Chapitre 1

Introduction à Python/Sage et Xcas/Giac

1.1 Ce qu'il faut savoir lorsque l'on utilise un logiciel

Nous expliquerons ici les questions qu'il faut se poser, et comment y répondre rapidement, avec la documentation ou des essais.

- Types élémentaires. Représentations des entiers, flottants, notion de multiprécision, symboles, fonctions.
- Variables, affectations(valeur/adresse), listes et indices
- Programmation : tests et boucles, utilisation de fonctions.
- Variables locales/globales. Argument des fonctions et affectations.

1.2 Présentation, Installation et documentation

Xcas est l'interface graphique du logiciel et de la librairie giac (cf `http://www-fourier.ujf-grenoble.fr/~parisse/giac.html`). Sa documentation interne est très importante. En plus de la documentation html disponible hors ligne on peut aussi utiliser : menus par thèmes, complétion par tabulation, bulles d'aide pour la saisie des arguments d'une fonction [1] ou recherche par mot clef [2].

1.3 Types et affectations

En python et dans xcas il existe de nombreux types différents. En revanche il n'y a pas obligatoirement de déclaration de types ni de définition des variables. C'est au moment de l'affectation qu'une variable obtient un type (typage dynamique). Les variables déclarées dans une fonction ont une portée locale.

1.3.1 Types élémentaires

Les entiers, les entiers de taille arbitraire , les flottants, les booléens, les chaines de caractères.

(python/sage)

```
>>> a=2**4  # la variable a recoit la valeur 2^4
>>> a
16
>>> 2**64    # 2^64
18446744073709551616
>>> 0.3**64
```

1. Ex : on tape : *int(* puis on attend sans bouger la souris que la bulle apparaisse
2. Ex : bien que *pgcd* ne soit pas un nom de fonction xcas, taper ?*pgcd* permet de trouver la bonne instruction.

```
3.4336838202925044e-34
>>> 5==5    # On fait apparaitre le booleen vrai en comparant 5 et 5
True
>>> a="debut du texte" # on definit une chaine de caractere
>>> b=' et la suite'   # on a le choix pour delimiter la chaine entre " ou '
>>> a+b # on juxtapose les deux chaines a et b
'debut du texte et la suite'
>>> a="Dans une chaine delimitee par \" il est inutile de proteger les ' "
>>> a
'Dans une chaine delimitee par " il est inutile de proteger les \' '
>>> print(a)
Dans une chaine delimitee par " il est inutile de proteger les '
```

Sous xcas, l'affectation se fait avec `:=` alors que pour python c'est `=`. En revanche de nombreux opérateurs valables en python font parties des syntaxes possibles sous xcas.

(giac/xcas)

```
0>>  a:=2**64          // Dans xcas l'affectation est differente, c'est :=
18446744073709551616
1>>  2^64              //on peut aussi utiliser 2^64 pour 2^64
18446744073709551616
2>>  5==5;   /* Il n'y a pas de type booleen, ce sont des entiers: 1 ou 0. */
  1
3>> c:="ne pas utiliser les ' pour delimiter les chaines dans xcas.";type(c);
DOM_STRING
4>> '2+2'; type('2+2') ;eval('2+2')//c'est reserve pour ne pas evaluer une
expression. cf forme inerte
2+2,DOM_SYMBOLIC,4
```

Exercice 1.3.1 *En python et dans xcas, utiliser la fonction* `type()` *sur les deux entiers ci dessus. Etudier dans les deux logiciels les résultats et le type des entrées suivantes :* `1/3`, `1.0/3`, `(1.0/3)**2000` *? Trouver (éventuellement dans l'aide) une fonction permettant de convertir un entier en un décimal. (Ex* `2` *en* `2.0`*) Dans quel cas les logiciels diffèrent ils ? Trouver dans xcas comment faire apparaitre une valeur approchée de* $1/3$ *avec 100 chiffres.*

	affectation	puissance	reste	quotient	commentaires
Python 2 et 3	=	**	2%3	4//3	#
giac ou Xcas	:=	^ ou **	irem(2,3)	iquo(4,3)	// ou /* */

NB : (4/3 ne donne pas la même chose en python2 et python3)

1.3.2 Séquences : T-uples, listes, ensembles, dictionnaires

On utilisera essentiellement les listes. On retiendra juste l'existence des ensembles et dictionnaires et comment s'informer rapidement sur ces notions dans la documentation en cas de besoin.

(python/sage)

```
>>> a=(11,22,"peu importe",44,11)  # un t-uple. Ses entrees ne sont pas modifiables.
>>> a[0]
11
>>> b=[11,22,"peu importe le type",44,11,range(3,8)]  # une liste.
>>> b[1]=222;b   #Les entrees d'une liste sont modifiables.
[11, 222, 'peu importe', 44, 11,[3,4,5,6,7]]
>>> b[0],b[3],b[5]
(11, 44, [3,4,5,6,7])
>>> b[5][0] #L'element d'indice 0 de la liste b[5]. En python b[5,0] est incorrect.
3
>>> c=set(a);c # on peut creer un ensemble a partir d'une liste ou d'un t-uple
set(['peu importe', 11, 44, 22])
>>> c[0] # Il n'y a pas d'indices pour les ensembles en python.
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> bureau={"han":"9D11","hermann":"9D23","danila":"9D3",112:"pas de bureau"}
>>> bureau["hermann"] #on a cree un dictionnaire. On accede aux elements
'9D23'
>>> bureau[112] #par des objets de type quelconque et non par des indices.
'pas de bureau'
```

Dans xcas il n'y a que des listes (modifiables). On peut utiliser () ou [], c'est sur la façon de traiter les emboitements que les choses diffèrent.

(giac/xcas)

```
9>> a:=(11,22,"texte",(1,2),11) // Ici les niveaux de () ne comptent pas.
11,22,"texte",1,2,11
10>> b:=[11,22,"texte",[1,2],11] // Les niveaux de []  comptent
[11,22,"texte",[1,2],11]
11>> a[0]:=111  // Dans xcas les objets sont modifiables
111,22,"texte",1,2,11
12>> b[0]:=111
[111,22,"texte",[1,2],11]
13>> b[3][0]; // ou bien b[3,0].
1
14>> set[1,3,2,2] // Dans xcas, les ensembles sont aussi notes: %{1,3,2,2%}
set[1,2,3]
15>> bureau:=table("han"="9D11","hermann"="9D23","danila"="9D3",112="pas de bureau");
//NB: Dans une table tous les types sont acceptes. Cf dictionnaire en python
table(
112 = "pas de bureau",
"danila" = "9D3",
"han" = "9D11",
"hermann" = "9D23"
)
16>> bureau["han"];bureau[112] //On accede aux elements par des objets
"9D11","pas de bureau"
```

1.3.3 Affectations : valeur ou adresse ?

Une première difficulté. Des questions à se poser avec chaque logiciel ou language.

(python)

```
a=1;b=a;b=2;print(a,b)
la=[11,22,33];lb=la;lb[0]=-7;print(la,lb)
```

(giac/xcas)

```
a:=1:;b:=a:;b:=2:; //Dans xcas pour ne pas afficher la reponse on utilise :;
print(a,b); // le ; sert a separer les instructions dans un programme
la:=[11,22,33]:;lb:=la:;lb[0]:=-7:;
afficher(la,lb)  // le ; est facultatif en mode interactif.
```

Remarque 1.3.2 *xcas possède une autre affectation que* `:=` *appelée affectation sur place dans la documentation. Elle se fait par le symbole* `=<`

(giac/xcas)

```
la=<[11,22,33]:;lb=<la:;lb[0]=<-7:;
afficher(la,lb)  // le ; est facultatif en mode interactif.
```

Exercice 1.3.3 *Essayez* `lb[0]:=7` *au lieu du* `=<`

1.4 Instructions de contrôle

⚠ En python il n'y a pas de délimiteurs de blocs, c'est l'indentation qui compte :

1.4.1 Tests

(python)

```
if (a<7):
   print("a inferieur a 7")
   print("a inferieur a 7")
else:
   print("a vaut au moins 7")
   print("a vaut au moins 7")
```

Xcas possède une syntaxe proche du C (Les boucles et tests coïncident à part l'affectation qui est `:=` sous xcas et `=` en C) :

(giac/xcas)

```
  if(a<7){
    print("a<7")
  }
  else {
    print("a>=7")
  }
```

On peut aussi utiliser un syntaxe de type algorithmique.

(giac/xcas)

```
  si (a<7) alors
    afficher("a<7")
  sinon
    print("a>=7")
  fsi
```

Les instructions de comparaison en python et xcas sont les mêmes :

```
x == y       # x est \'egal \`a y
x != y       # x est diff\'erent de y. NB <> est obsolete en python.
x > y        # x est plus grand que y
x < y        # x est plus petit que y
x >= y       # x est plus grand que, ou \'egal \`a y
x <= y       # x est plus petit que, ou \'egal \`a y
```

1.4.2 Boucles

(python/sage)

```
>>> a = 0
>>> while (a < 7):          # (n'oubliez pas le double point !)
...     a = a + 1           # (n'oubliez pas l'indentation !)
...     print(a)
```

(giac/xcas)

```
  a:=0;
  while(a<7) // faire shift entree pour aller a la ligne sans evaluer
    {
      a:=a+1; // ou bien a++ ou bien a+=1
      print(a);
```

```
  }
tantque a<17 faire
   a:=a+2;
   print(a);
ftantque;  //en fait xcas tolere l'absence de ; mais pas giac
```

En python il faut créer une liste pour pouvoir faire une boucle `for`.

(python/sage)

```
>>> range(10) # on peut utiliser range ou xrange pour creer une liste
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(3,10,2) # help(range) donne les options de range
[3, 5, 7, 9]
```

On calcule donc $\sum_{i=0}^{9} 3^i$ ainsi :

(python)

```
s=0
for i in range(10):
    s=s+3**i //le corps de la boucle est donne par l'indentation.
print(s)
```

Sous xcas[3] la lettre i est réservée au nombre complexe.

(giac/xcas)

```
s:=0;
for(j:=0;j<10;j++){  //pour aller de 2 en 2 on remplace j++ par j:=j+2 (ou j+=2)
  s:=s+3^j;
}
```

(giac/xcas)

```
s:=0;
pour j de 0 jusque 9 faire
 s:=s+3^j;
fpour
```

(giac/xcas)

```
s:=0;
pour j de 0 jusque 9 pas 2 faire // pour aller de 2 en 2
 s:=s+3^j;
fpour
```

Exercice 1.4.1 *En utilisant dans xcas la syntaxe* `for( ; ; )` *créez une bouble de type* `while`.

On peut aussi utiliser des listes avec xcas. L'analogue python de `range` est `seq`

(giac/xcas)

```
2>> s:=seq(j^2,j,1,10)  // seq donne une suite entre ( )
(1,4,9,16,25,36,49,64,81,100)
3>> seq(j^2,j=1..10)
1,4,9,16,25,36,49,64,81,100
4>> seq(j^2,j=1..10,2) // pour aller de 2 en 2 on ajoute un argument
1,9,25,49,81
5>>[s] // pour passer d'une suite entre ( ) a une liste entre [ ]
```

3. sauf en mode maple

(giac/xcas)

```
s:=0;
L:=seq(t,t,0,9);
for j in L do //ne marche pas en francais
  s:=s+3^j;
end;
```

1.5 Fonctions

1.5.1 Définition et type de variables

En python et xcas il n'y a pas de déclaration de type pour les arguments d'une fonction, ni pour les valeurs de retour. On peut retourner n'importe quel objet (même une liste)

(python)

```
def discri(a,b,c):
  """Cette fonction calcule le discriminant de ax^2+bx+c""" # aide facultative
  d=b**2-4*a*c #En python, variables locales par defaut. (Sinon utiser global)
  return d
```

(python/sage)

```
>>> discri(1,2,1)
0
>>> help(discri)
Help on function discri in module __main__:

discri(a, b, c)
    Cette fonction calcule le discriminant de ax^2+bx+c
```

(giac/xcas)

```
discri(a,b,c):={
  local d; //par defaut les variables sont globales mais un warning s'affiche
  d:=b^2-4*a*c;
  return(d);// ou bien: d;  le retour est la derniere evaluation
  }
```

(giac/xcas)

```
fonction discri(a,b,c)
  local d; //par defaut les variables sont globales mais un warning s'affiche
  d:=b^2-4*a*c;//ou bien b**2-4*a*c
  retourne(d);//ou bien: d; ou bien: retourne d;
ffonction
```

1.5.2 Arguments des fonctions et affectations en Python

(python)

```
def test1(b):
  b=b+1
  print("b dans la fonction",b)
  return b
```

(python/sage)

```
>>> a=1
>>> test1(a)
```

```
('b dans la fonction', 2)
2
>>> print(a)
1
```

(python)

```
def test2(b):
    b=[0]
    print("b dans la fonction",b)
    return b
```

(python/sage)

```
>>> a=[1]
>>> test2(a)
('b dans la fonction', [0])
[0]
>>> print(a)
[1]
>>> a=1
```

(python)

```
def test3(b):
    b[0]=b[0]+1
    print("b dans la fonction",b)
    return b
```

(python/sage)

```
>>> a=[1]
>>> test3(a)
('b dans la fonction', [2])
[2]
>>> print(a)
[2]
```

1.5.3 Arguments des fonctions et affectations sous Xcas

(giac/xcas)

```
test1(b):={
  b:=b+1;
  print("b dans la fonction",b);
  return b;
};
```

(giac/xcas)

```
test2(b):={
  b:=[2];//il y a recopie locale de b
  print("b dans la fonction",b);
  return b;
}
```

(giac/xcas)

```
test3(b):={
  b[0]:=b[0]+1;//il y a tout de meme recopie locale de b
  print("b dans la fonction",b);
  return b;
}
```

En utilisant l'affectation sur place =< on obtient le même comportement qu'en Python. (ie que les listes sont des pointeurs, mais que l'argument d'une fonction est recopié localement et donc non modifié.)

(giac/xcas)

```
test4(b):={
  b=<[2];//affectation sur place
  print("b dans la fonction",b);
  return b;
}
```

(giac/xcas)

```
test5(b):={
  b[0]=<b[0]+1;//affectation sur place
  print("b dans la fonction",b);
  return b;
}
```

(giac/xcas)

```
4>> (a:=1),test1(a),a ;
"b dans la fonction",2
1,2,1
5>> (a:=[1]),test2(a),a ;
"b dans la fonction",[1]
[1],[2],[1]
6>> (a:=[1]),test3(a),a ;
"b dans la fonction",[2]
[1],[2],[1]
7>> (a:=[1]),test4(a),a ;
"b dans la fonction",[2]
[1],[2],[1] //a n'est pas modifie car c'etait l'argument.
8>> (a:=[1]);test5(a);a ;
"b dans la fonction",[2]
[2],[2],[2]  //l'affichage  a lieu apres evaluation de toute la ligne ce qui
explique le premier 2. Le contenu de la liste a  est modifie car on a modifie
une entree de la liste alors que l'argument de la fonction etait la liste.
```