: Les opérations que l'on peut effectuer sur des tuples sont syntaxiquement similaires à celles que l'on effectue sur les listes, sauf qu'ils sont des listes non-modifiables.

```
1 L=(1500,"ab",1.25,"py",1.25)
2 L[2]="vie"
3 print("L=",L)
4
```

```
In [1]:
Traceback (most recent call last):
  File "[illegible]", line 2, in <module>
    L[2]="vie"
TypeError: 'tuple' object does not support item assignment
In [2]:
```

4) Range : La fonction range() renvoie une séquence de nombres entiers de valeurs croissantes (c'est-à-dire que range(n) génère les nombres de 0 à n-1) qu'on peut convertir en une liste avec la fonction **list()**, ou en tuple avec la fonction **tuple()**. On peut aussi utiliser range(m,n) qui génère les nombres de m à n, ou bien range(m,n,p) qui génère les nombres de m à n avec le pas p pour passer d'une valeur à la suivante.

```
1 print(list(range(8)))
2 print(tuple(range(8)))
3 print(list(range(2,8)))
4 print(list(range(0,8,2)))
5 print(list(range(0,-8,-2)))
6
```

```
In [1]:
[0, 1, 2, 3, 4, 5, 6, 7]
(0, 1, 2, 3, 4, 5, 6, 7)
[2, 3, 4, 5, 6, 7]
[0, 2, 4, 6]
[0, -2, -4, -6]
In [2]:
```

1.5 Chaîne de caractères

Python considère qu'une chaîne de caractères comme une collection ordonnée d'éléments. Chaque caractère peut être désigné à l'aide d'un index. On peut convertir la en une liste avec la fonction **list()**.

```
1 ch="Python 3"
2 print(ch)
3 print("ch[2]=",ch[2],type(ch[2]))
4 print("ch[6]=",ch[6],type(ch[6]))
5 print("ch[7]=",ch[7],type(ch[7]))
6 print(list(ch))
7
```

```
In [1]:
Python 3
ch[2]= t <class 'str'>
ch[6]=   <class 'str'>
ch[7]= 3 <class 'str'>
['P', 'y', 't', 'h', 'o', 'n', ' ', '3']
In [2]:
```

split() : convertit une chaîne en une liste de sous-chaînes. **join()** : rassemble une liste de chaînes.

```
1 chaine="Language de programmation Python"
2 print(chaine)
3 print(chaine.split())
4 print("".join(chaine))
5
```

```
In [1]:
Language de programmation Python
['Language', 'de', 'programmation', 'Python']
Language de programmation Python
In [2]:
```

Formatage de chaîne est un outil très puissant permettant de créer des chaînes de caractères en remplaçant certains champs (entre accolades) par des arguments. Les champs peuvent contenir des numéros d'ordre (0,1,...) pour désigner précisément lesquels des arguments transmis à format() devront les remplacer. Ils peuvent aussi contenir des indications de formatage (la précision, la notation scientifique, le nombre total de caractères...).

```
1 r=3
2 print("L'aire d'un disque de rayon {} est \
3 {}.".format(r, 3.14*r**2))
4 print("L'aire d'un disque de rayon {0}={1} est\n \
5 aire=3.14*{0}^2={2}.".format("r",r, 3.14*r**2))
6 # autre méthode
7 print(f"L'aire d'un disque de rayon {r} est \
8 {3.14*r**2}.")
9
```

```
In [1]:
L'aire d'un disque de rayon 3 est 28.26.
L'aire d'un disque de rayon r=3 est
aire=3.14*r^2=28.26.
L'aire d'un disque de rayon 3 est 28.26.
In [2]:
```

La précision est représentée par un entier préfixé par un point, qui spécifie le nombre de chiffres significatifs après la virgule.

```
1 p=14.1592653589793
2 print("{0:s}= {1:2.2f}".format("p",p))
3 print("{0:s}= {1:2.6f}".format("p",p))
4 print("{0:s}= {1:e}".format("p",p))
5
```

```
In [1]:
p= 14.16
p= 14.159265
p= 1.415927e+01
In [2]:
```

```
1 print("{: <7d} | {: <7d} | {: <7d}".format(2,200,20000))
2 print("{:.<7d} | {:.<7d} | {:.<7d}".format(2,200,20000))
3 print("{:.^7d} | {:.^7d} | {:.^7d}".format(2,200,20000))
4 print("{:.>7d} | {:.>7d} | {:.>7d}".format(2,200,20000))
5
```

```
In [1]:
2       | 200     | 20000
2...... | 200.... | 20000..
...2... | ..200.. | .20000.
......2 | ....200 | ..20000
In [2]:
```

Ancienne méthode:

```
1 pi=3.141592653589793
2 print("pi=",pi)              # %s chaine de caractère
3 print("%s= %d" % ("pi",pi))    # %d entier décimal signé
4 print("%s= %1.2f" % ("pi",pi)) # %f float
5 print("%s= %1.6f" % ("pi",pi))
6 print("%s= %e" % ("pi",pi))    # %e notation scientifique
7
```

```
In [1]:
pi= 3.141592653589793
pi= 3
pi= 3.14
pi= 3.141593
pi= 3.141593e+00
In [2]:
```

Cette méthode s'avère parfois complexe lorsqu'il y a beaucoup d'éléments à remplacer. Par exemple:

```
1 x,y=input("entrer deux entiers: ").split()
2 x,y=[int(x),int(y)]
3 print("\n{0}+{1}={2};  {0}-{1}={3};  \
4 {0}*{1}={4};  {0}^3={5};  {0}/{1}={6};\
5 ".format(x,y,x+y,x-y,x*y,x**3,x/y))
6
```

```
In [1]:
entrer deux entiers: 2 5
2+5=7;  2-5=-3;  2*5=10;  2^3=8;  2/5=0.4;
In [2]:
```

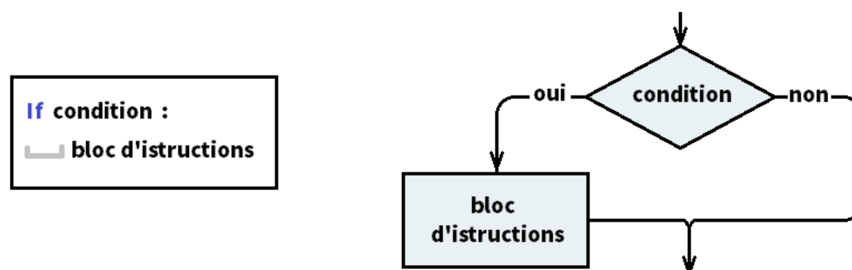
1.6 Structures de contrôle

Les structures de contrôle sont les groupes d'instructions (blocs d'instructions) qui déterminent l'ordre dans lequel les actions sont effectuées. Python utilise l'indentation (décalage vers la droite) pour faire apparaître ces blocs.

1.6.1 Instruction conditionnelle

Les instructions conditionnelles permettent d'effectuer une série d'instructions si une certaine condition est vérifiée. La syntaxe est la suivante :

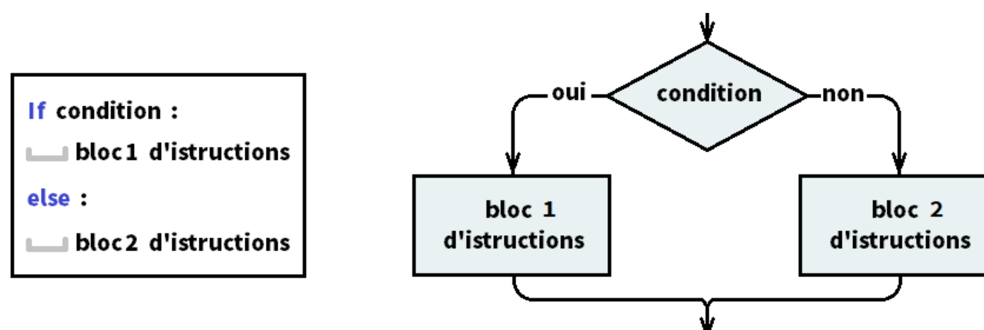
1) Test simple « si ..., alors ...»




```
1 x=int(input("Entrer un entier x: "))
2 if x<0:
3     print("x est négatif")
4 print("{0}^2={1}".format("x",x**2))
5
```

```
In [1]: Entrer un entier x: 5
x^2=25
In [2]: Entrer un entier x: -5
x est négatif
x^2=25
In [3]:
```

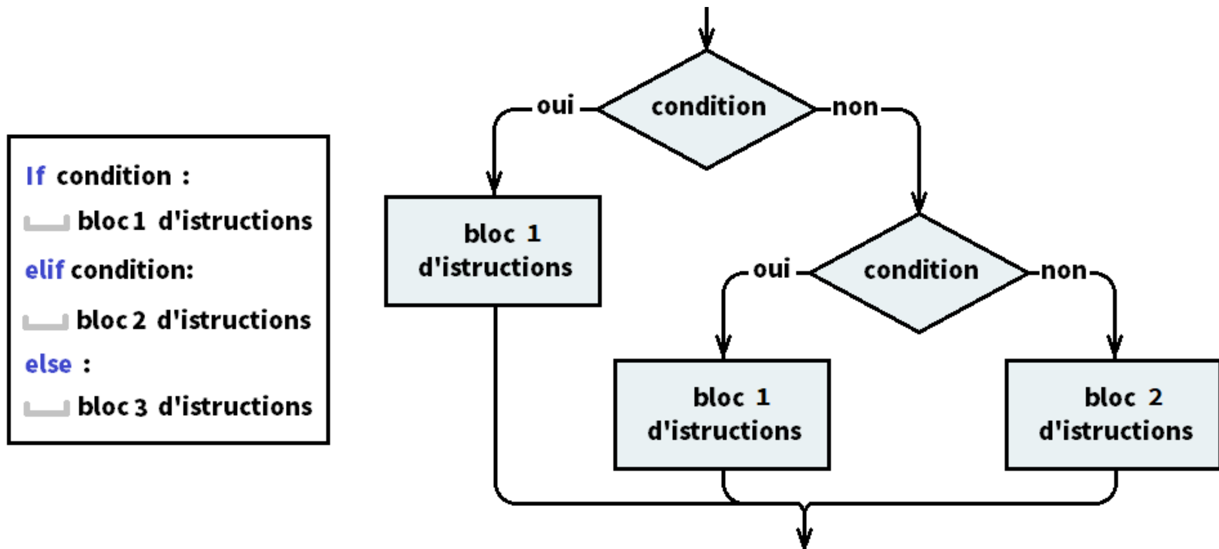
2) Alternative simple « si ..., alors ..., sinon ...»




```
1 x=int(input("Entrer un entier x: "))
2 if x<1:
3     print("x est inférieur à 1")
4 else:
5     print("x est supérieur à 1")
6
```

```
In [1]: Entrer un entier x: 5
x est supérieur à 1
In [2]: Entrer un entier x: -2
x est inférieur à 1
In [3]:
```

3) Alternative multiple « si ..., alors ..., sinon (si ..., alors ..., sinon ...)»




```python
x=int(input("Entrer un entier x: "))
if x<0:
    print("x est négatif")
elif x>0:
    print("x est positif")
else:
    print("x est nul")
print("{0}^2={1}".format("x",x**2))
```

```
In [1]: Entrer un entier x: -5
x est négatif
x^2=25
In [2]: Entrer un entier x: 0
x est nul
x^2=0
In [3]: Entrer un entier x: 5
x est positif
x^2=25
In [4]:
```

4) Expressions conditionnelles

Python offre la possibilité de former des expressions dont l'évaluation est soumise à une condition.

```python
x=int(input("Entrer un entier x: "))
print("x inférieur à 1" if x<1 else "x supérieur à 1")
```

```
In [1]: Entrer un entier x: 5
x supérieur à 1
In [2]: Entrer un entier x: -2
x inférieur à 1
In [3]:
```

```python
x=int(input("Entrer un entier x: "))
print("x négatif" if x<0 else "x positif" if x>0 else "x nul")
```

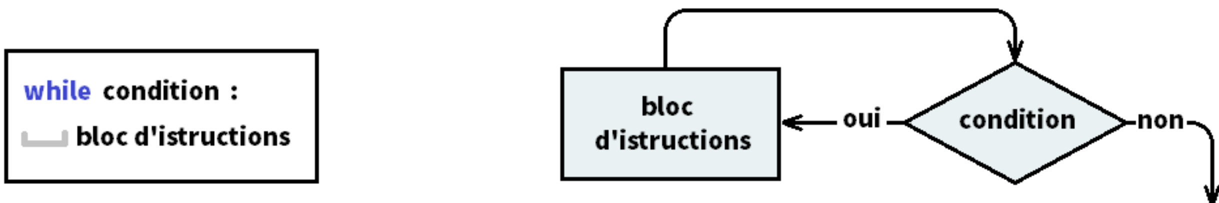
```
In [1]: Entrer un entier x: -5
x est négatif
x^2=25
In [2]: Entrer un entier x: 0
x est nul
x^2=0
In [3]: Entrer un entier x: 5
x est positif
x^2=25
In [4]:
```

O.henaoui

1.6.2 Instructions répétitives

Instructions répétitives sont des instructions qui permettent de répéter plusieurs fois une série d'instructions.

1) Répétitions conditionnelles (while) La boucle while permet d'exécuter un bloc d'istructions tant qu'une condition est vérifiée. Lorsque l'expression n'est plus vraie la boucle s'arrête.



Exemple: Calculer la somme des nombres de 1 à 5.

```python
s = 0
k = 1
while k <= 5 :
    s = s + k
    k = k + 1
print("la somme est:", s)
```

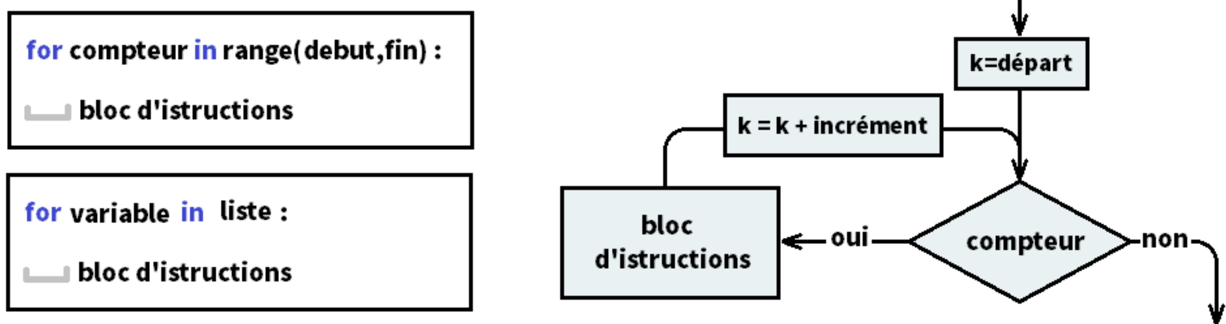
```
In [1]:
la somme est: 15

In [2]:
```

voici comment cela fonctionne

	k	s	
1 + 1	1	0	0 + 1
2 + 1	2	1	1 + 2
3 + 1	3	3	3 + 3
4 + 1	4	6	6 + 4
	5	10	10 + 5
	stop	15	

2) Répétitions inconditionnelles (boucles for)



Exemple: Calculer la somme des nombres de 1 à 5.

```python
s= 0
for k in range(1,6) :
    s= s + k
print("la somme est", s)
```

```
In [1]:
la somme est 15
In [2]:
```

voici comment cela fonctionne

k	s	
1	0	0 + 1
2	1	1 + 2
3	3	3 + 3
4	6	6 + 4
5	10	10 + 5
	15	

O.henaoui

La fonction **print()** affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne.

Si toutefois, on ne veut pas afficher ce retour à la ligne, on peut utiliser end

```
1 L=["l","i","r","e"]
2 print("les lettres sont:")
3 for k in L :
4     print(k)
5
```

```
In [1]: les lettres sont:
l
i
r
e
In [2]:
```

```
1 L=["l","i","r","e"]
2 print("le mot est: ")
3 for k in L :
4     print(k,end='')
5
```

```
In [1]: le mot est:
lire
In [2]:
```

Deux instructions supplémentaires permettent d'agir sur le déroulement de l'instruction

Break interrompre une boucle. Elle est utile si un élément d'une liste est une condition de sortie.

```
1 for x in range(1, 8):
2     if x == 5:
3         break
4     print(x, end=" ")
5 print("\nBoucle interrompue pour x =", x)
6
```

```
In [1]:
1 2 3 4
Boucle interrompue pour x = 5
In [2]:
```

Continue interrompre l'exécution de la boucle pour l'élément en cours et passe à l'élément suivant. Cette instruction est utile dans le cas où l'élément en cours n'est pas concerné par le traitement.

```
1 for x in range(1,8):
2     if x == 5:
3         continue
4     print(x, end=" ")
5 print("\nLa boucle a sauté la valeur 5")
6
```

```
In [1]:
1 2 3 4 6 7
La boucle a sauté la valeur 5
In [2]:
```

1.7 Fonctions

Une fonction est un bloc d'instructions, dépendant de certaines variables, qui peut être exécutée directement à l'aide de son nom. Elle permet de ne pas retaper plusieurs fois une même série d'instructions et elle peuvent être mises à disposition d'autres programmes. Une fonction possède :

- **un nom**: Vous pouvez choisir n'importe quel nom à l'exception des mots réservés du langage, il doit commencer par une lettre, aucune lettre accentuée ou caractère spécial n'est permis, sauf _.
- **des paramètres**, qui sont ses entréees ;
- **un résultat**, ou valeur de retour

```
def nom (paramètres) :
    bloc d'istructions
    return valeur
```

Dans le corps d'un programme, un appel de fonction est constitué du nom de la fonction suivi de parenthèses. On place dans ces parenthèses le ou les arguments pour chacun des paramètres.

O.henaoui

Exemple: la fonction **lambda** est utilisée lorsqu'une fonction se limite à une seule expression.

```
1 def f(x):
2     return 2*x**3+x**2+7
3 # appel de la fonction
4 x=float(input("Entrer une valeur:"))
5 print("f({0})={1}".format(x,f(x)))
6
```

```
1 f=lambda x:2*x**3+x**2+7
2 # appel de la fonction
3 x=float(input("Entrer une valeur:"))
4 print("f({0})={1}".format(x,f(x)))
5
```

```
In [1]:
Entrer une valeur:-1.5
f(-1.5)=2.5
In [2]:
```

Exemple: Ecrire une fonction qui affiche le nombre de solutions réelles de l'équation du second degré

$$ax^2 + bx + c = 0.$$

nbre_solutions.py

```
1 ######################################################################
2 #     Nombre de solutions réelles de l'équation du second degré       #
3 #                        ax^2 + b x + c = 0.                          #
4 ######################################################################
5 #                      fonction nbre_solutions                        #
6 ######################################################################
7 def nbre_solutions(a,b,c):
8     D = b**2 - 4*a*c
9     if a == 0:
10        print("L'équation est du premier degré.")
11    elif D > 0:
12        s="possède deux solutions réelle."
13    elif D== 0:
14        s="possède une solution réelle."
15    else:
16        s="ne possède pas de solution réelle."
17    return s
18 ######################################################################
19 #                        Appel de la fonction                         #
20 ######################################################################
21 a,b,c=input("Entrer les coefficients de l'équation:").split()
22 a,b,c=[int(a),int(b),int(c)]
23 print("L'équation", a,"x^2 +",b,"x +",c,"=0", nbre_solutions(a,b,c ))
24
```

Console 2/A

```
In [1]: Entrer les coefficients de l'équation:2 5 3
L'équation 2 x^2 + 5 x + 3 =0 possède deux solutions réelle.
In [2]: Entrer les coefficients de l'équation:3 2 7
L'équation 3 x^2 + 2 x + 7 =0 ne possède pas de solution réelle.
In [3]: Entrer les coefficients de l'équation:1 2 1
L'équation 1 x^2 + 2 x + 1 =0 possède une solution réelle.
In [4]:
```

1.8 Modules

Les fonctions intégrées au langage sont les plus utilisées comme print() et input(), les autres sont regroupées dans des fichiers séparés que l'on appelle des modules. Ces modules couvrent des domaines très divers: mathématiques, administration système, programmation réseau, manipulation de fichiers. Vous pouvez accéder à la documentation sur: https://docs.python.org/fr/3/py-modindex.html. Vous pouvez trouver d'autres modules chez divers fournisseurs. Nous en présenterons ici seulement ceux dont nous avons besoin dans la suite de notre ouvrage.

1.8.1 Importation de modules

Pour avoir accès aux fonctions d'un module existant, il faut importer le module au début du programme via l'instruction **import**, ce qui peut se faire de différentes manières:

import module1, module2,...: l'instruction import donne accès à toutes les fonctions des modules.

```
1 import math
2 print("pi=", math.pi)
3 print("exp(2)=", math.exp(2))
4 print("sin(pi/2)=", math.sin(math.pi/2))
5
```

```
In [1]:
pi= 3.141592653589793
exp(2)= 7.38905609893065
sin(pi/2)= 1.0
In [2]:
```

import module **as** alias: permet d'utiliser un alias plus explicite ou plus court.

```
1 import math as m
2 print("pi=", m.pi)
3 print("exp(2)=", m.exp(2))
4
```

```
In [1]:
pi= 3.141592653589793
exp(2)= 7.38905609893065
In [2]:
```

from module **import** fonction1, fonction2,...: importer une ou plusieurs fonctions d'un module.

```
1 from  math import pi,exp
2 print("pi=", pi)
3 print("exp(2)=", exp(2))
4
```

```
In [1]:
pi= 3.141592653589793
exp(2)= 7.38905609893065
In [2]:
```

from module **import** *: On peut également importer toutes les fonctions d'un module.

```
1 from math import *
2 print("pi=", pi)
3 print("exp(2)=", exp(2))
4 print("sin(pi/2)=", sin(math.pi/2))
5
```

```
In [1]:
pi= 3.141592653589793
exp(2)= 7.38905609893065
sin(pi/2)= 1.0
In [2]:
```

O.henaoui

1.8.2 Bibliothèque standard

Python offre plus de deux cents modules. Nous en présenterons ici seulement quatres dont nous avons besoin.

1) Module math:

math.e: la valeur de exp(1).	**math.exp(x)**: exp(x).
math.pi: la valeur approimative de pi.	**math.log(x)**: ln(x).
math.sqrt(x): la valeur de la racine carrée de x.	**math.log10(x)**: le log base 10 de x.
math.pow(x,n) : x puissance n.	**math.cos(x)**: cos(x), x en radians.
math.fmod(x,y) : x modulo y avec x et y réels.	**math.sin(x)**: sin(x), x en radians.
math.gcd(a,b): le plus grand diviseur commun de a et b.	**math.tan(x)**: tan(x), x en radians.
math.floor(x): E(x) la partie entière du réel x.	**math.acos(x)**: arccos(x) en radians.
math.ceil(x): E(x)+1.	**math.asin(x)**: arcsin(x) en radians.
math.fabs(x): la valeur absolue de x.	**math.atan(x)**: arctan(x) en radians.
math.factorial(x): le factoriel x!.	**math.cosh(x)**: cosh(x).
math.radians(x): convertit x en radians.	**math.sinh(x)**: sinh(x).
math.degrees(x): convertit x en degrés.	**math.tanh(x)**: tanh(x).

math.isinf(x): teste si x est infini.

math.isnan(x): teste si x est nan (Not a Number).

math.fsum(L): la somme des éléments d'une liste L.

2) Module cmath: reprend la plupart des fonctions du module math pour les nombres complexes.

3) Module decimal:

Les flottants sont mémorisés avec une précision fixée (15 chiffres significatifs). Cependant contrairement aux entiers où les calculs sont toujours exacts, les flottants posent un problème de précision et les erreurs d'arrondi s'accumulent à chaque étape du calcul. ce problème n'est pas spécifique à Python, il existe pour tous les langages, mais est dû à la technique de codage des nombres flottants sous forme binaire. En effet, un nombre rationnel peut tout à fait posséder un développement décimal fini et un développement binaire illimité! Par exemple:

base 10	base 2
1.1	01.000110011001100110011...
2.2	10.001100110011001100110...

O.henaoui

```
1 x=1.1+2.2
2 print(x==3.3)
3 print("x=",x)
4
```

```
In [1]:
False
x= 3.3000000000000003
In [2]:
```

Le module decimal permet d'effectuer des calculs exacts sur les nombres décimaux, dans les limites d'une précision fixée par l'utilisateur (28 chiffres significatifs par défaut).
L'instruction **getcontext().prec = n** fixe la précision à n chiffres significatifs.

```
1 from decimal import Decimal
2 x=Decimal('1.1')+Decimal('2.2')
3 print(x==Decimal('3.3'))
4 print("x=",x)
5
```

```
In [1]:
True
x= 3.3
In [2]:
```

Un certain nombre des fonctionnalités du module decimal sont des ajouts récents. Vous pouvez accéder à la documentation Python sur: `https://docs.python.org/fr/3/library/decimal.html` qui montrent comment programmer le calcul de π, exp(x), cos(x), sin(x) avec une précision donnée.

```
1 import decimal as d
2 import math as m
3 d.getcontext().prec=50   # précision
4 print("1/7=")
5 print(1/7)
6 print(d.Decimal(1)/d.Decimal(7))
7 print("sqrt(2)=")
8 print(m.sqrt(2))
9 print(d.Decimal(2).sqrt())
10 print("exp(1)=")
11 print(m.exp(1))
12 print(d.Decimal(1).exp())
13
```

```
In [1]:
1/7=
0.14285714285714285
0.14285714285714285714285714285714285714285714285714
sqrt(2)=
1.4142135623730951
1.4142135623730950488016887242096980785696718753769
exp(1)=
2.718281828459045
2.7182818284590452353602874713526624977572470937000
In [2]:
```

4) Module fractions: permet d'effectuer des calculs exacts sur les nombres rationnels.

```
1 import fractions as f
2 f1= f.Fraction(3,12)
3 f2= f.Fraction(1,5)
4 print('{} + {} = {}'.format(f1,f2,f1+f2))
5 print('{} - {} = {}'.format(f1,f2,f1-f2))
6 print('{} * {} = {}'.format(f1,f2,f1*f2))
7 print('{} / {} = {}'.format(f1,f2,f1/f2))
8 print('{} ^ {}   = {}'.format(f1,2,f1**2))
9 print('{} ^ {} = {}'.format(f1,f2,f1**f2))
10
```

```
In [1]:
1/4 + 1/5 = 9/20
1/4 - 1/5 = 1/20
1/4 * 1/5 = 1/20
1/4 / 1/5 = 5/4
1/4 ^ 2   = 1/16
1/4 ^ 1/5 = 0.757858283255199
In [2]:
```

O.henaoui

5) Module random: regroupe plusieurs fonctions permettant de travailler avec des valeurs aléatoires. La distribution des nombres aléatoires est réalisée par le générateur de nombres pseudo-aléatoires Mersenne Twister, l'un des générateurs les plus testés et utilisés dans le monde informatique.

random.random(): renvoie un décimal aléatoire dans [0,1[.

random.randrange(m,n,h): choisit un entier aléatoirement dans range (m,n, h).

random.randint(a,b): choisit un entier aléatoirement dans $[a, b]$.

random.uniform(a,b): choisit un décimal aléatoirement dans $[a, b]$.

random.choice(L): choisit un entier aléatoirement dans la liste L.

Le module random permet bien plus d'utilisations que la simple attribution d'une valeur dans un intervalle donné (voir `https://docs.python.org/fr/3/library/random.html`)

```
1 import random
2 print(random.random())
3 print(random.randrange(2,10,2))
4 print(random.randint(2,5))
5 print(random.uniform(2,5))
6 print(random.choice([1500,"ab",1.25,"py",1.25]))
7
```

```
In [1]:
0.08956050548917238
2
3
3.5847784199363733
1.25
In [2]:
```

6) Module timeit: permet de mesurer le temps d'exécution d'une fonction afin de le comparer avec le temps d'exécution d'une autre.

```
1 import timeit
2 f1=lambda x: 20*x**5 - 3*x**4 + 5*x**3 - 2*x**2 + 9*x + 7
3 f2=lambda x: (((((20*x) - 3)*x + 5)*x - 2)*x + 9)*x + 7
4 dureef1 = timeit.Timer("f1(99)","from __main__ import f1")
5 print("duréef1=",dureef1.timeit(10))# 10 appels à la fonction
6 dureef2 = timeit.Timer("f2(99)","from __main__ import f2")
7 print("duréef2=",dureef1.timeit(10))
8
```

```
In [1]:
duréef1= 3.560000000035757e-05
duréef2= 3.289999999900317e-05
In [2]:
```

1.8.3 Bibliothèque tierces

Il existe de nombreux modules externes qui ne sont pas installés de base dans Python mais qui sont très utilisés. Le site `https://pypi.org` (The Python Package Index) recense des milliers de modules.

1) Module numpy est une bibliothèque de fonctions indispensables pour faire du calcul scientifique. Il permet d'effectuer des calculs sur des vecteurs ou des matrices, élément par élément, via un nouveau type appelé **array** (tableau) au lieu de celui de liste. Il existe un autre module **scipy** complémentaire de numpy pour le traitement de données (statistiques, optimisation, traitement du signal...)

Création de tableaux

np.array()

```
1 import numpy as np
2 X = np.array([1, 2, 3])
3 print(X)
4 print(type(X))      # type de X: tableau
5 print(X.dtype)      # Type des données du tableau
6 print(X.shape)      # Forme du tableau
7 A = np.array([[6, 1], [3, 2], [1.2, 7]])
8 print(A)
9 print(type(A))      # type de A: tableau
10 print(A.dtype)      # Type des données du tableau
11 print(A.shape)      # Forme du tableau
12
```

```
In [1]:
[1 2 3]
<class 'numpy.ndarray'>
int32
(3,)
[[6.  1. ]
 [3.  2. ]
 [1.2 7. ]]
<class 'numpy.ndarray'>
float64
(3, 2)
In [2]:
```

np.arange

```
1 import numpy as np
2 X = np.arange(10)       # debut=0,fin=9
3 print(X)
4 Y = np.arange(1,10,2)  # debut=1, fin=9, pas=2
5 print(Y)
6 Z = np.arange(0,1,0.2)  # debut=0, fin=0.8, pas=0.2
7 print(Z)
8
```

```
In [1]:
[0 1 2 3 4 5 6 7 8 9]
[1 3 5 7 9]
[0.  0.2 0.4 0.6 0.8]
In [2]:
```

np.linspace

```
1 import numpy as np
2 X = np.linspace(1,9,5) # debut=1, fin=9, données=5
3 print(X)
4 Y = np.linspace(1,9,4,endpoint=False)
5 print(Y)
6
```

```
In [1]:
[1. 3. 5. 7. 9.]
[1. 3. 5. 7.]
In [2]:
```

np.random

```
1 import numpy as np
2 X = np.random.rand(4)
3 print(X)
4
```

```
In [1]:
[0.41184996 0.28200466 0.2928583  0.21503636]
In [2]:
```

Tableaux constants

```
1 import numpy as np
2 A = np.zeros((2,3))
3 print(A)
4 B = np.ones((2,3))
5 print(B)
6 C = np.identity(3)
7 print(C)
8
```

```
In [1]:
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
In [2]:
```

O.henaoui

Changement de format

```
1 import numpy as np
2 A = np.array([1, 2, 3, 4, 5, 6])
3 A.shape=(2,3) # A = np.array([1, 2, 3, 4, 5, 6]).reshape(2,3)
4 print(A)
5 B = np.ones((4,4)).reshape(2,8)
6 print(B)
7 C = np.arange(6).reshape(3,2)
8 print(C)
9 E = C.transpose() # transposée de C
10 print(E)
11
```

```
In [1]:
[[1 2 3]
 [4 5 6]]
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
[[0 1]
 [2 3]
 [4 5]]
[[0 2 4]
 [1 3 5]]
In [2]:
```

Décaler la diagonale

```
1 import numpy as np
2 A1 = np.eye(3)
3 print(A1)
4 A2 = np.eye(3,k=-1)
5 print(A2)
6 A3 = np.eye(3,k=1)
7 print(A3)
8 B1 = np.diag([1,2,3])
9 # B1 = np.diag(np.array([1,2,3]))
10 # B1 = np.diag(np.arange(2))
11 print(B1)
12 B2 = np.diag([1,2,3],k=-2)
13 print(B2)
14 B3 = np.diag([1,2,3],k=1)
15 print(B3)
16
```

```
In [1]:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]]
[[0 1 0 0]
 [0 0 2 0]
 [0 0 0 3]
 [0 0 0 0]]
In [2]:
```

Changer le type des données

```
1 import numpy as np
2 X = np.array([1, 2, 3, 4, 5, 6],dtype = "float")
3 print(X)
4 print(X.dtype)
5 Y = np.arange(10,dtype = "float")
6 print(Y)
7 print(Y.dtype)
8 A = np.ones((3,3),dtype = "int")
9 print(A)
10 print(A.dtype)
11 B = A.astype(float)
12 print(B)
13 print(A.dtype)
14
```

```
In [1]:
[1. 2. 3. 4. 5. 6.]
float64
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
float64
[[1 1 1]
 [1 1 1]
 [1 1 1]]
int32
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
int32
In [2]:
```

Tableau à partir d'une fonction

```
1 import numpy as np
2 Y = np.fromfunction(lambda i: i**2, (10,))
3 print(Y)
4
```

```
In [1]:
[ 0.  1.  4.  9. 16. 25. 36. 49. 64. 81.]
In [2]:
```

Le module numpy dispose d'un grand nombre de fonctions mathématiques qui peuvent être appliquées directement à un tableau. Exemple: np.sin(A), np.cos(A), np.exp(A), np.log(A), np.pi, np.abs(A)...

Accès aux tableaux L'accès aux éléments d'un tableau se fait de la même manière que pour les listes.

```
X = [  5     6     7     8     9    10    11    12    13  ]
X = [ X[0]  X[1]  X[2]  X[3]  X[4]  X[5]  X[6]  X[7]  X[8] ]
```

```
1 import numpy as np
2 X = np.arange(5,14)
3 print("X = ",X)
4 print("X[0]=",X[0])
5 print("X[7]=",X[7])
6 print("X[-1]=",X[-1])
7 print("X[0:5]=",X[0:5])
8
```

```
In [1]:
X =  [ 5  6  7  8  9 10 11 12 13]
X[0]= 5
X[7]= 12
X[-1]= 13
X[0:5]= [5 6 7 8 9]
In [2]:
```

```
A = [[5   6   7 ]     A = [[A[0,0]  A[0,1]  A[0,2]]  ← A[0]
     [8   9   10]          [A[1,3]  A[1,4]  A[1,5]]  ← A[1]
     [11  12  13]]         [A[2,6]  A[2,7]  A[2,8]]] ← A[2]
```

```
1 import numpy as np
2 A = np.arange(5,14).reshape(3,3)
3 print("A = \n",A)
4 # élément de 1ère ligne 2ème colonne
5 print("A[0,1]=",A[0,1],"\t A[0][1]=",A[0][1])
6 # élément de dernière ligne avant dernière colonne
7 print("A[-1,-1]=",A[-1,-2],"\t A[-1][-1]=",A[-1][-2])
8 print("A[1]=",A[1])            # la 2ème ligne
9 print("A[-1]=",A[-1])           # dernière ligne
10 print("A[1:3]= \n",A[1:3])       # de la ligne 1 à 3
11 print("A[[0,2]]= \n",A[[0,2]])    # ligne 1 et ligne 3
12 print("A[:, [0,1]]= \n",A[:, [0,1]]) # colonne 1 et 2
13 print("A[1,[0,1]]= \n",A[1,[0,1]]) # ligne 2,colonne 1 et 2
14
```

```
In [1]:
A =
 [[ 5  6  7]
 [ 8  9 10]
 [11 12 13]]
A[0,1]= 6       A[0][1]= 6
A[-1,-1]= 12    A[-1][-1]= 12
A[1]= [ 8  9 10]
A[-1]= [11 12 13]
A[1:3]=
 [[ 8  9 10]
 [11 12 13]]
A[[0,2]]=
 [[ 5  6  7]
 [11 12 13]]
A[:, [0,1]]=
 [[ 5  6]
 [ 8  9]
 [11 12]]
A[1,[0,1]]=
 [8 9]
In [2]:
```

O.henaoui

Modifier un tableau

```
1 import numpy as np
2 A = np.array([[1, 1, 1, 1], [2, 2, 2, 2]])
3 A[1] = [3, 4, 5, 6]
4 print("A = \n",A)
5 A[0,1] = 0
6 print("A = \n",A)
7 A = A.astype(float) # attention changer le type de A.
8 A[1,3] = 0.2         # pour remplacer un float
9 print("A = \n",A)
10
```

```
In [1]:
A =
 [[1 1 1 1]
 [3 4 5 6]]
A =
 [[1 0 1 1]
 [3 4 5 6]]
A =
 [[1.  0.  1.  1. ]
 [3.  4.  5.  0.2]]
In [2]:
```

```
1 import numpy as np
2 X = np.arange(5,14)
3 print("X = ",X)
4 X = np.append(X,14) # ajouter l'élément 14
5 print("X = ",X)
6 X = np.delete(X,(-1)) # suprimer le dernier élément
7 print("X = ",X)
8 X = np.delete(X, ([1,5])) # suprimer X[1] et X[5]
9 print("X = ",X)
10
```

```
In [1]:
X = [ 5  6  7  8  9 10 11 12 13]
X = [ 5  6  7  8  9 10 11 12 13 14]
X = [ 5  6  7  8  9 10 11 12 13]
X = [ 5  7  8  9 11 12 13]
In [2]:
```

```
1 import numpy as np
2 A = np.arange(5,14).reshape(3,3)
3 print("A = \n",A)
4 X = np.array([[1], [0], [1]])
5 A = np.append(A,X, axis = 1) # ajouter la colonne X à A
6 print("A = \n",A)
7 W = np.array([14, 15, 16, 17])
8 A = np.vstack([A,W])  # ajouter une ligne W
9 print("A = \n",A)
10 A = np.delete(A, (0), axis=1) # suprimer la 1ère colonne
11 # A = np.delete(A,([0,2]),axis=1) suprimer 1ère, 3ème colonne
12 print("A = \n",A)
13 A = np.delete(A, (1), axis=0) # suprimer la 2ème ligne
14 #A = np.delete(A,([0,2]),axis=0) suprimer 1ère, 3ème ligne
15 print("A = \n",A)
16
```

```
In [1]:
A =
 [[ 5  6  7]
 [ 8  9 10]
 [11 12 13]]
A =
 [[ 5  6  7  1]
 [ 8  9 10  0]
 [11 12 13  1]]
A =
 [[ 5  6  7  1]
 [ 8  9 10  0]
 [11 12 13  1]
 [14 15 16 17]]
A =
 [[ 6  7  1]
 [ 9 10  0]
 [12 13  1]
 [15 16 17]]
A =
 [[ 6  7  1]
 [12 13  1]
 [15 16 17]]
In [2]:
```

Copier un tableau

```python
import numpy as np
A = np.array([[1, 2, 3],[4,5,6]])
B = A   # créer une référence et non pas une copie.
C = np.array(A)  # 1ère méthode pour copier
D = A.copy()     # 2ème méthode pour copier
A[0,1]=0
print("A =\n",A)
print("B =\n",B)
print("C =\n",C)
print("D =\n",D)
```

```
In [1]:
A =
 [[1 0 3]
 [4 5 6]]
B =
 [[1 0 3]
 [4 5 6]]
C =
 [[1 2 3]
 [4 5 6]]
D =
 [[1 2 3]
 [4 5 6]]
In [2]:
```

Opérations sur les tableaux

```python
import numpy as np
A = np.array([[1, 2, 3],[4, 5, 6]])
B = np.array([[7, 8, 9],[10, 11, 12]])
print("A =\n",A)
print("B =\n",B)
print("A + B =\n",A+B) # addition terme à terme
print("A - B =\n",A-B) # soustraction terme à terme
print("A + 2 =\n",A+2) # ajoutr 2 à tous les éléments
print("A - 2 =\n",A-2) # soustraire 2 à tous les éléments
print("A * 2 =\n",A*2) # multiplier tous les éléments par 2
print("A / 2 =\n",A/2) # diviser tous les éléments par 2
print("A **2 =\n",A**2)# élever chaque élément à la puissance 2
```

```
In [1]:
A =
 [[1 2 3]
 [4 5 6]]
B =
 [[ 7  8  9]
 [10 11 12]]
A + B =
 [[ 8 10 12]
 [14 16 18]]
A - B =
 [[-6 -6 -6]
 [-6 -6 -6]]
A + 2 =
 [[3 4 5]
 [6 7 8]]
A - 2 =
 [[-1  0  1]
 [ 2  3  4]]
A * 2 =
 [[ 2  4  6]
 [ 8 10 12]]
A / 2 =
 [[0.5 1.  1.5]
 [2.  2.5 3. ]]
A ** 2 =
 [[ 1  4  9]
 [16 25 36]]
In [2]:
```

np.dot pour effectuer un produit matriciel.

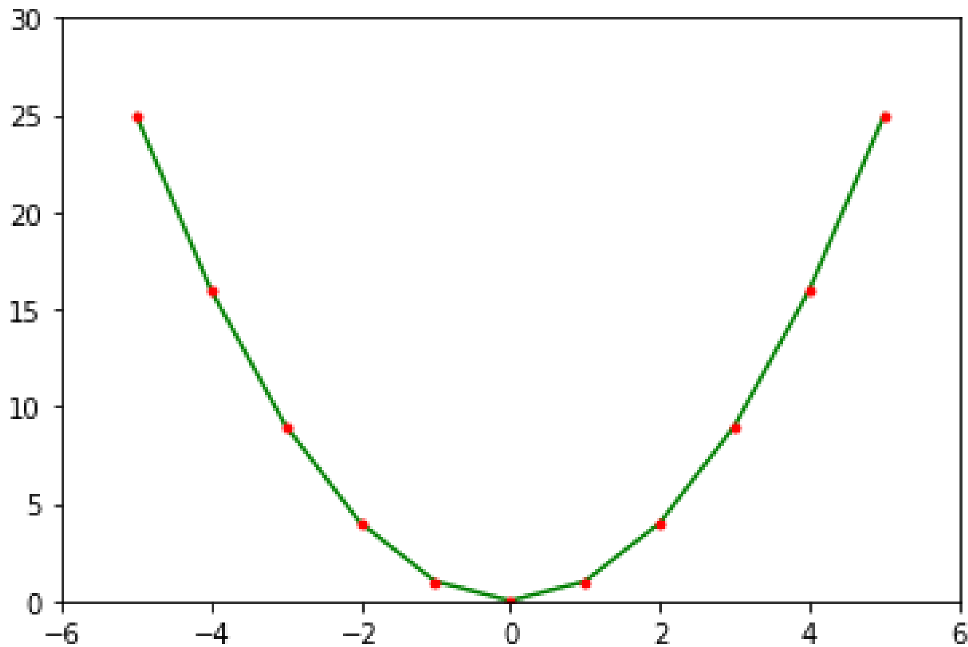
```python
import numpy as np
A = np.array([[1, 2, 3],[4, 5, 6]])
B = np.array([[7, 8, 9],[10, 11, 12]])
X = np.array([[1], [2], [3]])
Y = np.array([[7], [8], [9]])
Bt = B.transpose()  # transposée de B
print("A =\n",A)
print("Bt =\n",Bt)
print("A*Bt =\n",np.dot(A,Bt)) # produit matriciel
print("A*Bt =\n",A @ Bt)
print("X =\n",X)
print("Y =\n",Y)
print("Xt*Y =\n",np.dot(X.transpose(),Y))
print("X*Y =\n",np.vdot(X,Y)) # produit scalaire de X et Y
```

```
In [1]:
A =
 [[1 2 3]
 [4 5 6]]
Bt =
 [[ 7 10]
 [ 8 11]
 [ 9 12]]
A*Bt =
 [[ 50  68]
 [122 167]]
A*Bt =
 [[ 50  68]
 [122 167]]
X =
 [[1]
 [2]
 [3]]
Y =
 [[7]
 [8]
 [9]]
Xt*Y =
 [[50]]
X*Y =
 50
In [2]:
```

2) Module matplotlib permet de faire des représentations graphiques très variées. Pour illustrer les fonctionnalités de matplotlib, nous allons avoir besoin du module numpy.

Tracer un graphe Pour tracer un graphe sur un repère cartésien, on utilise la fonction plot().

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 x = np.array([-5,-4,-3,-2,-1,0,1,2,3,4,5])
4 y = np.array([25,16,9,4,1,0,1,4,9,16,25])
5 plt.plot(x,y,"g-")     # des lignes vertes qui relient les points(x,y)
6 plt.plot(x,y,"r.")     # ou bien plt.plot(x,y,"g-",x,y,"r.")
7 plt.axis([-6,6,0,30]) # axe des x: [-6,6], axe des y:[0,30]
8 plt.show()
9
```



Type de couleurs: pour changer la couleur on utilise **color = "orange"** ou bien

"b"	blue	"m"	magenta	"r"	red	"k"	black
"g"	green	"y"	yellow	"c"	cyan	"w"	white

Type de lignes: pour changer le type des lignes on utilise **linestyle = "-"** (style des lignes)

"-"	ligne pleine	"-."	points et tirets	"–"	tirets	":"	pointillés

linewidth = 0.1 (épaisseur des lignes)

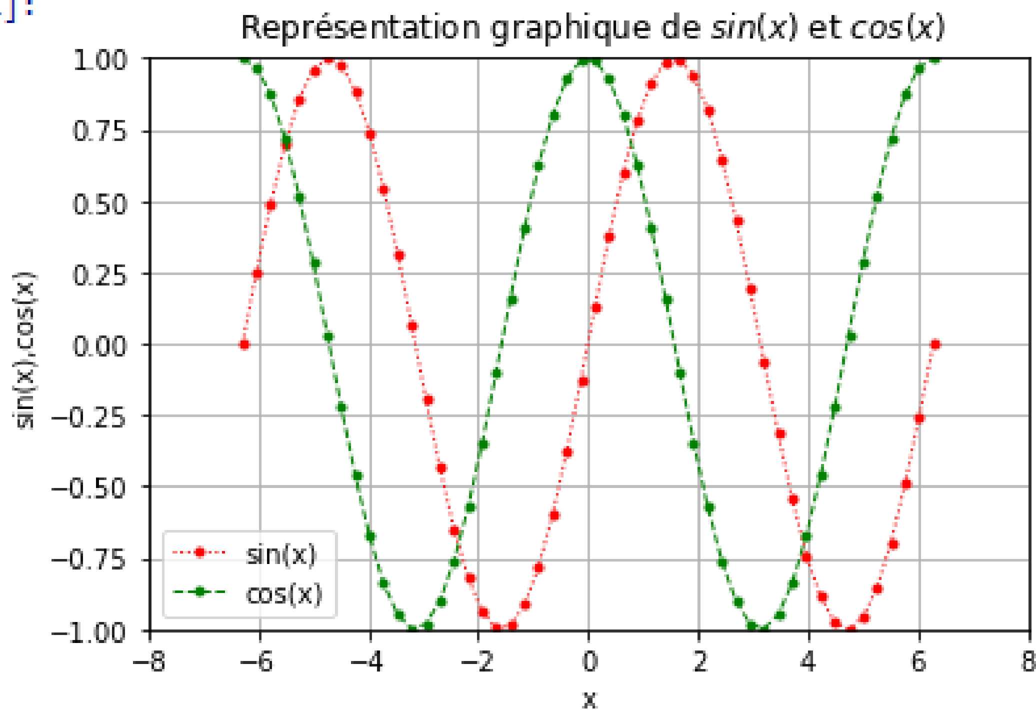
marker="." (marqueurs à chaque point); **markerfacecolor="r"** (couleur); **markersize=1** (taille).

O.henaoui

<table>
<tr><td>.</td><td>points</td><td>v</td><td>triangles vers le bas</td><td>1</td><td>tri-down</td></tr>
<tr><td>,</td><td>pixels</td><td>o</td><td>cercles vides</td><td>2</td><td>tri-up</td></tr>
<tr><td>^</td><td>triangles vers le haut</td><td>s</td><td>carré</td><td>3</td><td>tri-left</td></tr>
<tr><td>+</td><td>symbole plus</td><td>p</td><td>pentagone</td><td>4</td><td>tri-right</td></tr>
<tr><td>*</td><td>astérisque</td><td>h</td><td>hexagone</td><td><</td><td>triangles vers la gauche</td></tr>
<tr><td>|</td><td>ligne verticale</td><td>x</td><td>symbole multiplier</td><td>></td><td>triangles vers la droite</td></tr>
<tr><td>_</td><td>ligne horizontale</td><td>d</td><td>losange</td><td></td><td></td></tr>
</table>

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  x = np.linspace(-2*np.pi,2*np.pi,50)
4  plt.plot(x,np.sin(x),"r",linewidth=1,linestyle=":",marker=".",label="sin(x)")
5  plt.plot(x,np.cos(x),"g",linewidth=1,linestyle="--",marker=".",label="cos(x)")
6  plt.axis([-8,8,-1, 1])  # axe des x: [-8,8], axe des y:[-1,1]
7  plt.title("Représentation graphique de $sin(x)$ et $cos(x)$")
8  plt.xlabel('x')
9  plt.ylabel('sin(x),cos(x)')
10 plt.legend()
11 plt.grid(True)
12 plt.show()
13 
```

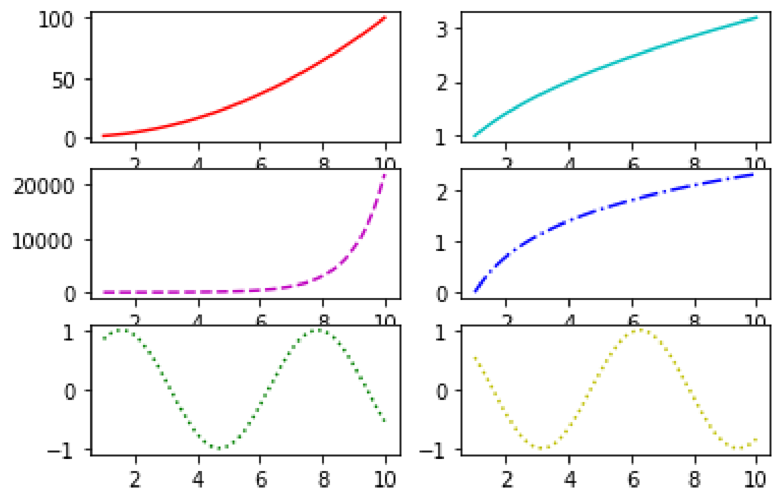
In [1]:



O.henaoui

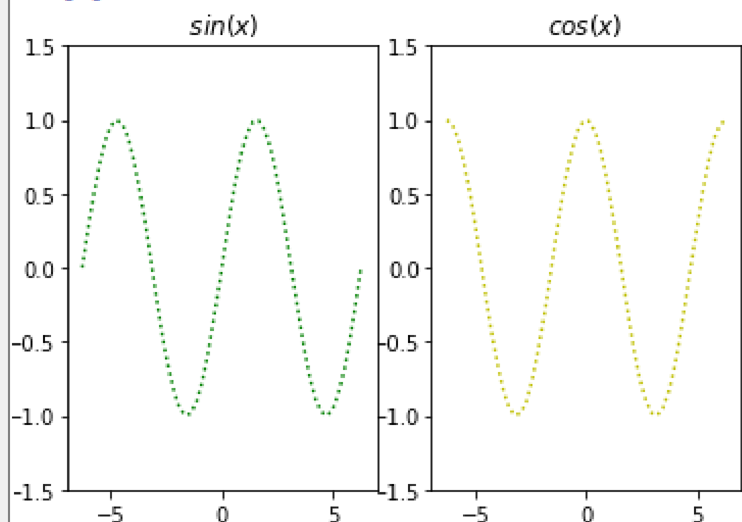
Plusieurs graphes sur une figure: On utilise la fonction subplot() pour placer des graphes, comme dans une matrice, les uns à côté des autres.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(1,10,50)
# Ligne 1,colonne 1
plt.subplot(3,2,1)
plt.plot(x, x**2,"r-")
# Ligne 1,colonne 2
plt.subplot(3,2,2)
plt.plot(x, np.sqrt(x),"c-")
# Ligne 2,colonne 1
plt.subplot(3,2,3)
plt.plot(x, np.exp(x),"m--")
# Ligne 2,colonne 2
plt.subplot(3,2,4)
plt.plot(x, np.log(x),"b-.")
# Ligne 3,colonne 1
plt.subplot(3,2,5)
plt.plot(x, np.sin(x),"g:")
# Ligne 3,colonne 2
plt.subplot(3,2,6)
plt.plot(x, np.cos(x),"y:")
plt.show()
```



Il est souvent utile d'avoir plusieurs figures à côté l'une de l'autre pour les comparer. Il est très utile pour étudier un graphe d'avoir un titre, de modifier les limites de l'axe des abscisses et de l'axe des ordonnées...

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-2*np.pi,2*np.pi,50)
# Ligne 1,colonne 1
plt.subplot(1,2,1)
plt.plot(x, np.sin(x),"g:")
plt.axis([-7,7,-1.5, 1.5])
plt.title(" $sin(x)$ ")
# Ligne 1,colonne 2
plt.subplot(1,2,2)
plt.plot(x, np.cos(x),"y:")
plt.axis([-7,7,-1.5, 1.5])
plt.title("$cos(x)$")
plt.show()
```

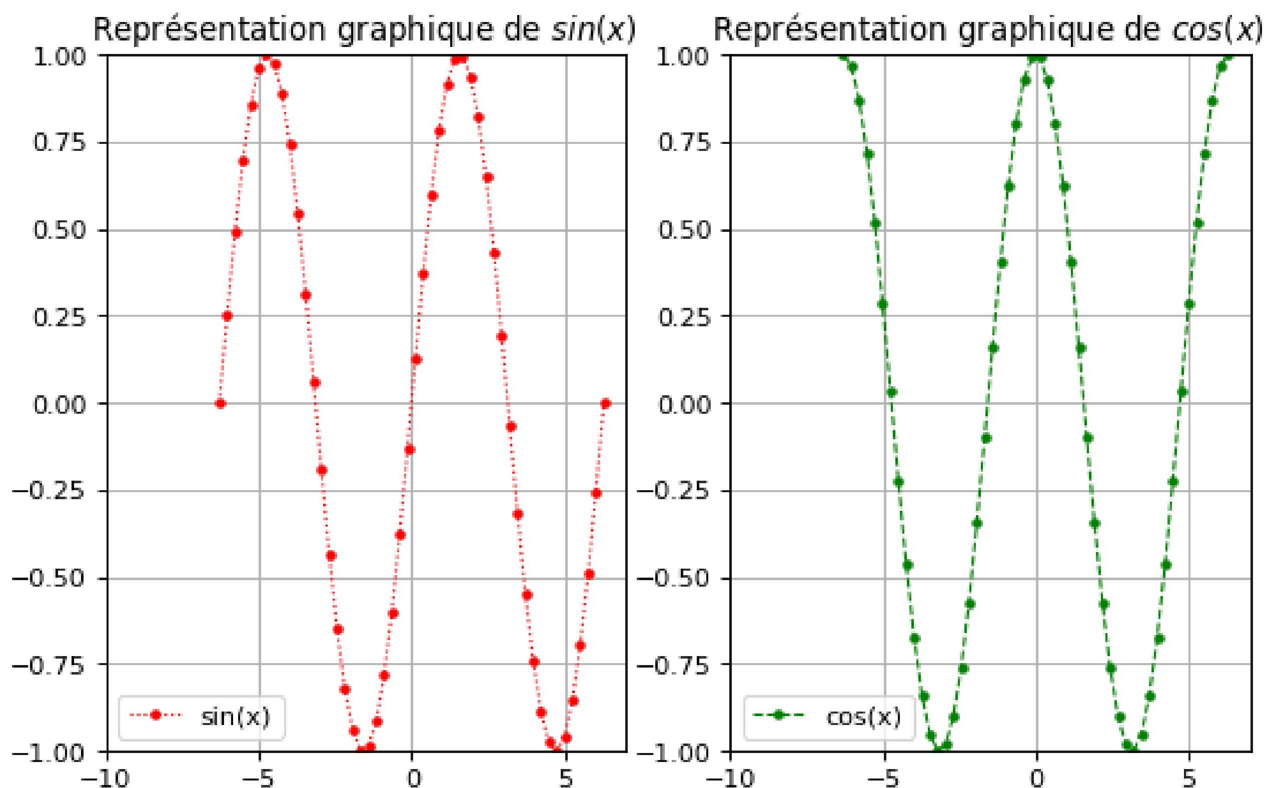


O.henaoui

Personnalisation des graphes

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 # un graphe de 8x5 pouces avec une résolution de 80 points par pouce
4 plt.figure(figsize=(8,5), dpi=80)
5 x = np.linspace(-2*np.pi,2*np.pi,50)
6 # Ligne 1,colonne 1
7 plt.subplot(1,2,1)
8 plt.plot(x,np.sin(x),"r",linewidth=1,linestyle=":",marker=".",label="sin(x)")
9 plt.axis([-10,7,-1, 1])
10 plt.title("Représentation graphique de $sin(x)$ ")
11 plt.legend()
12 plt.grid(True)
13 # Ligne 1,colonne 2
14 plt.subplot(1,2,2)
15 plt.plot(x,np.cos(x),"g",linewidth=1,linestyle="--",marker=".",label="cos(x)")
16 plt.axis([-10,7,-1, 1])
17 plt.title("Représentation graphique de $cos(x)$")
18 plt.legend()
19 plt.grid(True)
20 plt.show()
21
```

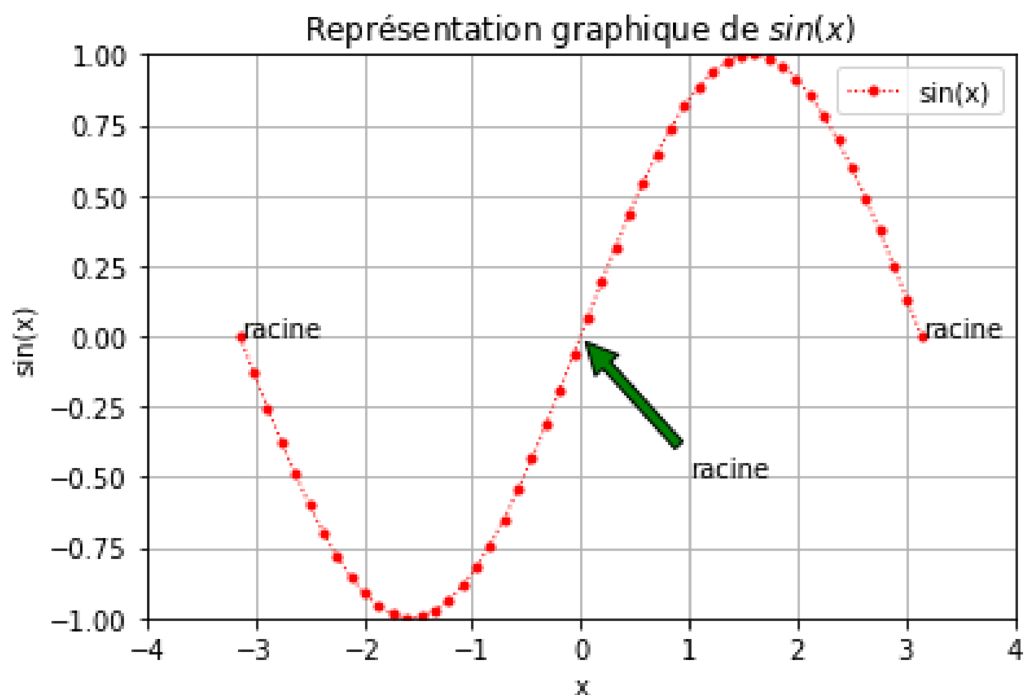
In [1]:



O.henaoui

Ecrire dans un graphe on utilise les commandes text() et annotate() pour afficher du texte et une flèche d'indication.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 plt.grid(True)
4 x = np.linspace(-np.pi,np.pi,50)
5 plt.plot(x,np.sin(x),"r",linewidth=1,linestyle=":",marker=".",label="sin(x)")
6 plt.text(np.pi, 0, r'racine') # écrire "racine dans le graphe"
7 plt.text(-np.pi, 0, r'racine')
8 # Ajouter une flèche descriptive écrire "racine
9 plt.annotate('racine',xy=(0,0),xytext=(1,-0.5),
10              arrowprops=dict(facecolor='green', shrink=0.05))
11 plt.axis([-4,4,-1, 1])
12 plt.title("Représentation graphique de $sin(x)$")
13 plt.xlabel('x')
14 plt.ylabel('sin(x)')
15 plt.legend()
16 plt.show()
17
```



Vous pouvez faire varier ces réglages pour voir ce que cela donne (voir la documentation matplotlib `https://matplotlib.org/tutorials/index.html`)

3) Module personnel Il est possible d'alimenter un module personel avec toutes les fonctions réalisées durant l'année.

O.henaoui