Introduction aux scripts Python pour du géotraitement ArcGIS

Licences professionnelles TIG et SIG-WEB

IUT STID Carcassonne-Perpignan

Juin 2012 – Avril 2013

IUT Perpignan

STID CARCASSONNE

UPVD Université de Perpignan Via Domitia

Table des matières

1. Quelques bases du langage Python.

1.1. L'éditeur Python IDLE.

La version Python 2.6.5 est livrée avec un éditeur « IDLE » (Integrated DeveLopment Environment).
Il s'ouvre sur une fenêtre interactive, dite fenêtre « Shell » dans laquelle chaque ligne est exécutée lors de sa validation.
Dans la fenêtre Shell, l'instruction print permet d'obtenir l'affichage d'une valeur.

```
>>> print 3/4,3.0/4
0 0.75
>>>
```

Pour ne pas aller à la ligne, il suffit de terminer par une virgule.

```
>>> for i in (1,2,3):
        print i,


1 2 3
>>>
```

Par Menu → File → New Window ou par Ctrl + N, on accède à une nouvelle fenêtre susceptible d'accueillir un script.
C'est dans une fenêtre de ce type que s'ouvre un fichier de script Python, d'extension py, quand on fait un clic droit sur son nom et que l'on demande son édition avec IDLE.
Depuis une fenêtre « Script », on peut ouvrir une fenêtre « Shell » par Menu → Run → Python Shell.

1.2. Edition.

Python distingue la casse.
Les commentaires commencent par le symbole dièse.

```
>>> # Ceci est un commentaire. Il est ignoré par l'interpréteur Python.
>>>
```

Par défaut, Python ne reconnaît que les caractères « unicode » de code ASCII inférieur ou égal à 128. Pour pouvoir utiliser, par exemple, des lettres accentuées dans les commentaires, il faut préciser en début de script que l'on souhaite appliquer le codage « windows-1252 » de l'Europe de l'Ouest. Cela se fait en écrivant le commentaire particulier suivant :
-*- coding: cp1252 -*-
Pour continuer une instruction sur la ligne suivante, on termine la ligne par un antislash (\).

```
>>> a="Commencement"\
   +" suite et fin"
>>> a
'Commencement suite et fin'
>>>
```

Le symbole d'affectation est le symbole égale (=). Les affectations peuvent être regroupées. Elles peuvent être transitives.

```
>>> a=b=1
>>> a,b
(1, 1)
>>> a,b=1,2
>>> a,b
(1, 2)
>>>
```

Les opérateurs de comparaison sont : > < >= <= == != is is not
Les comparaisons peuvent être regroupées.

```
>>> 1 < 2 < 3
True
>>>
```

Les opérateurs is et is not testent l'identité de deux objets.

```
>>> a,b = 1,1.0
>>> a == b, a is b
(True, False)
>>> 1<2
True
>>> 1!=2 # 1 différent de 2
True
>>> 2<=1+1
True
>>> 1==.5+.5 # 1 égale 0,5 + 0,5
True
>>> 2<1
False
>>>
```

1.3. Quelques types d'objets.

1.3.1. Les booléens.

Les booléens (type bool) sont True et False. La majuscule est obligatoire.
Opérateurs logiques : and, or, not.

```
>>> a,b = (1!=2),(1==2)
>>> a,b
(True, False)
>>> not a,not b, a and b, a or b, a and not b, not (a and b)
(False, True, False, True, True, True)
>>>
```

Tout objet peut être utilisé comme un booléen.
Le nombre zéro, la constante None, toute séquence vide (chaîne de caractères, liste, tuple), tout ensemble vide, tout dictionnaire vide sont considérés comme False.
Tout nombre non nul, toute séquence non vide, tout ensemble non vide, tout dictionnaire non vide sont considérés comme True.

```
>>> nb = 0
>>> ensemble = set((1,2,3))
>>> while ensemble:
        nb += 1
        ensemble.pop() # Retire un élément à l'ensemble


1
2
3
>>>
```

1.3.2. Les nombres.

Il y a des entiers (int) , des entiers longs (long) et des réels flottants (float).
Les symboles opératoires sont : + - * / // **

```
>>> a1,a2,a3,a4,a5,a6,a7,a8 = 5+2,5.0+2,5-2,5/2,5.0/2,5.0//2,5*2,5**2
>>> a1,a2,a3,a4,a5,a6,a7,a8
(7, 7.0, 3, 2, 2.5, 2.0, 10, 25)
>>>
```

Pour obtenir un cumul tel que « s reçoit s plus a », on peut écrire : s += a
De manière analogue, on peut écrire s -= a ou s *= a ou s /= a.

```
>>> s=5
>>> s+=3
>>> s
8
>>>
```

Les divisions sur des entiers sont entières ; la division entière générale est //
Conversions : int() ; long() ; float().

```
>>> x=(int("1"),float("2"),float(3),long(True),float(False))
>>> x
(1, 2.0, 3.0, 1L, 0.0)
>>>
```

Quelques fonctions mathématiques. La plupart sont dans le module math. Avant de les utiliser, il faut écrire : `>>> import math`

Fonction	Valeur retournée	type
abs(x)	Valeur absolue de x	float
round(x[,n])	Arrondi de x à n décimales	float
divmod(x,y)	(Quotient, Reste) de la division entière de a par b	float
math.sqrt(x)	Racine carrée de x	float
math.trunc(x)	Partie entière de x	float
math.exp(x)	Exp(x)	float
math.log(x)	Ln(x)	float
math.log10(x)	log (x) (logarithme décimal)	float
math.sin(x)	Sinus de x	float
math.cos(x)	Cosinus de x	float
math.tan(x)	Tangente de x	float

Constantes : math.pi ; math.e

```
>>> import math
>>> print math.sin(math.pi/2), math.e
1.0 2.71828182846
>>>
```

Nombres aléatoires : on y accède par le module random. Il faut d'abord écrire : `>>> import random`
La fonction random.seed() initialise le générateur de nombres aléatoires.
La fonction random.random() retourne un réel aléatoire de [0, 1[.
La fonction random.uniform(a, b) retourne un réel aléatoire de [a, b].
La fonction random.randint(a, b) retourne un entier aléatoire de [a, b].

```
>>> import random
>>> random.seed()
>>> random.random()
0.0356643412310765 35
>>> random.randint(1,100)
100
>>> random.uniform(1,100)
3.48923143097511
>>>
```

1.3.3. Les chaînes de caractères.

Les chaînes de caractères sont écrites entre simples cottes (sous le 4) ou entre doubles cottes (sous le 3).
Concaténation : + On peut aussi « multiplier » une chaîne de caractères.

```
>>> 2*("Un "+"Deux ")
'Un Deux Un Deux '
>>>
```

Caractères successifs d'une chaîne de caractères x : x[0], x[1], …
Caractères successifs d'une chaîne de caractères x de i jusqu'à j-1 : x[i:j]
Caractères successifs d'une chaîne de caractères x jusqu'à j-1 : x[:j]

Caractères successifs d'une chaîne de caractères x depuis i : x[i:]

```
>>> x="Voici un texte"
>>> x[0:],x[6],x[:6],x[6:],x[0:5],x[6:8]
('Voici un texte', 'u', 'Voici ', 'un texte', 'Voici', 'un')
>>>
```

Passage en majuscules, en minuscules : méthodes upper(), lower()

```
>>> x.upper()
'TEXTE'
>>>
```

Eclatement d'une chaîne en une liste de sous-chaînes : méthode split(séparateur)

```
>>> "mot1 mot2 mot3".split(" ")
['mot1', 'mot2', 'mot3']
>>>
```

len(x) retourne la longueur de la chaîne x.
str(x) convertit une variable non chaine x en variable chaine.
Les chaînes préfixées par u sont des chaînes unicode.
Si la chaîne n'est pas préfixée par r, l'antislash (\) a une signification particulière :

Escape Sequence	Meaning
\newline	Ignored
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\N{name}	Character named *name* in the Unicode database (Unicode only)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\uxxxx	Character with 16-bit hex value *xxxx* (Unicode only)
\Uxxxxxxxx	Character with 32-bit hex value *xxxxxxxx* (Unicode only)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value *ooo*
\xhh	Character with hex value *hh*

Si la chaîne est préfixée par r ou par R alors l'antislash est un caractère ordinaire. Dans ce cas la chaîne est dite : **raw string** (chaîne brute).

chr(i) retourne le caractère de code ASCII i ; ord(x) retourne le code ASCII du caractère x.

```
>>> ord('a'), ord('A'), ord('0'), ord('9'),chr(97), chr(65), chr(48), chr(57)
(97, 65, 48, 57, 'a', 'A', '0', '9')
>>>
```

Remarque.
Les chaînes de caractères apparaissent comme des tuples de caractères.

1.3.4. Les listes.

Une liste est une collection d'éléments appelés items. Elle est écrite entre crochets.

```
>>> x = [1,3.0/4,"Texte"]
>>> x
[1, 0.75, 'Texte']
>>>
```

Les items successifs sont séparés par une virgule. Les items ne sont pas forcément du même type.
Une liste peut être un item d'une autre liste.
Une liste est comparable à une chaîne de caractères. Cependant les items d'une liste sont modifiables alors que les caractères d'une chaîne de caractères ne le sont pas.
Items successifs d'une liste x : x[0], x[1], …
Items successifs d'une liste x de i jusqu'à j-1 : x[i:j]
Items successifs d'une liste x jusqu'à j-1 : x[:j]
Items successifs d'une liste x depuis i : x[i:]
len(x) retourne le nombre d'items de la liste x.

```
>>> x=[1,"un",2,"deux",3,"trois"]
>>> len(x),x[0],x[0:5],x[5:0]
(6, 1, [1, 'un', 2, 'deux', 3], [])
>>>
```

On peut « multiplier » une liste par un nombre entier positif ; on peut concaténer deux listes.

```
>>> liste1,liste2=[1,"un"],[2,"deux"]
>>> 2*(liste1+liste2)+[3,"trois"]
[1, 'un', 2, 'deux', 1, 'un', 2, 'deux', 3, 'trois']
>>>
```

Les items, les séquences d'items peuvent subir des modifications, des insertions et des suppressions.

```
>>> liste=[0,3,6,9]
>>> liste
[0, 3, 6, 9]
>>> liste[1:3]=["trois","six"]  # Modification des items 1 et 2
>>> liste
[0, 'trois', 'six', 9]
>>> liste[3:3]=[7,8]            # Insertion avant l'item 3
>>> liste
[0, 'trois', 'six', 7, 8, 9]
>>> liste[3:5]=[]               # Suppression des items 3 et 4
>>> liste
[0, 'trois', 'six', 9]
>>>
```

La méthode append ajoute un item à la liste.

```
>>> liste=["a","bc"]
>>> liste.append("d")
>>> liste
['a', 'bc', 'd']
>>>
```

La méthode insert insère un item dans la liste à une place choisie.

```
>>> liste=["a","c"]
>>> liste.insert(1,"b")
>>> liste
['a', 'b', 'c']
>>>
```

La méthode count retourne le nombre d'occurrences d'une valeur donnée, passée en paramètre.

```
>>> liste=[1,"un",2,"deux"]
>>> a=1
>>> liste.count(a)
1
>>>
```

La méthode pop retourne l'item dont l'indice est passé en paramètre et supprime cet item. En l'absence de paramètre, c'est le dernier item qui est choisi.

La fonction range([début,]fin[,pas])) retourne une liste d'entiers.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(4,-3)
[]
>>> range(4,-3,-1)
[4, 3, 2, 1, 0, -1, -2]
>>> range(1,10,2)
[1, 3, 5, 7, 9]
>>>
```

Les opérateurs in et not in testent la présence d'un élément dans une liste.

```
>>> liste=[0,1,2,3]
>>> 0 in liste
True
>>> 4 not in liste
True
>>> 2 in liste[:3]
True
>>> 3 in liste[:3]
False
>>>
```

1.3.5. Les tuples.

Un tuple, comme une liste, est une collection d'éléments appelés items. Elle est écrite entre parenthèses. Contrairement aux items d'une liste, ceux d'un tuple ne sont pas modifiables. Contrairement aux listes, les tuples (comme les chaînes de caractères) sont figés.
On accède aux items d'un tuple comme à ceux d'une liste. La fonction len est disponible. La méthode count également.

```
>>> tuple=(1,"un",2,"deux")
>>> tuple
(1, 'un', 2, 'deux')
>>> tuple[1:2]
('un',)
>>> len(tuple)
4
>>> tuple.count("deux")
1
>>>
```

1.3.6. Les ensembles.

Un ensemble (set) est une collection désordonnée. Elle est modifiable par les méthodes add(elt), remove(elt), discard(elt) (retire l'élément s'il est présent), pop() (extrait et retourne un certain élément ; le choix de cet élément est inconnu), clear(). Il existe aussi la classe frozenset des ensembles gelés. Ceux-là ne sont pas modifiables. Voici quelques opérations permises sur les ensembles (gelés ou modifiables) :
len(ens), x in ens, x not in ens, ens1 <= ens2, ens1 < ens2, ens1 >= ens2, ens1 > ens2, ens1 | ens2 | ens3 … (union), ens1 & ens2 & ens3 … (intersection), ens1 – ens2, ens1 ^ ens2 (différence symétrique).
Les méthodes copy() et isdisjoint(ens) sont disponibles dans les deux classes set et frozenset.

```
>>> ens1, ens2, ens3 = set((1,2,2,3)), set((2,3,4,4)), set ((3,4,5))
>>> ens1, ens2, ens3
(set([1, 2, 3]), set([2, 3, 4]), set([3, 4, 5]))
>>> ens1 | ens2 | ens3
set([1, 2, 3, 4, 5])
>>> ens1 & ens2 & ens3
set([3])
>>> len(ens1)
3
>>> ens1.isdisjoint(ens2)
False
>>> ens1.discard(5)
>>> ens1
set([1, 2, 3])
>>> ens1.discard(3)
>>> ens1
set([1, 2])
>>> ens3.copy(ens1 | ens2)
>>> ens = ens3.copy()
>>> ens
set([3, 4, 5])
>>>
```

1.3.7. Les dictionnaires.

Un dictionnaire (dict) est une collection désordonnée de couples clef:valeur. Les clefs doivent être toutes différentes. Il se présente ainsi : {clef1:valeur1, clef2:valeur2, clef3:valeur3, …}.
Le dictionnaire {clef1:valeur1, clef2:valeur2, clef3:valeur3} peut être obtenu par chacune des affectations suivantes :
dico = dict(clef1=valeur1, clef2=valeur2, clef3=valeur3)
dico = dict({clef1:valeur1, clef2:valeur2, clef3:valeur3})
dico = dict(zip((clef1 ,clef2 ,clef3), (valeur1, valeur2, valeur3)))

```
>>> dico = dict(clef1="valeur1", clef2="valeur2", clef3="valeur3")
>>> dico
{'clef1': 'valeur1', 'clef3': 'valeur3', 'clef2': 'valeur2'}
>>> dico = dict({"clef1":"valeur1", "clef2":"valeur2", "clef3":"valeur3"})
>>> dico
{'clef1': 'valeur1', 'clef3': 'valeur3', 'clef2': 'valeur2'}
>>> dict(zip(("clef1" ,"clef2" ,"clef3" ), ("valeur1", "valeur2", "valeur3")))
{'clef1': 'valeur1', 'clef3': 'valeur3', 'clef2': 'valeur2'}
>>>
```

Voici quelques fonctions et quelques méthodes applicables à un dictionnaire nommé dico.

Fonction ou méthode	Valeur retournée ou action
len(dico)	Nombre de paire clef:valeur
dico[clef]	Valeur
dico[clef] = valeur	Affectation
del dico[clef]	Supprime la paire clef:valeur
del dico	Supprime dico
dico.clear()	Vide dico
clef in dico	Booléen
clef not in dico	Booléen
dico.copy()	Retourne une copie de dico
dico.get(clef)	Retourne la valeur ; retourne None si clef est absente.
dico.items()	Retourne la liste des paires (clef :valeur)
dico.keys()	Retourne la liste des clefs
dico.values()	Retourne la liste des valeurs

```
>>> "clef1" in dico
True
>>> "valeur1" in dico
False
>>> dico.get("clef2")
'valeur2'
>>> dico.items()
[('clef1', 'valeur1'), ('clef3', 'valeur3'), ('clef2', 'valeur2')]
>>> dico.keys()
['clef1', 'clef3', 'clef2']
>>> dico.values()
['valeur1', 'valeur3', 'valeur2']
>>>
```

1.3.8. Les dates.

Les méthodes et les propriétés utiles dans le traitement des dates se trouvent dans le module datetime.
Il faut donc importer ce module. Les exemples illustrant ce paragraphe sont précédés de la ligne :

```
>>> from datetime import *
```

La date définie par les entiers an, mois (de 1 à 12), jour est : date (an, mois, jour)

```
>>> print date(1789,7,14)
1789-07-14
>>>
```

La date du jour est : date.today().

```
>>> print date.today()
2011-08-10
>>>
```

On accède à l'année, au mois, au jour et au jour de la semaine d'une date par ses attributs year, month, day et par la méthode weekday(). Ce sont des entiers. Les jours de la semaine sont numérotés de 0 à 6, du lundi au dimanche.

```
>>> aujourdhui=date.today()
>>> print aujourdhui.year,aujourdhui.month,aujourdhui.day,aujourdhui.weekday()
2011 8 10 2
>>>
```

La date-heure actuelle est : datetime.now().

```
>>> print datetime.now()
2011-08-10 16:16:16.808000
>>>
```

Une date-heure contient l'année, le mois, le jour, l'heure, les minutes et les secondes. On accède à ces informations par les attributs year, month, day, hour, minute, second, microsecond.
On dispose encore de la méthode weekday(), comme pour une date et de la méthode date() qui retourne la date.

```
>>> maintenant=datetime.now()
>>> maintenant
datetime.datetime(2011, 8, 10, 16, 28, 13, 984000)
>>> print maintenant.date(), maintenant.weekday()
2011-08-10 2
>>> print maintenant.year, maintenant.month, maintenant.day
2011 8 10
>>> print maintenant.hour, maintenant.minute
16 28
>>> print maintenant.second, maintenant.microsecond
13 984000
>>>
```

La méthode isoformat() retourne une chaîne de caractères donnant la date ou la date-heure au format ISO 8601.

```
>>> print aujourdhui.isoformat(), maintenant.isoformat()
2011-08-10 2011-08-10T16:28:13.984000
>>>
```

La méthode strftime(chaîne-format) retourne la date ou la date-heure au format défini par la chaîne de caractères passée en paramètre. Le tableau qui suit précise la syntaxe.
Inversement, la méthode strptime(chaîne, chaîne-format) retourne la date ou la date-heure, définie par l'argument chaîne, selon le format défini par l'argument chaîne-format.

Directive	Meaning
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%c	Locale's appropriate date and time representation.
%d	Day of the month as a decimal number [01,31].
%f	Microsecond as a decimal number [0,999999], zero-padded on the left
%H	Hour (24-hour clock) as a decimal number [00,23].
%I	Hour (12-hour clock) as a decimal number [01,12].
%j	Day of the year as a decimal number [001,366].
%m	Month as a decimal number [01,12].
%M	Minute as a decimal number [00,59].
%p	Locale's equivalent of either AM or PM.
%S	Second as a decimal number [00,61].
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.
%w	Weekday as a decimal number [0(Sunday),6].
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.
%x	Locale's appropriate date representation.
%X	Locale's appropriate time representation.
%y	Year without century as a decimal number [00,99].
%Y	Year with century as a decimal number.
%z	UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive).
%Z	Time zone name (empty string if the object is naive).
%%	A literal '%' character.

Voici quelques exemples.

```
>>> print datetime.strptime("31/03/2012","%d/%m/%Y")
2012-03-31 00:00:00
>>> print aujourdhui.strftime("%x"), maintenant.strftime("%X")
08/10/11 16:28:13
>>> print aujourdhui.strftime("%c"), maintenant.strftime("%c")
08/10/11 00:00:00 08/10/11 16:28:13
>>> print aujourdhui.strftime("Carcassonne, le %A %d %B %Y")
Carcassonne, le Wednesday 10 August 2011
>>>
```

La différence entre deux dates-heures, chacune pouvant être une simple date, est un objet de la classe timedelta. La syntaxe pour créer un objet timedelta est :

timedelta([*days*[, *seconds*[, *microseconds*[, *milliseconds*[, *minutes*[, *hours*[, *weeks*]]]]]]])

Tous les paramètres sont facultatifs et nuls par défaut.
Un objet timedelta possède les attributs days, seconds et microseconds.

```
>>> delai=timedelta(366) #2012 est une année bissextile
>>> aujourdhui=date.today()
>>> dans_un_an=aujourdhui+delai
>>> print delai, aujourdhui, dans_un_an
366 days, 0:00:00 2011-08-10 2012-08-10
>>>
```

1.3.9. La constante None.

Cette constante est en particulier utilisée pour représenter l'absence de valeur, par exemple lorsqu'un paramètre facultatif d'une fonction n'est pas renseigné. Elle est la seule valeur de type NoneType. On peut tester si une variable est None à l'aide des opérateurs de comparaison is ou is not.

```
>>> a,b,c = 0, "", None
>>> a is None, b is None, c is None
(False, False, True)
>>>
```

1.3.10. Les fichiers.

Nous nous limiterons aux fichiers textes.

Si le nom d'un fichier est simple, sans le chemin d'accès, Python recherche le fichier dans le dossier courant. Ce dossier, par défaut, est celui du script. La fonction path du module sys retourne une liste de dossiers. sys.path[0] (en lecture et en écriture) est le dossier courant.

Code	Effet
mon_fichier = open(nom_du_fichier, "w")	Crée un objet mon_fichier qui fait référence à un fichier. nom_du_fichier est une chaîne de caractères. S'il existe un fichier nommé nom_du_fichier, il est vidé puis ouvert en écriture. Sinon, il est créé (éventuellement dans le répertoire courant) puis ouvert en écriture
mon_fichier = open(nom_du_fichier, "r")	Crée un objet mon_fichier qui fait référence à un fichier. nom_du_fichier est une chaîne de caractères. S'il existe un fichier nommé nom_du_fichier, il est ouvert en lecture. Sinon une erreur se produit
mon_fichier.readline()	Retourne la prochaine ligne du fichier ouvert en lecture. La ligne est retournée en tant que chaîne de caractères. Si le fichier est épuisé, la méthode retourne la chaîne vide "".
mon_fichier.readlines()	Retourne la liste des lignes restant à lire dans le fichier ouvert en lecture. Si le fichier est épuisé, la méthode retourne la liste vide [].
mon_fichier.tell()	Retourne le numéro du caractère actuel dans le fichier : 0, 1, 2, …
mon_fichier.seek(numéro_de_caractère)	Force le caractère actuel.
mon_fichier.write(chaîne_de_caractères)	Ecrit la chaîne_de_caractères dans le fichier, s'il est ouvert en écriture. Par défaut, l'écriture se fait en fin de fichier. Si l'on veut écraser à partir du caractère numéro i, il faut d'abord écrire : mon_fichier.seek(i) Le changement de ligne n'est pas automatique. Il faut l'écrire à la fin de la chaîne de caractères ("\n"). Il compte pour deux caractères.

mon_fichier.writelines(liste de chaînes_de_caractères)	Ecrit les chaînes_de_caractères dans le fichier, s'il est ouvert en écriture. Attention : les changements de lignes ne sont pas automatiques. Ils doivent être incorporés aux chaînes_de_caractères.
for ligne in mon_fichier:	Parcours des lignes du fichier ouvert en lecture

```
>>> fic = open("mon_fichier.txt","w")
>>> fic.writelines(["ligne 1\n","ligne 2\n","ligne 3\n","ligne 4\n"])
>>> fic.tell()
36L
>>> fic.seek(9) #Début de la seconde ligne
>>> fic.write("abcdefg") # On remplace "ligne 2" par "abcdefg"
>>> fic.seek(36) # fin du fichier
>>> fic.write("ligne 5\n") # On ajoute une 5ème ligne
>>> fic.close()
>>> fic = open("mon_fichier.txt","r") #Ouvert en lecture
>>> contenu = fic.readlines()
>>> contenu
['ligne 1\n', 'abcdefg\n', 'ligne 3\n', 'ligne 4\n', 'ligne 5\n']
>>> for ligne in fic:
        print ligne,


>>> fic.seek(0)
>>> for ligne in fic:
        print ligne


ligne 1

abcdefg

ligne 3

ligne 4

ligne 5

>>> fic.seek(9) #début de la seconde ligne
>>> fic.readline()
'abcdefg\n'
>>> fic.close()
>>> fic = open("mon_fichier.txt","w")
>>> fic.close()
>>> fic = open("mon_fichier.txt","r")
>>> fic.readlines()
[]
>>> # le fichier a été vidé lors de son ouverture en écriture.
```

1.3.11. Quelques types.

Voici quelques types fondamentaux :

int long float bool str list tuple dict set frozenset file NoneType

La fonction type() retourne le type de l'objet passé en paramètre.

```
>>> type(0)
<type 'int'>
>>> type(0L)
<type 'long'>
>>> type(0.0)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type('0')
<type 'str'>
>>> type([0,1,2])
<type 'list'>
>>> type((0,1,2))
<type 'tuple'>
>>> type(None)
<type 'NoneType'>
>>>
```

1.4. Quelques instructions.

1.4.1. L'instruction del.

L'instruction del est l'instruction d'effacement.

```
>>> liste = ["zéro","un","deux","trois", "quatre", "cinq"]
>>> liste
['z\xe9ro', 'un', 'deux', 'trois', 'quatre', 'cinq']
>>> del liste[0]
>>> liste
['un', 'deux', 'trois', 'quatre', 'cinq']
>>> del liste [3:]
>>> liste
['un', 'deux', 'trois']
>>> liste.append([4,5,6])
>>> liste
['un', 'deux', 'trois', [4, 5, 6]]
>>> del liste[1:3]
>>> liste
['un', [4, 5, 6]]
>>> del liste
>>> liste

Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    liste
NameError: name 'liste' is not defined
>>>
```

1.4.2. L'instruction pass.

L'instruction pass ne produit aucune action. On l'utilise quand on a besoin d'une instruction pour la correction de la syntaxe (dans un bloc except par exemple) mais qu'on ne souhaite aucune action.

1.4.3. La fonction time.sleep(nb_secondes).

Le module time contient une fonction de temporisation : la fonction sleep. Son argument (entier ou pas) est le nombre de secondes de temporisation.

```
>>> import time
>>> for i in range(5):
        time.sleep(1) # Temporisation d'une seconde
        print i
```

1.4.4. Edition d'un bloc d'instructions.

Les principales instructions composées sont for, while et if. La ligne déclarative se termine par deux points (:). Le bloc associé est indenté (décalé d'une tabulation vers la droite). Il se termine quand se termine l'indentation.

1.4.5. L'instruction for.

La syntaxe est du type : for identificateur in liste:
L'identificateur prend successivement pour valeur chaque item de la liste ; le bloc « for » est exécuté successivement pour chacune de ces valeurs.

```
>>> for i in ["un","deux","trois"]:
        print i,", ",


un ,  deux ,  trois ,
>>> for i in ["un","deux","trois"]:
        print i,


un deux trois
>>> for i in range(5):
        print i,


0 1 2 3 4
>>> somme=0
>>> for i in range(1,10,2):
        somme=somme+i
        print "i =",i,"    somme =",somme


i = 1     somme = 1
i = 3     somme = 4
i = 5     somme = 9
i = 7     somme = 16
i = 9     somme = 25

>>>
```

L'instruction break fait sortir du bloc for. L'instruction continue fait passer à l'itération suivante.

1.4.6. L'instruction while.

La syntaxe est du type : while condition :
condition est un booléen. Le bloc « while » est exécuté tant que condition est True.

```
>>> # Suite de Fibonacci:
>>> # Suite de Fibonacci. Chaque terme est la somme des 2 termes précédents.
>>> a,b=0,1
>>> while b<10:
        print b,
        a,b=b,a+b


1 1 2 3 5 8
>>>
```

L'instruction break fait sortir du bloc while. L'instruction continue fait passer à l'itération suivante.

1.4.7. L'instruction if.

La syntaxe est du type : if condition0 :
…
elif condition1:

```
        …
elif condition2:
        …
elif condition3:
        …
else:
        …
```

« elif » signifie « else if ». Il peut y avoir 0,1 ou plusieurs « elif ». Il peut y avoir 0 ou 1 « else ».

1.5. Les fonctions.

1.5.1. Comment écrire une fonction ?

Une fonction est déclarée par le mot clé def suivi du nom de la fonction et, entre parenthèses, des éventuels paramètres passés à cette fonction. La ligne de déclaration de la fonction se termine par deux points (:).
Le bloc indenté suivant les deux points constitue la fonction. Il est exécuté à chaque appel de la fonction.
Le mot clé return est suivi de la valeur que retourne la fonction. Il met fin à l'exécution de la fonction.
Il peut y avoir 0, 1 ou plusieurs « return ». Une fonction qui ne possède aucun return retourne la valeur None.

```
>>> def ma_fonction():
        print "Fin"


>>> ma_fonction() is None
Fin
True
>>>
```

Il est conseillé de faire suivre la ligne déclarative d'un texte décrivant la fonction. Ce texte constitue la « docstring » de la fonction.

```
>>> def somme(premier, dernier):
        "Retourne la somme des entiers de premier à dernier"
        cumul=0
        for i in range(premier,dernier+1):
                cumul+=i
        return cumul

>>> somme(1,10)
55
>>>
```

La valeur retournée par une fonction est récupérable dans une variable. Ci-dessus, on aurait pu écrire :

```
>>> Total = somme(1,10)
>>> Total
55
>>>
```

Pour rendre un paramètre facultatif, il suffit de lui affecter une valeur dans la déclaration de la fonction. Tout paramètre facultatif prend la valeur qu'on lui affecte lors de l'appel si on lui en affecte une ; sinon, il garde la valeur qui lui a été affectée dans la déclaration. Il peut être judicieux d'affecter la valeur None à un paramètre facultatif dans la déclaration d'une fonction.

```
>>> def classe(objet=None):
        "Retourne la classe de l'objet"
        son_type=type(objet)
        if son_type in (int, long, float):
                return "Nombre"
        elif son_type is str:
                return "Texte"
        elif son_type is bool:
                return "Vrai-Faux"
        elif son_type in (list, tuple):
                return "Sequence"
        elif objet is None:
                return "Non defini"
        else:
                return "Classe inconnue"


>>> a1,a2,a3=classe(0),classe("0"),classe(False)
>>> a4,a5=classe((0,1)),classe()
>>> a1,a2,a3,a4,a5
('Nombre', 'Texte', 'Vrai-Faux', 'Sequence', 'Non defini')
>>>
```

Par défaut, toute variable affectée dans une fonction est locale à la fonction. Pour la rendre visible à l'extérieur de la fonction, il faut la déclarer global avant son affectation.

```
>>> x,y
(0, 0)
>>> x,y = 0,0
>>> x,y = 0,0
>>> def fonct():
        global x
        x,y = 1,1


>>> x,y
(0, 0)
>>> fonct()
>>> x,y
(1, 0)
>>>
```

1.5.2. Quelques fonctions d'information prédéfinies.

La fonction help(*{argument}*) retourne une aide sur son argument.
La fonction dir(*{argument}*) retourne la liste des propriétés de son argument.
La fonction type(*objet*) retourne le type de l'objet passé en argument.

```
>>> help(dir)
Help on built-in function dir in module __builtin__:

dir(...)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attribut
es
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.

>>> x = "texte"
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__
format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__get
slice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__', '__mo
d__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
 '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook
__', '_formatter_field_name_split', '_formatter_parser', 'capitalize', 'center',
 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index
', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper',
'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', '
rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', '
strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>

>>> type(0)
<class 'int'>
>>> type(0.0)
<class 'float'>
>>> del datetime
>>> type(0.0)
<class 'float'>
>>> type(0)
<class 'int'>
>>> type(0.0)
<class 'float'>
>>> import datetime
>>> delta = datetime.timedelta()
>>> type(delta)
<class 'datetime.timedelta'>
>>> dir(delta)
['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__divmod__', '__
doc__', '__eq__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '_
_gt__', '__hash__', '__init__', '__le__', '__lt__', '__mod__', '__mul__', '__ne_
_', '__neg__', '__new__', '__pos__', '__radd__', '__rdivmod__', '__reduce__', '_
_reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '__rmul__', '__rsub__',
'__rtruediv__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclassho
ok__', '__truediv__', 'days', 'max', 'microseconds', 'min', 'resolution', 'secon
ds', 'total_seconds']
>>>
```

```
>>> import arcpy
>>> help(arcpy)
Help on package arcpy:

NAME
    arcpy

FILE
    c:\program files (x86)\arcgis\desktop10.0\arcpy\arcpy\__init__.py

DESCRIPTION
    # -*- coding: utf-8 -*-
    #COPYRIGHT 2010 ESRI
    #
    #TRADE SECRETS: ESRI PROPRIETARY AND CONFIDENTIAL
    #Unpublished material - all rights reserved under the
    #Copyright Laws of the United States.
    #
    #For additional information, contact:
    #Environmental Systems Research Institute, Inc.
    #Attn: Contracts Dept
    #380 New York Street
    #Redlands, California, USA 92373
    #
    #email: contracts@esri.com

PACKAGE CONTENTS
    _base
    _ga
    _graph
    _importable_modules
    _management
    _mapping
    analysis
    arc
    arcobjects (package)
    ba
    cartography
    conversion
```

(C'est seulement le début !)

1.6. Les boîtes à messages.

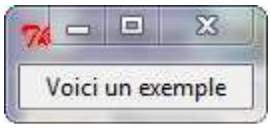
Il n'existe pas de boîte à message standard. Il est facile d'en construire une. Voici un code possible :

```
from Tkinter import *
def msg(texte):
    fen = Tk()
    tex = Label(fen, text = texte, fg = 'black')
    tex.pack()
    fen.mainloop()
```

Pour utiliser cette boîte à message et lui faire afficher "Voici un exemple", on écrira :

```
msg("Voici un exemple")
```

On obtient :



1.7. Les modules.

Les bibliothèques sont organisées en « packages » et en modules. Les packages sont matérialisés par des dossiers et les modules par des fichiers Python. Un module contient des classes et des fonctions. Un package peut lui aussi contenir des classes et des fonctions ; il contient zéro, un ou plusieurs packages ; il contient zéro, un ou plusieurs modules. Par exemple, le package ArcPy (arcpy) (fourni avec ArcGIS DeskTop) contient (parmi d'autres) le module mapping (arcpy.mapping) et le package Spatial Analyst (arcpy.sa).
Le terme « module » est utilisé pour désigner les packages aussi bien que les modules.
Pour importer un module (ou un package), on utilise l'instruction import éventuellement accompagnée du mot clé from.

```
>>> import os, sys, math, string
>>> # arcpy est un gros package. On importe seulement quelques modules
>>> from arcpy import env, mapping as carte
>>> # au lieu d'écrire arcpy.mapping, on écrira carte
>>> from compiler import * # Comme import compiler, mais plus besoin du préfixe
>>> # compiler
```

On peut écrire soi-même un script pour en faire un module. Pour importer le module « mon_module.py », on écrira : import mon_module. Si « mon_module.py » est dans le dossier courant, il sera importé. S'il est ailleurs, on commencera par ajouter le nom complet de son dossier à la liste sys.path des chemins à explorer. Lors de l'importation, le code éventuel du script « mon_module.py » est exécuté. Les fonctions définies dans ce script deviennent disponibles. Il faut les préfixer par mon_module. Voici par exemple un module « moyennes.py » situé dans le dossier C:\Cours\ArcGIS\Python 2011-2012\Documentation et un script d'appel « appel ».
Module « moyennes.py » :

```
# -*- coding: cp1252 -*-

# Module de calcul de moyennes.

print "Module de calcul de moyenne"

import math #Pour l'acces aux logarithmes et aux exponentielles.

def moy_arith(nbs):
    " Retourne la moyenne arithmétique des nombres de la liste nbs."
    n = len(nbs) # Nombre d'items dans la liste nbs
    if n > 0:
        total = 0.0
        for nb in nbs:
            total += nb
        return total/n

def moy_geom(nbs):
    """ Retourne la moyenne géométrique des nombres strictement positifs
    de la liste nbs."""
    n = len(nbs) # Nombre d'items dans la liste nbs
    if n > 0:
        pdt = 1
        for nb in nbs:
            pdt *= nb
        return math.exp(math.log(pdt)/n)
```

```
def moy_harm(nbs):
    " Retourne la moyenne harmonique des nombres non nuls de la liste nbs."
    n = len(nbs) # nombre d'items dans la liste nbs
    if n > 0:
        total_inv = 0.0
        for nb in nbs:
            total_inv += 1.0/nb
        return float(n)/total_inv
```

Script d'appel « appel.py » :

```
# -*- coding: cp1252 -*-

# Script d'appel du module moyennes.

import sys
sys.path.append(r'C:\Cours\ArcGIS\Python 2011-2012\Documentation')
import moyennes

nbs = [1,3,5,7,9]
arith = moyennes.moy_arith(nbs)
print "Moyenne arithmétique de 1, 3, 5, 7, 9 :",arith
geom = moyennes.moy_geom(nbs)
print "Moyenne géométrique de 1, 3, 5, 7, 9 :",geom
harm = moyennes.moy_harm(nbs)
print "Moyenne harmonique de 1, 3, 5, 7, 9 :",harm
```

Résultats obtenus dans la fenêtre Shell :

```
>>>
Module de calcul de moyenne
Moyenne arithmétique de 1, 3, 5, 7, 9 : 5.0
Moyenne géométrique de 1, 3, 5, 7, 9 : 3.93628342704
Moyenne harmonique de 1, 3, 5, 7, 9 : 2.79751332149
>>>
```

Lors de la première importation, le module est « compilé ». Un fichier d'extension .pyc est produit. C'est lui que Python utilise. Si l'on souhaite modifier le module, il faut détruire ce fichier .pyc.

1.8. La gestion des erreurs.

Quand une erreur d'exécution se produit, une « exception » survient. L'erreur peut être plus ou moins grave. Elle nécessite parfois l'interruption de l'exécution. D'autres fois il s'agit d'une simple mise en garde. Voici le tableau des exceptions.

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
```

```
|     |         +-- WindowsError (Windows)
|     |         +-- VMSError (VMS)
|     +-- EOFError
|     +-- ImportError
|     +-- LookupError
|     |    +-- IndexError
|     |    +-- KeyError
|     +-- MemoryError
|     +-- NameError
|     |    +-- UnboundLocalError
|     +-- ReferenceError
|     +-- RuntimeError
|     |    +-- NotImplementedError
|     +-- SyntaxError
|     |    +-- IndentationError
|     |         +-- TabError
|     +-- SystemError
|     +-- TypeError
|     +-- ValueError
|          +-- UnicodeError
|               +-- UnicodeDecodeError
|               +-- UnicodeEncodeError
|               +-- UnicodeTranslateError
+-- Warning
     +-- DeprecationWarning
     +-- PendingDeprecationWarning
     +-- RuntimeWarning
     +-- SyntaxWarning
     +-- UserWarning
     +-- FutureWarning
     +-- ImportWarning
     +-- UnicodeWarning
     +-- BytesWarning
```

De nouvelles exceptions ont été définies dans les classes arcpy.ExecuteError et arcpy.ExecuteWarning pour gérer les incidents qui se produisent lors de l'exécution d'un outil de géotraitement.
Les exceptions peuvent être récupérées par les instructions composées try, except, else et finally.
Si une exception survient dans l'exécution du bloc try, cette exécution est interrompue. Si l'exception est prévue dans une clause except, le contrôle passe à ce bloc except.
La clause except peut se présenter ainsi :
except: interception de toutes les exceptions
except Exception : interception de toutes les exceptions.
except ZeroDivisionError: interception de la seule division par zéro (c'est un exemple).
except (FloatingPointError, OverflowError, ZeroDivisionError) : interception de ces 3 exceptions.
Il faut au moins une clause except après une clause try.
On peut récupérer puis afficher un message d'erreur :

```
except Exception as mon_exception :
    print mon_exeption
```

Le bloc finally, facultatif, est toujours exécuté quand il est présent, qu'il y ait une erreur ou qu'il n'y en ait pas.
En résumé :

- Le bloc try est exécuté jusqu'à ce qu'une éventuelle exception survienne ;
- Chaque bloc except (il y en a au moins 1 après un bloc try) est exécuté si, et seulement si, l'une des exceptions correspondantes survient ; les exceptions non prévues ne sont pas interceptées ;
- Le bloc else est facultatif ; s'il est présent, il est exécuté si, et seulement si, aucune exception ne survient ;
- Le bloc finally est facultatif ; s'il est présent, il est exécuté qu'une exception se produise ou pas.

Voici quelques illustrations.

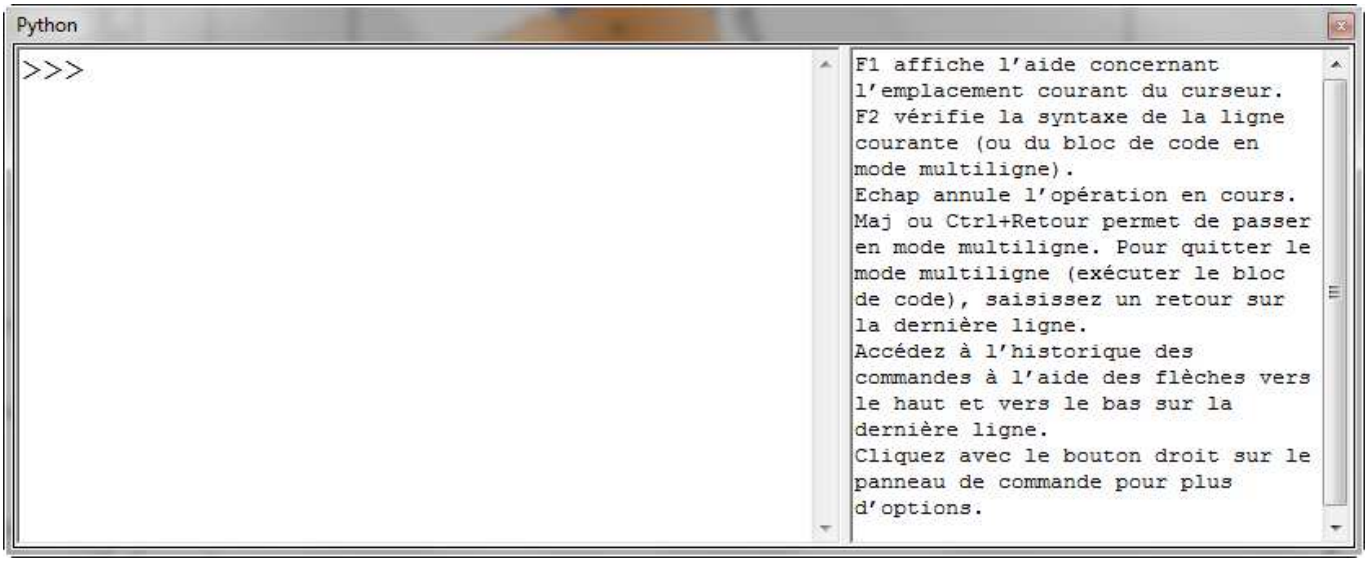
```
>>> def exemple1():
        try:
                1/0
        except Exception as mon_exception:
                print mon_exception
        finally:
                print "fin de l'exemple1"

>>> exemple1()
integer division or modulo by zero
fin de l'exemple1
>>>
>>> def exemple2(a,b):
        """Division de a par b"""
        try:
                a/b
        except ZeroDivisionError as division_par_zero:
                print "division par zero",division_par_zero
        except Exception as autre_exception:
                print "autre exception", autre_exception
        else:
                print a,"sur",b,"=",a/b
        finally:
                print "fin de l'exemple2"
>>> exemple2(3.0,5.0)
3.0 sur 5.0 = 0.6
fin de l'exemple2
>>> exemple2(3,0)
division par zero integer division or modulo by zero
fin de l'exemple2
>>> exemple2("trois","cinq")
autre exception unsupported operand type(s) for /: 'str' and 'str'
fin de l'exemple2
>>>
```

2. Python dans ArcGIS.

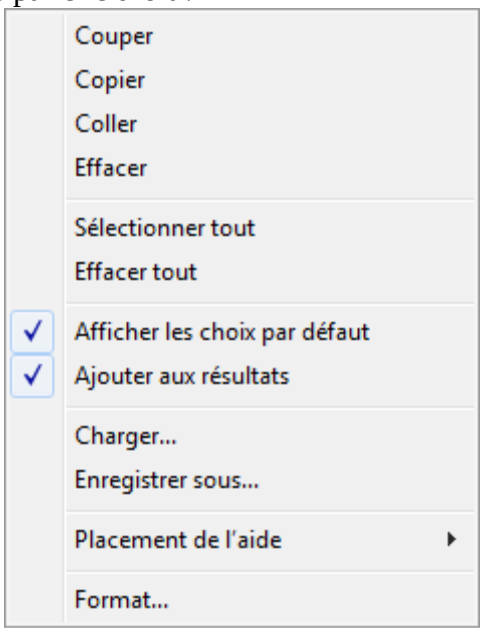
2.1. Les scripts

2.1.1. La fenêtre Python.

Une fenêtre interactive pour édition de script Python est disponible dans ArcMap. On l'ouvre par l'icône de la barre d'outils Standard ou par Menu, Géotraitement, Python. Le volet de gauche est pour l'édition. Celui de droite est pour l'aide. Le mode d'emploi est visible sur la copie d'écran que voici.



Un menu contextuel est disponible par clic droit :



2.1.2. Les outils de scripts.

La meilleure façon d'exécuter un script Python dans ArcGIS est d'en faire **un outil de script**. On dispose ainsi des boîtes de dialogues standards pour les outils. Il est conseillé de créer une boîte à outils spécifique pour recevoir les outils de scripts.

2.1.2.1. Comment créer une nouvelle boîte à outils ?

Dans la fenêtre Catalogue, on sélectionne le dossier d'accueil et, par clic droit, on accède à Nouveau, Boîte à outils. Une boîte à outils est implémentée sous la forme d'un fichier d'extension .tbx. Une boîte à

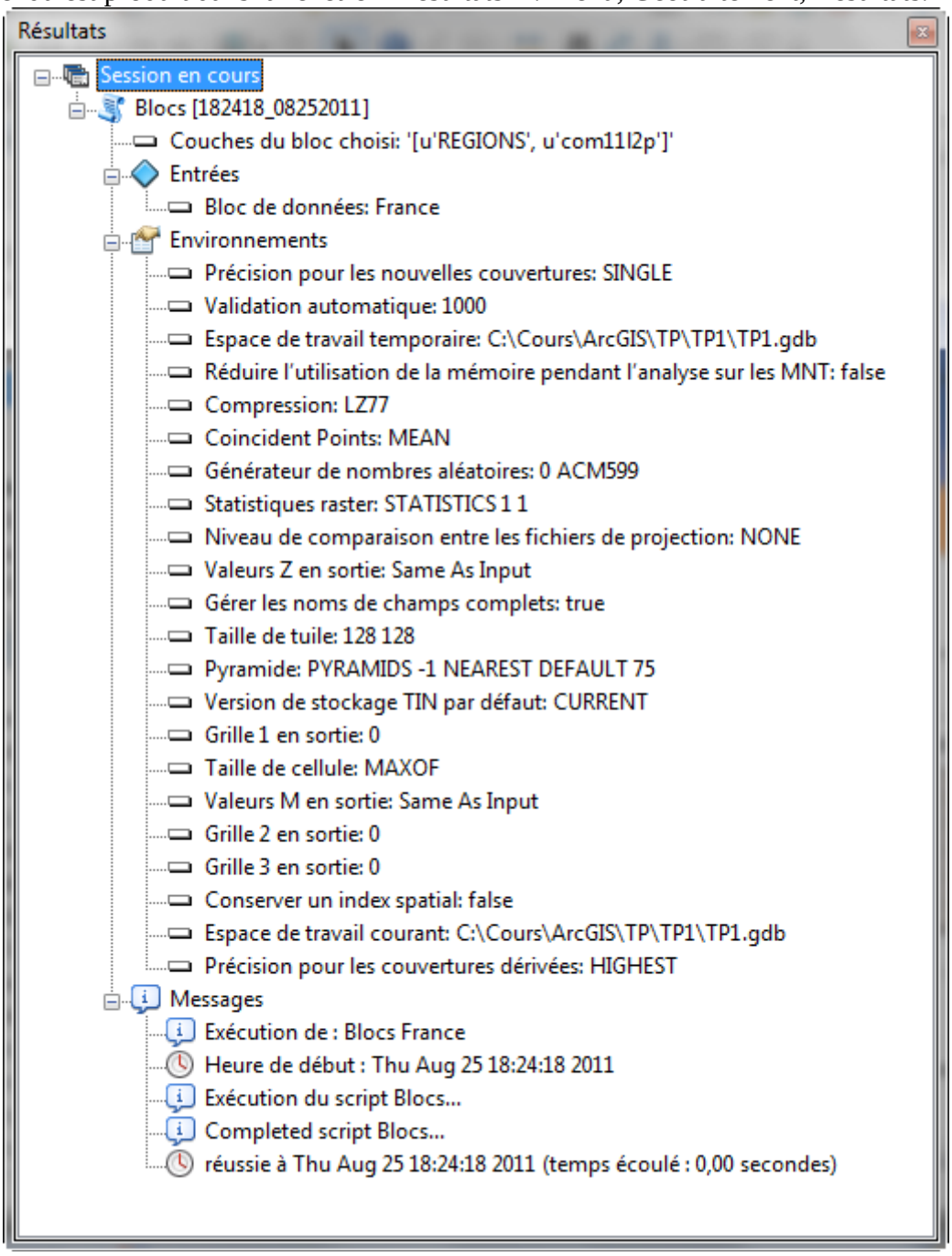
outils est plus facile à sauvegarder puis à importer (sur un autre ordinateur par exemple) qu'un outil particulier.

2.1.2.2. Comment ajouter un outil de script à une boîte à outils ?

Dans la fenêtre Catalogue, par clic droit sur la boîte d'accueil, on accède à Ajouter, Script. Il s'ouvre un assistant très voisin de la fenêtre « Propriétés » décrite plus bas.

2.1.2.3. Comment suivre l'exécution d'un outil ?

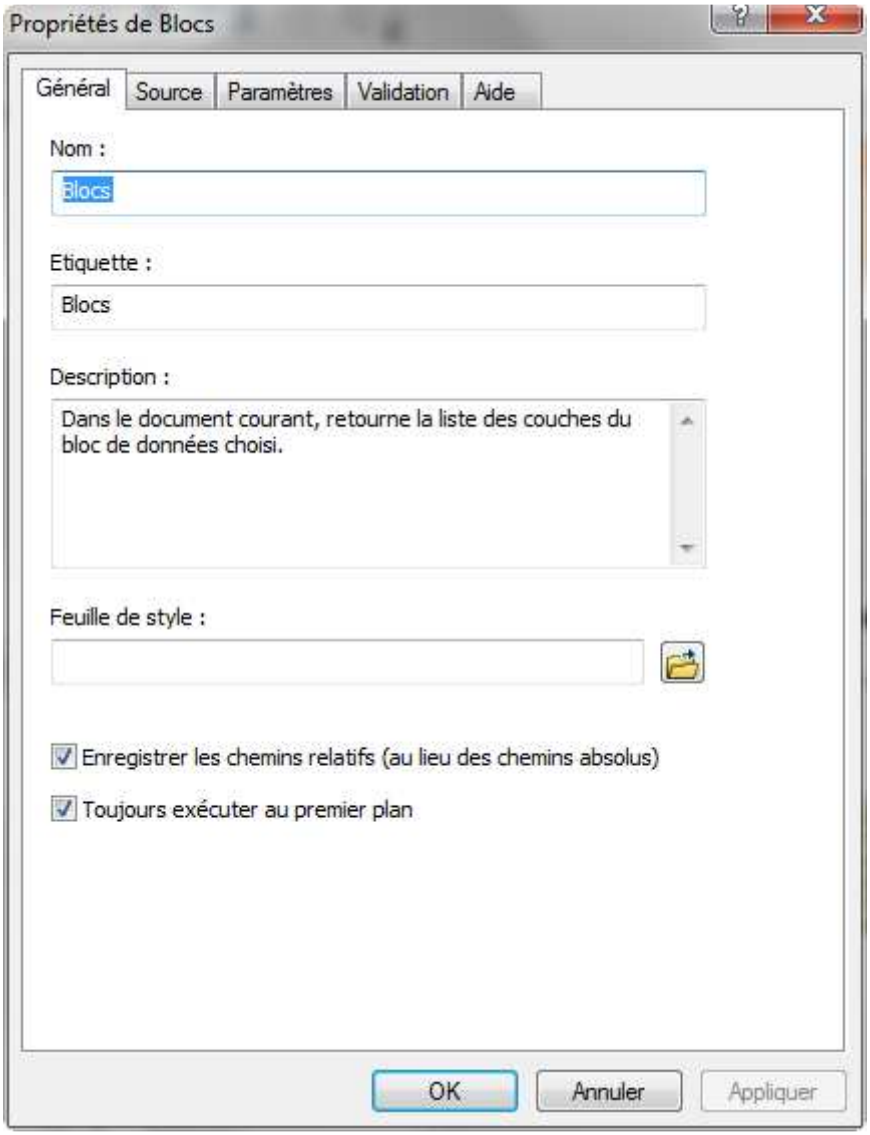
Un compte-rendu est produit dans la fenêtre « Résultats » : Menu, Géotraitement, Résultats.



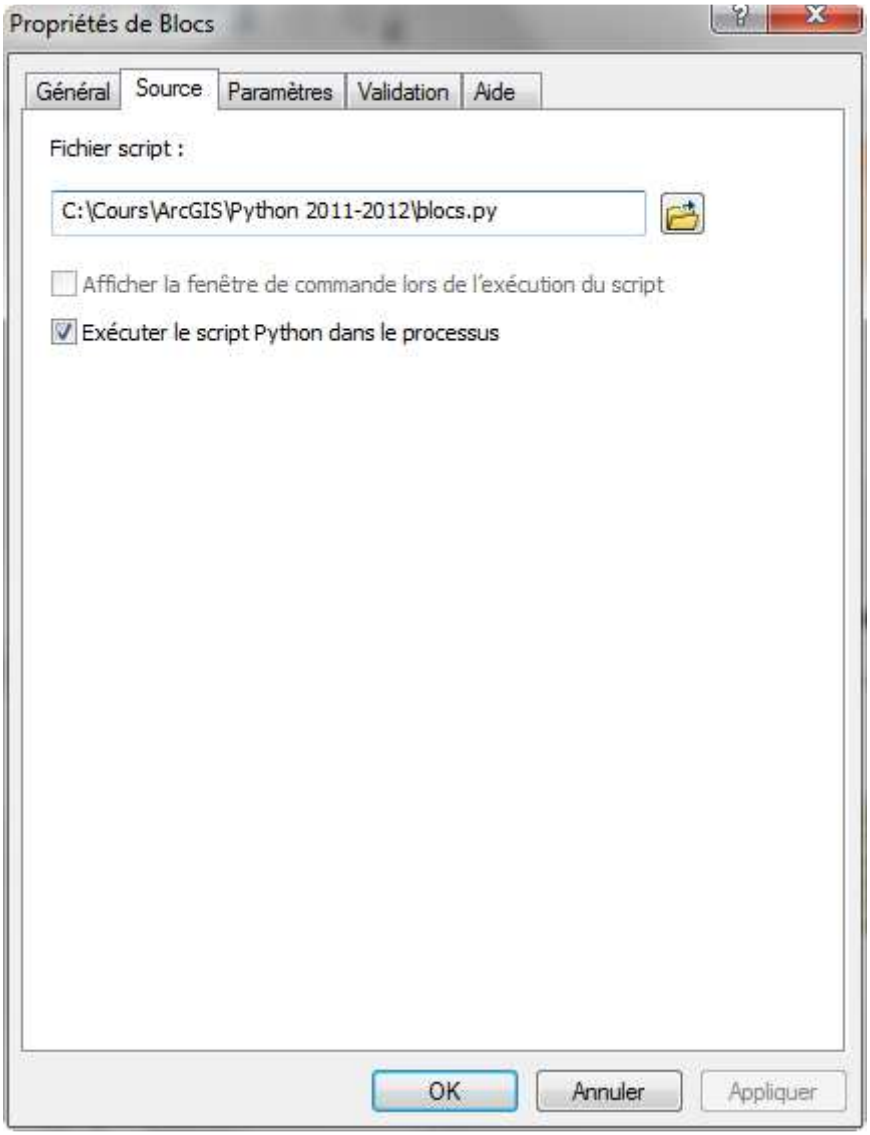
L'outil de script décrit dans cette fenêtre est celui qui est décrit dans le paragraphe suivant. Le code est reproduit plus bas.

2.1.2.4. Comment accéder aux propriétés d'un outil de script ?

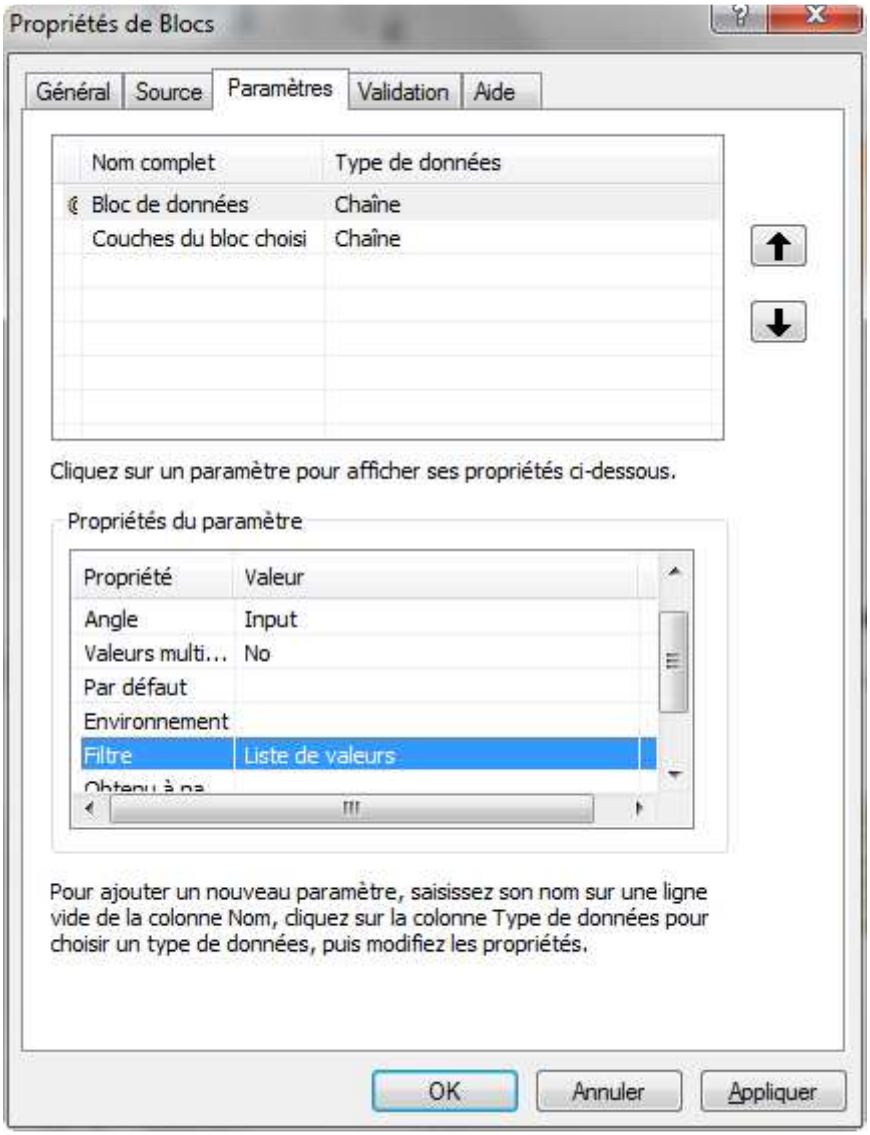
Dans le catalogue, par clic droit sur l'outil, Propriétés, on ouvre la boîte de dialogue suivante :



Sous l'onglet « Général », on définit en particulier le nom et la description de l'outil.
Cocher la case « Enregistrer les chemins relatifs » si l'outil (la boîte qui le contient) et son script se trouvent dans le même dossier.

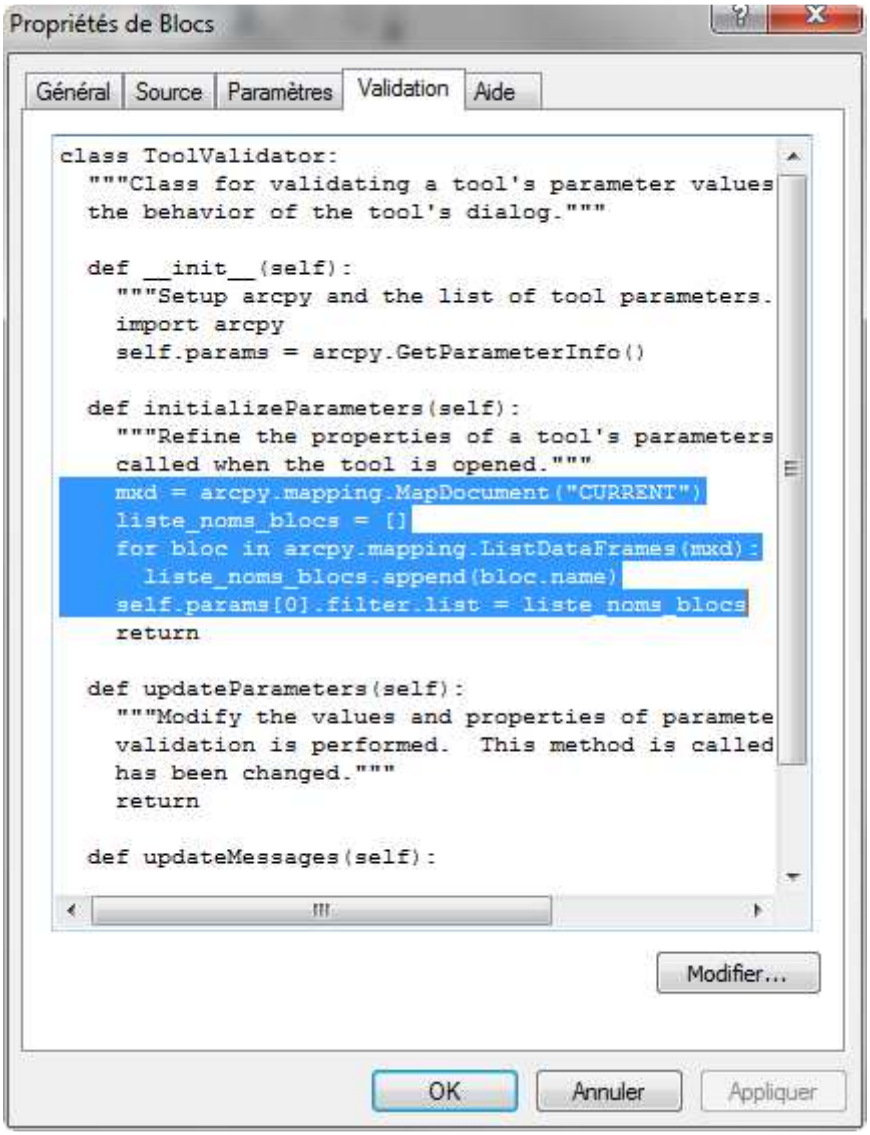


Sous l'onglet « Source » on précise le nom complet du script, avec son chemin d'accès.



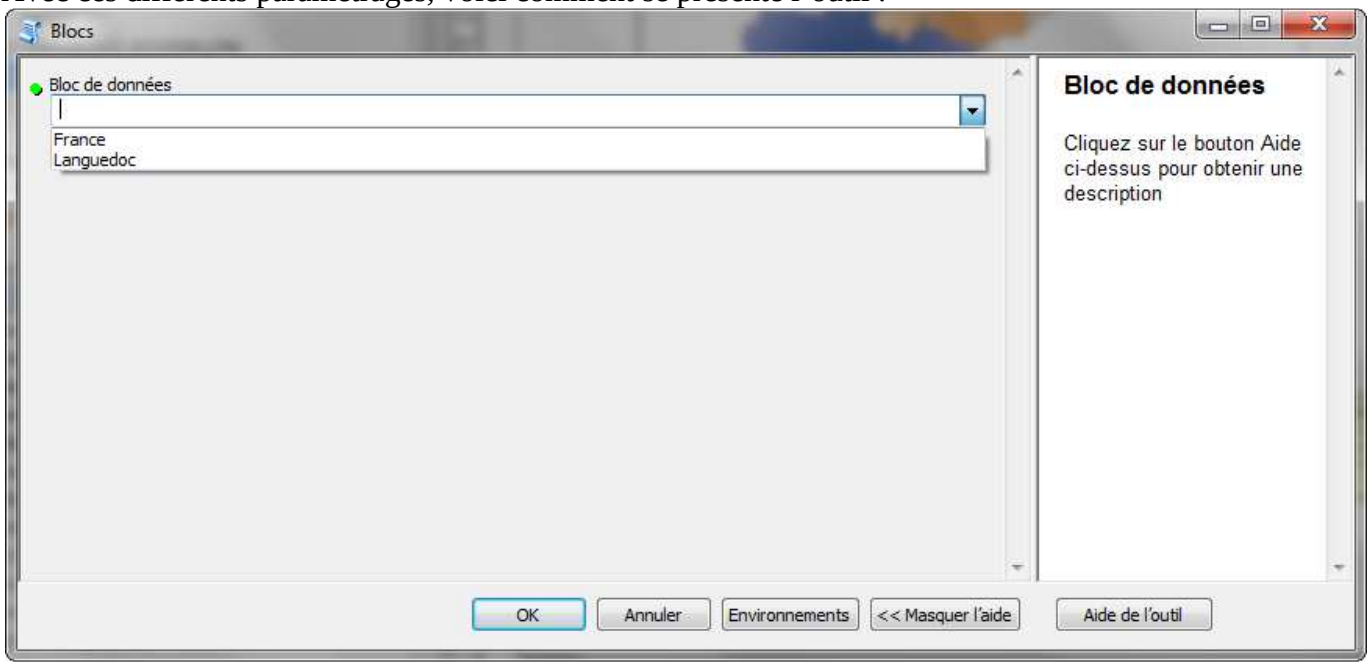
Sous l'onglet « Paramètres », on décrit les paramètres d'entrée et de sortie. Ici, le paramètre d'entrée est le nom d'un bloc de données du document actif. Il est de type Chaîne et d'angle Input. Il est filtré par la liste des noms des bloc de données du document actif. Le paramètre de sortie est la liste des noms des couches du bloc choisi. Il est du type Chaîne, à valeurs multiples.

Les paramètres d'un outil sont décrits par des objets de la classe arcpy.Parameter. La liste des paramètres d'un outil est retournée par la fonction arcpy.GetParameterInfo().



Sous l'onglet « Validation », on trouve la définition de la classe ToolValidator pour l'outil. Cette classe est définie par défaut mais on peut la modifier. Ici, la partie surlignée a été ajoutée pour filtrer le nom du bloc.

Avec ces différents paramétrages, voici comment se présente l'outil :



2.1.2.5. La classe ToolValidator d'un outil de script.

Elle est définie (et modifiable) sous l'onglet Validation de la fenêtre des propriétés de l'outil. Elle se présente comme un script Python. Elle comprend les méthodes suivantes :

__init__(self)	C'est une méthode d'initialisation
initializeParameters(self)	Elle est appelée à l'ouverture de l'outil
updateParameters(self)	Elle est appelée lors de toute modification d'un paramètre
updateMessages(self)	Elle est appelée lors de l'exécution de l'outil.

C'est dans cette classe que l'on gère la boîte de dialogue de l'outil.

On peut être amené à utiliser des propriétés et méthodes de la classe arcpy.Parameter.

Dans l'exemple suivant, il y a 4 paramètres : un bloc de données du document ArcMap courant, une couche de ce bloc, un champ de cette couche et le type de ce champ. L'activation et le filtrage des paramètres sont actualisés selon les choix de l'opérateur.

```
class ToolValidator:
  """Class for validating a tool's parameter values and controlling
  the behavior of the tool's dialog."""

  def __init__(self):
    """Setup arcpy and the list of tool parameters."""
    import arcpy
    self.params = arcpy.GetParameterInfo()
    global mxd
    mxd = arcpy.mapping.MapDocument("CURRENT") # Document
    global blocs
    blocs = arcpy.mapping.ListDataFrames(mxd) # Liste des blocs de données

  def initializeParameters(self):
    """Refine the properties of a tool's parameters.  This method is
    called when the tool is opened."""
    liste = []
    for bloc in blocs:
      liste.append(bloc.name)
    self.params[0].filter.list = liste
    self.params[1].enabled = False
    self.params[2].enabled = False
    self.params[3].enabled = False
    global nom0 # Nom du bloc
    nom0 = self.params[0].value
    global nom1 # Nom de la couche
    nom1 = self.params[1].value
    global nom2 # Nom du champ
    nom2 = self.params[2].value
    return

  def updateParameters(self):
    """Modify the values and properties of parameters before internal
    validation is performed.  This method is called whenever a parmater
    has been changed."""
    if self.params[0].value !=  nom0: #Le nom du bloc a changé
      global nom0 # Nom du bloc
      nom0 = self.params[0].value
      self.params[1].value = None
      self.params[1].enabled = False
      self.params[2].value = None
      self.params[2].enabled = False
      self.params[3].value = None
      self.params[3].enabled = False
      trouve = False # Vrai si le bloc est trouvé
      for bloc in blocs:
        if bloc.name == nom0:
          trouve = True
          break
```

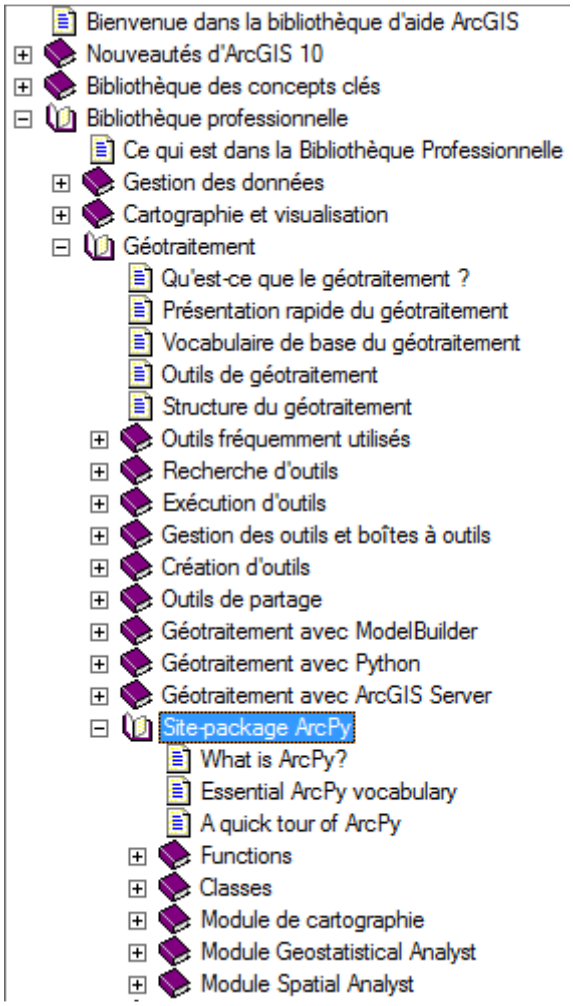
```
      if trouve:
        global couches
        couches = arcpy.mapping.ListLayers(mxd,"",bloc) # Liste des couches
        liste = []
        for couche in couches:
          liste.append(couche.name)
        self.params[1].filter.list = liste
        self.params[1].enabled = (len(liste) > 0)

    elif self.params[1].value != nom1: # Le nom de la couche a changé
      global nom1 #Nom de la couche
      nom1 = self.params[1].value
      self.params[2].value = None
      self.params[2].enabled = False
      self.params[3].value = None
      self.params[3].enabled = False
      trouvee = False # Vrai si la couche est trouvée
      for couche in couches:
        if couche.name == nom1:
          trouvee = True
          break
      if trouvee:
        if couche.isFeatureLayer:
          fc = couche.dataSource # FeatureClass associée à la couche
          global champs
          champs = arcpy.ListFields(fc) # Liste des champs
          liste = []
          for champ in champs:
            liste.append(champ.name)
          self.params[2].filter.list = liste
          self.params[2].enabled = True

    elif self.params[2].value != nom2: # Le nom du champ a changé
      global nom2 # Nom du champ
      nom2 = self.params[2].value
      self.params[3].value = None
      self.params[3].enabled = False
      trouve = False # Vrai si le champ est trouvé
      self.params[3].enabled = False
      for champ in champs:
        if champ.name == nom2:
          trouvee = True
          break
      if trouvee:
        self.params[3].filter.list = [champ.type]
        self.params[3].enabled = True
    return

  def updateMessages(self):
    """Modify the messages created by internal validation for each tool
    parameter.  This method is called after internal validation."""
    return
```

2.2. Le « site-package » ArcPy.

C’est un package pour ArcGIS permettant de traiter des données attributaires ou cartographiques (analyse, conversion, gestion) et d’automatiser des tâches de géotraitement. On y trouve des outils (ceux de l’Arc ToolBox et ceux que l’on aura ajoutés), des fonctions, des classes et des modules (qui peuvent être des packages). Pour l’utiliser dans un script on écrira :

```
import arcpy
```

Il est très bien documenté dans l’aide ArcGIS :



2.2.1. Les outils (outils de l'ArcToolBox ou nouveaux outils ajoutés).

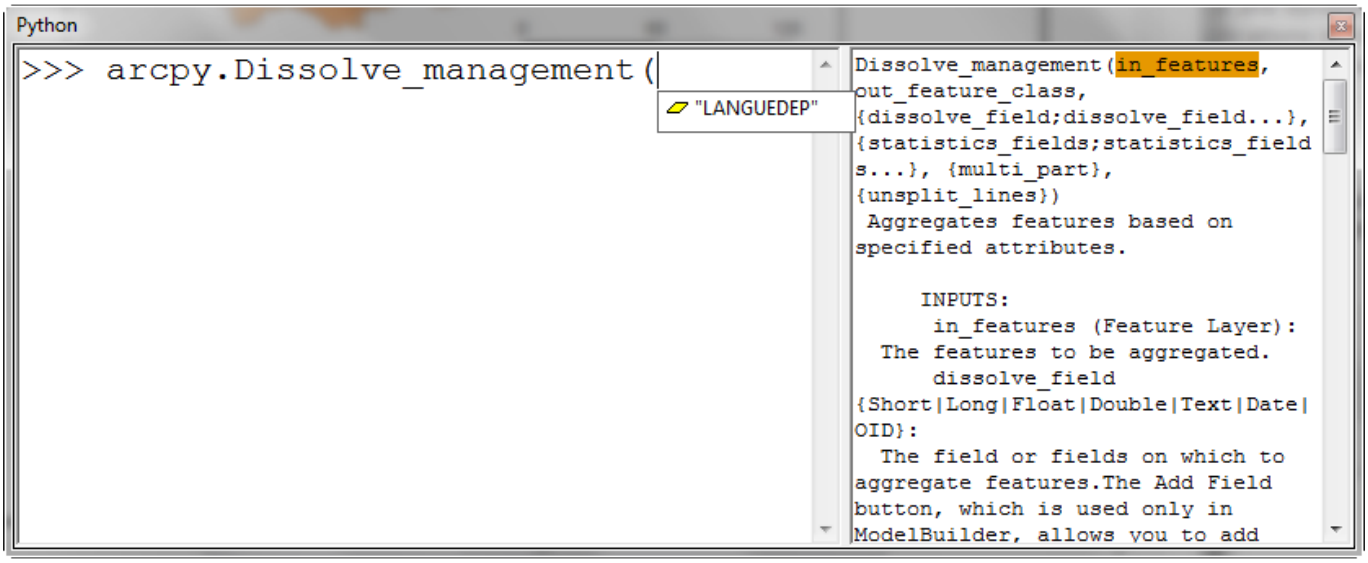
2.2.1.1. Intégration d'un outil à un script.

La syntaxe d'exécution est : arcpy.nom_outil(paramètres).
Si l'on souhaite récupérer des informations sur l'exécution de l'outil, et particulièrement des paramètres de sorties, il vaut mieux utiliser la syntaxe suivante : bilan = arcpy.nom_outil(paramètres). « bilan » est un objet de la classe arcpy.Result décrite plus bas.
Pour intégrer un outil de l'ArcToolbox à un script, on peut faire glisser l'outil depuis la fenêtre ArcToolBox dans la fenêtre Python. Dans la fenêtre Python, on bénéficie d'une aide et d'une assistance « complétion automatique ».
On peut ajouter une boîte à outils (que l'on a créée, par exemple) dans ArcPy par la fonction arcpy.ImportToolbox. Exemple :

```
>>> arcpy.ImportToolbox \
(r'C:\Cours\ArcGIS\Python\TP\TP7_Outil\Ma_boite.tbx',"Mesoutils")
```

Après cet ajout, les outils de la boîte « Ma_boite.tbx » sont disponibles dans la fenêtre Python. Ils sont préfixés par arcpy.Mesoutils. Ils bénéficient de l'aide et de l'auto complétion du code. Ils sont également disponibles dans tout script Python dans lequel on a importé le module ArcPy (par import arcpy).

Voici une capture de la fenêtre Python après ajout de l'outil Outils de gestion des données, Généralisation, Fusionner.



2.2.1.2. Récupérer les résultats de l'exécution d'un outil.

Les objets de la classe arcpy.Result décrivent le compte-rendu de l'exécution d'un outil. Pour récupérer ce compte-rendu dans un objet « bilan », on peut écrire : bilan = arcpy.nom_outil(paramètres).
Les propriétés d'un tel bilan sont en lecture seule.

Propriété	Commentaire	Type
inputCount	Nombre de paramètres en entrée	Integer
maxSeverity	Sévérité maximale des messages émis (voir Gestion des erreurs)	Integer
messageCoiunt	Nombre de messages émis	Integer
outputCount	Nombre de paramètres en sortie	Integer
resultID	Identifiant du travail	String
status	0 New 1 Submitted 2 Waiting 3 Executing 4 Succeeded 5 Failed 6 Timed out 7 Cancelling 8 Cancelled 9 Deleting 10 Deleted	Integer

Les méthodes suivantes sont disponibles.

Méthode	Effet
cancel()	Interrompt un travail associé
getInput(indice)	Retourne une entrée donnée, recordset ou string
getMapImageURL ({parameter_list}, {height}, {width}, {resolution})	Fournit un "map service image" pour une sortie donnée, s'il en existe.
getMessage (indice)	Retourne un message donné
getMessages ({severity})	Retourne une liste de messages.
getOutput (indice)	Retourne une sortie donnée, recorset ou string. Si la sortie de l'outil, tel que MakeFeatureLayer, est une couche, getOutput retournera un objet de la classe layer.
getSeverity (indice)	Retourne la sévérité d'un message donné.

2.2.1.3. La gestion des erreurs lors de l'exécution d'un outil.

On peut se reporter au paragraphe qui traite de la gestion des erreurs de manière générale.
Les outils retournent 3 types de messages :
Messages d'information (sévérité 0)
Messages d'avertissement (sévérité 1)
Messages d'erreur (sévérité 2)
Ces messages peuvent être retournés par les fonctions arcpy.GetMessage(Indice) et arcpy.GetMessages({Sévérité}). Ils peuvent également être retournés par les méthodes GetMessage(Indice) et GetMessages({Sévérité}) de la classe arcpy.Result.
Quand une erreur se produit lors de l'exécution d'un outil, il survient une « exception » arcpy.ExecuteError.
Quand un incident de « sévérité 1» se produit lors de l'exécution d'un outil, il survient une « exception » arcpy.ExecuteWarning.
Ces exceptions, comme toute autre exception Python, peuvent être traitées dans une clause except.
L'illustration qui suit est tirée d'un document de formation ESRI.

```
# Démarrer un bloc
try:
   arcpy.Buffer_analysis("c:/ws/roads.shp", "c:/outws/roads10.shp", 10)

# Si une erreur se produit lors de l'exécution de l'outil alors on affiche
# le message
except arcpy.ExecuteError:
   print arcpy.GetMessages(2)

# Si un autre type d'erreur se produit, on capture l'exception et on l'affiche
except Exception as e:
   print e.message
```

2.2.1.4. Le passage de paramètres dans un outil de script.

Pour passer les paramètres d'entrée, on utilise les fonctions arcpy.GetParameter() et arcpy.GetParameterAsText(). GetParameter retourne un objet ; GetParameterAsText retourne une chaîne de caractères. L'argument est l'entier 0 pour le 1er paramètre, l'entier 1 pour le second, etc.
Pour passer les paramètres de sortie, on utilise les fonctions arcpy.SetParameter() et arcpy.SetParameterAsText(). SetParameter retourne un objet ; SetParameterAsText retourne une chaîne de caractères.
A titre d'exemple, voici le code pour l'outil de script décrit plus haut :

```
# Outil de script
# Entrée : le nom d'un bloc
# Sortie : liste des noms des couches du bloc
import arcpy
nom_bloc = arcpy.GetParameterAsText(0) # Nom du bloc
mxd = arcpy.mapping.MapDocument("CURRENT") #Document
liste_noms_couches = [] # Future liste des noms de couches
blocs = arcpy.mapping.ListDataFrames(mxd) # Blocs de données du document
for bloc in blocs:
    if bloc.name == nom_bloc:
        couches = arcpy.mapping.ListLayers(mxd,"*",bloc) # couches du bloc
        for couche in couches:
            liste_noms_couches.append(couche.name)
        break
arcpy.SetParameter(1,liste_noms_couches)
del mxd, blocs, bloc, couches, couche
```

2.2.1.5. La classe arcpy.Parameter.

Les objets de la classe arcpy.Parameter décrivent les paramètres d'un outil de script.

Propriété	Commentaire (en anglais, tiré de l'aide ArcGIS)	Type
altered (Read Only)	True if the user has modified the value.	Boolean
category (Read and Write)	The category of the parameter.	String
datatype (Read Only)	Data type as defined on the Parameters tab of the tool's properties.	String
defaultEnvironmentName (Read Only)	Environment as defined on the Parameters tab of the tool's properties.	String
direction (Read Only)	Input/Output direction of the parameter as defined on the Parameters tab of the tool's properties.	String
displayName (Read Only)	The parameter name as shown on the tool's dialog box.	String
enabled (Read and Write)	False if the parameter is unavailable.	Boolean
filter (Read Only)	The filter to apply to values in the parameter.	Filter
hasBeenValidated (Read Only)	True if the internal validation routine has checked the parameter.	Boolean
name (Read Only)	Parameter name as defined on the Parameters tab of the tool's properties.	String
parameterDependencies (Read and Write)	A list of indexes of each dependent parameter.	Integer
parameterType (Read Only)	Type as defined on the Parameters tab of the tool's properties.	String
schema (Read Only)	The schema of the output dataset.	Schema
symbology (Read and Write)	The pathname to a layer file (.lyr) used for drawing the output.	String
value (Read and Write)	The value of the parameter.	Object

Méthode	Commentaire (en anglais, tiré de l'aide ArcGIS)
clearMessage ()	Clears out any message text and sets the status to informative (no error or warning).
hasError ()	Returns true if the parameter contains an error.
hasWarning ()	Returns True if the parameter contains a warning.
isInputValueDerived ()	Returns True if the tool is being validated inside a Model and the input value is the output of another tool in the model.
setErrorMessage (message)	Marks the parameter as having an error with the supplied message. Tools do not execute if any of the parameters have an error.
setIDMessage (message_type, message_ID, {add_argument1}, {add_argument2})	Allows you to set a system message.
setWarningMessage (message)	Marks the parameter as having a warning with the supplied message. Unlike errors, tools will execute with warning messages.

2.2.2. Les fonctions.

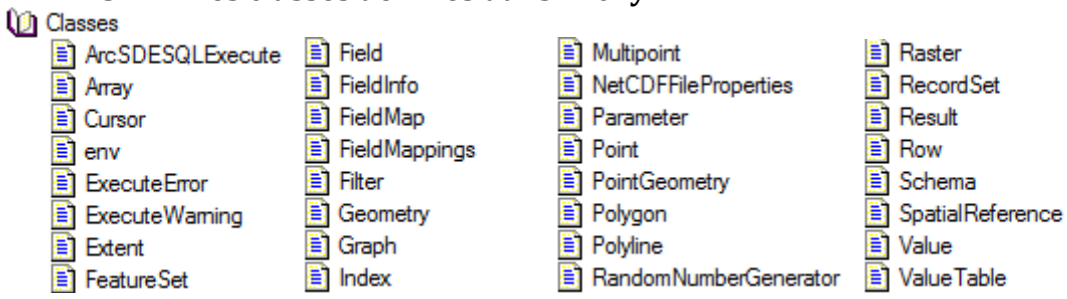
La syntaxe d'appel est : arcpy.nom_fonction(paramètres).
Voici quelques fonctions.

AddError	Ajoute un message d'erreur
AddMessage	Ajoute un message (récupérable par la fonction GetMessage)
AddReturnMessage	Ajoute un message retourné par un outil

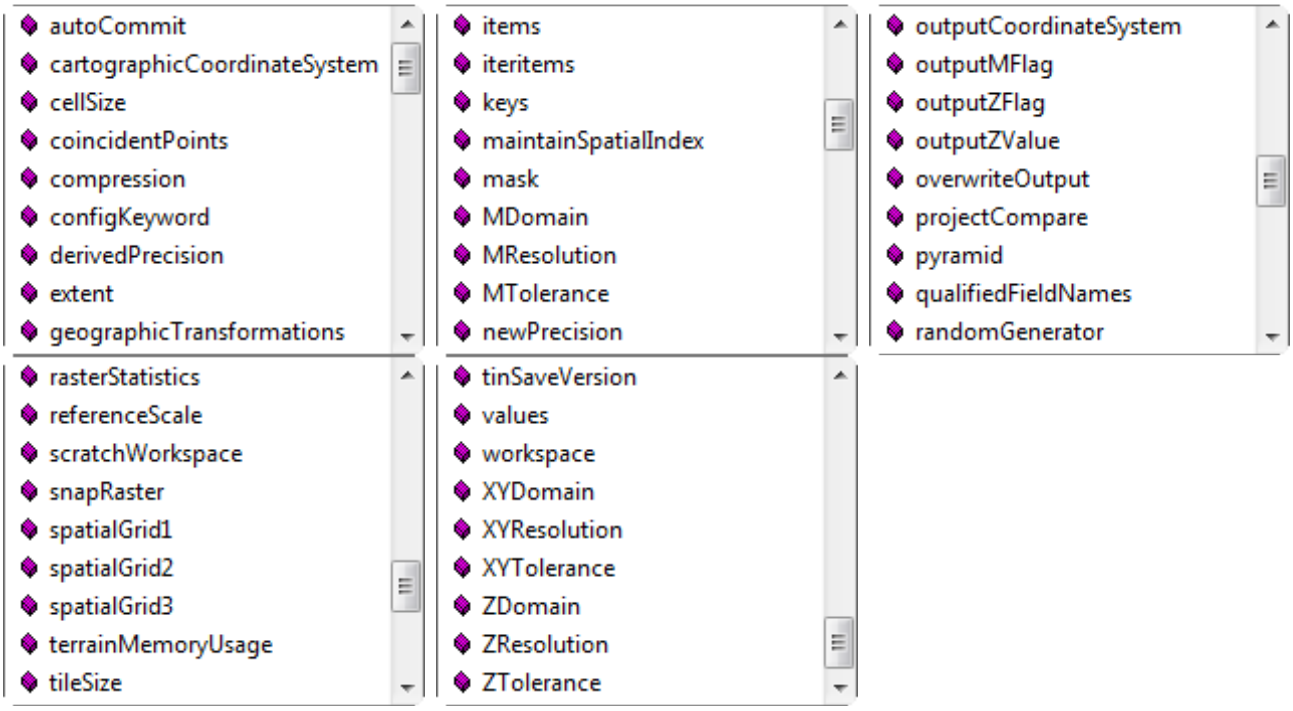
CopyParameter	Dans un outil de script, copie un paramètre d'entrée dans un paramètre de sortie
Describe	Retourne une description de l'argument
Exists	Teste la présence de l'argument dans l'espace de travail.
GetArgumentCount	Retourne le nombre d'arguments passés au script
GetMessage	Retourne un message (ajouté par un outil ou par la fonction AddMessage)
GetMessageCount	Retourne le nombre de messages
GetMessages	Retourne les messages séparés par un changement de ligne
GetParameter	Introduit un paramètre objet dans un outil de script.
GetParameterAsText	Introduit un paramètre chaîne de caractère dans un outil de script.
GetParameterCount	Retourne le nombre de paramètres d'un outil
ImportToolbox	Importe une boîte à outils dans ArcPy.
InsertCursor	Crée un curseur pour ajouter des lignes ou des entités
ListDatasets	Retourne la liste des ensembles de données de l'espace de travail actuel
ListFeatureClasses	Retourne la liste des classes d'entités de l'espace de travail actuel
ListFields	Retourne la liste des champs d'une table ou d'une classe d'entités
ListFiles	Retourne la liste des fichiers dans l'espace de travail actuel
ListRasters	Retourne la liste des rasters dans l'espace de travail actuel
ListTables	Retourne la liste des tables dans l'espace de travail actuel
ListToolboxes	Retourne la liste des boîtes à outils
ListTools	Retourne la liste des outils
ListWorkspaces	Retourne la liste des sous-espaces de travail de l'espace de travail actuel
RefreshActiveView	Rafraîchit la vue active du document actuel
RefreshTOC	Rafraîchit la table des matières (Table Of Contents)
RemoveToolbox	Supprime la boîte à outils spécifiée
SearchCursor	Crée un curseur pour parcourir des lignes ou des entités
SetParameter	Retourne un objet comme paramètre de sortie d'un script
SetParameterAsText	Retourne un texte comme paramètre de sortie d'un script
UpdateCursor	Crée un curseur pour mettre à jour ou supprimer des lignes ou des entités
Usage	Retourne la syntaxe de la fonction ou de l'outil spécifié

2.2.3. Les classes.

2.2.3.1. Les classes définies dans ArcPy.



La syntaxe pour créer un objet d'une classe donnée est : nom_objet = arcpy.nom_classe(paramètres).
Pour créer un objet vide, on omet les paramètres : nom_objet = arcpy.nom_classe().
Dans chaque classe, on dispose de propriétés en lecture et/ou en écriture et de méthodes. Voici par exemple la liste alphabétique des propriétés et des méthodes de la classe env, telle qu'elle apparaît dans la fenêtre Python lorsqu'on tape arcpy.env :



2.2.3.2. Le système de coordonnées (SpatialReference).

C'est un objet de la classe arcpy.SpatialReference. Il décrit la projection utilisée. On peut le définir à l'aide d'un fichier d'extension .prj. La syntaxe est :

arcpy.SpatialReference(Nom_du_fichier).

Le nom du fichier doit être complet.

Un objet SpatialReference peut être converti en chaîne de caractères par la méthode exportToString(). Inversement, il peut être reconstitué depuis une chaîne de caractères par la méthode loadFromString(chaine).

```
>>> sr = arcpy.SpatialReference("C:\Program Files (x86)\ArcGIS\Desktop10.0\Coordinate
Systems\Projected Coordinate Systems\National Grids\France\RGF 1993 Lambert-93.prj")
>>> sr.name
u'RGF_1993_Lambert_93'
>>> chaine = sr.exportToString()
>>> chaine
u"PROJCS['RGF_1993_Lambert_93',GEOGCS['GCS_RGF_1993',DATUM['D_RGF_1993',SPHEROID['GRS
_1980',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0],UNIT['Degree',0.017453292519
9433]],PROJECTION['Lambert_Conformal_Conic'],PARAMETER['False_Easting',700000.0],PARA
METER['False_Northing',6600000.0],PARAMETER['Central_Meridian',3.0],PARAMETER['Standa
rd_Parallel_1',44.0],PARAMETER['Standard_Parallel_2',49.0],PARAMETER['Latitude_Of_Ori
gin',46.5],UNIT['Meter',1.0]];-35597500 -23641900 10000;-100000 10000;-100000
10000;0,001;0,001;0,001;IsHighPrecision"
>>> sr2 = arcpy.SpatialReference()
>>> sr2.loadFromString(chaine)
>>> chaine2 = sr2.exportToString()
>>> chaine2 == chaine
True
>>> del sr,sr2,chaine, chaine2
```

Il y a une propriété spatialReference dans la classe arcpy.mapping.DataFrame et dans la description (par la fonction arcpy.Describe) d'une couche (layer), d'une classe d'entités (featureClass) et d'un jeu de données (dataSet).

```
>>> fc = r'C:\Cours\ArcGIS\Python \TP\TP1_Document\TP1 Document.mdb\DEPARTMT'
>>> descr = arcpy.Describe(fc)
>>> spref = descr.spatialReference
>>> spref.name
u'NTF_Lambert_II_Carto'
>>> del fc, descr, spref
```

2.2.3.3. L'étendue (Extent).

C'est un objet de la classe arcpy.Extent. Il décrit un rectangle défini par son coin inférieur gauche et son coin supérieur droit. Ce rectangle est souvent un rectangle d'emprise, c'est-à-dire le plus petit rectangle contenant un objet géométrique donné. La propriété extent d'un objet géométrique retourne son rectangle d'emprise. Ainsi, la propriété extent est disponible, par exemple, dans les classes arcpy.Geometry et arcpy.mapping.DataFrame.

Voici la syntaxe pour créer un objet Extent :

arcpy.Extent ({XMin}, {YMin}, {XMax}, {YMax}, {ZMin}, {ZMax}, {MMin}, {MMax})

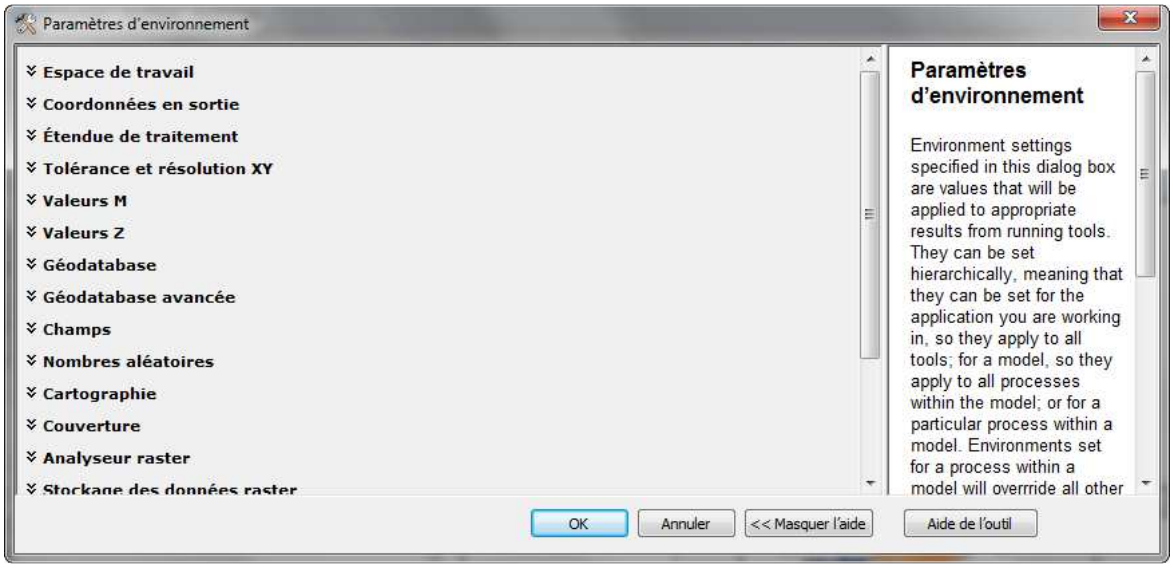
Voici le tableau des propriétés disponibles dans la classe acpy.Extent. Elles sont en lecture seule. Il y a cependant une exception : les objets Extent issus de la propriété extent de arcpy.mapping.DataFrame ont leurs propriétés modifiables.

Propriété	Commentaire en anglais (aide ArcGIS)	Type
MMax	The extent MMax value. None if no M value.	Double
MMin	The extent MMin value. None if no M value.	Double
XMax	The extent XMax value.	Double
XMin	The extent XMin value.	Double
YMax	The extent YMax value.	Double
YMin	The extent YMin value.	Double
ZMax	The extent ZMax value. None if no Z value.	Double
ZMin	The extent ZMin value. None if no Z value.	Double
depth	The extent depth value. None if no depth.	Double
height	The extent height value.	Double
lowerLeft	The lower left property: A point object is returned.	Point
lowerRight	The lower right property: A point object is returned.	Point
upperLeft	The upper left property: A point object is returned.	Point
upperRight	The upper right property: A point object is returned	Point
width	The extent width value.	Double

```
>>> emprise = arcpy.Extent(100,50,180,90)
>>> emprise.XMin,emprise.YMin,emprise.XMax,emprise.YMax
(100.0, 50.0, 180.0, 90.0)
```

2.2.3.4. Définition de l'environnement de géotraitement (env).

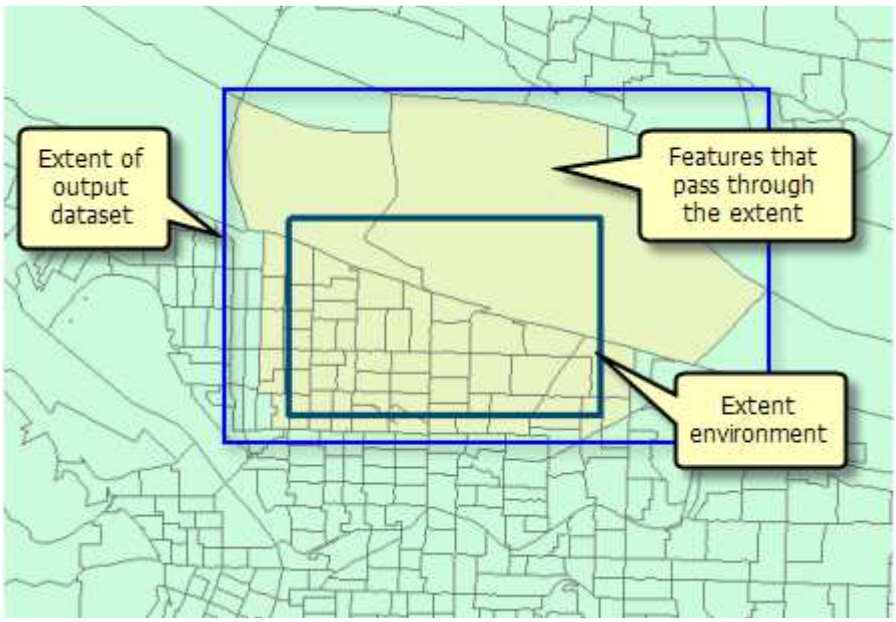
Les paramètres d'environnement de géotraitement sont les paramètres systèmes par défaut utilisés par chaque outil. On peut les définir manuellement par la boîte de dialogue Menu, Géotraitement, Environnements…



Ces paramètres sont accessibles par les propriétés de la classe arcpy.env. Voici les propriétés correspondant aux trois premiers paramètres.
L'espace de travail est défini par la propriété arcpy.env.workspace (en lecture et en écriture). C'est une chaîne de caractères. C'est le nom complet (avec son chemin d'accès) de l'espace de travail courant (dossier ou géodatabase).

```
>>> arcpy.env.workspace
u'C:\\Cours\\ArcGIS\\TP\\TP1\\TP1.gdb'
```

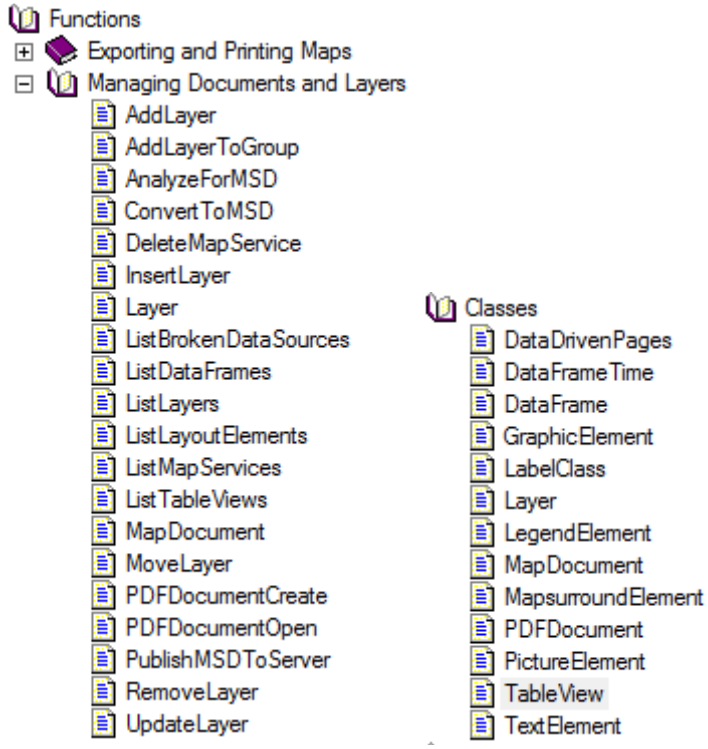
Les coordonnées en sortie sont définies par la propriété arcpy.env.outputCoordinateSystem (en lecture et en écriture). En écriture, on peut la définir par un nom d'extension .prj, par un nom de jeu de données ou par un objet de la classe arcpy.SpatialReference. En lecture, si elle est renseignée, elle retourne un objet de la classe arcpy.SpatialReference.
L'étendue de traitement est définie par la propriété arcpy.env.extent (en lecture et en écriture). En écriture, on peut la définir par un texte du type "xmin ymin xmax zmax". Le séparateur entre les nombres est l'espace. En lecture, si elle est renseignée, elle retourne un objet de la classe arcpy.extent. Un tel objet définit un rectangle d'emprise.




```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste_blocs = arcpy.mapping.ListDataFrames(mxd)
>>> bloc = liste_blocs[0]
>>> arcpy.env.extent = bloc.extent
>>> del mxd,liste_blocs,bloc
```

2.2.4. Les modules.

Il y a trois modules importants : les modules de cartographie (mapping), d'analyse géostatistique (ga) et d'analyse spatiale (sa). Nous présentons seulement le module de cartographie.
C'est le module arcpy.mapping. La syntaxe d'accès aux classes et aux fonctions de ce module est : arcpy.mapping.nom_classe_ou_fonction (paramètres).
Il permet de gérer les composants d'un document ArcMap : un document (MapDocument), un bloc de données (DataFrame), des étiquetttes (LabelClass), une couche (Layer), une table simple (TableView), une légende (LegendElement), un graphique (GraphicElement), une image (Picture Element), un texte (TextElement).
Cette liste n'est pas exhaustive.
Dans le module mapping, on trouve les fonctions et les classes suivantes.



2.3. Quelques illustrations.

2.3.1. Exploration d'un document ArcMap.

Un document ArcMap est un fichier d'extension .mxd.

2.3.1.1. Le document.

La référence à un document se fait par la fonction arcpy.mapping.MapDocument(nom_complet). On passe en paramètre une chaîne de caractères égale au nom complet du fichier, avec son chemin d'accès. Le plus simple est de faire glisser le document depuis le catalogue dans la fenêtre Python.

```
mxd = arcpy.mapping.MapDocument("C:/Cours/ArcGIS/TP/TP1/TP1a.mxd")
```

Pour accéder au document en cours, on remplace le nom complet du fichier par CURRENT.

```
mxd = arcpy.mapping.MapDocument("CURRENT")
```

La propriété filePath retourne le nom complet du fichier.

```
>>> mxd.filePath
u'C:\\Cours\\ArcGIS\\TP\\TP1\\TP1a.mxd'
```

La classe MapDocument contient en particulier la propriété title (en lecture et en écriture), la méthode save() et la méthode saveACopy (file_name, {version}).

2.3.1.2. Les blocs de données.

Ils sont appelés dataframes. Leur liste est retournée par la fonction arcpy.mapping.ListDataFrames(document) . Le paramètre est une référence au document.

```
>>> liste_blocs = arcpy.mapping.ListDataFrames(mxd)
>>> for bloc in liste_blocs:
...     print bloc.name
...
France

Languedoc
```

La classe DataFrame contient en particulier les propriétés extent (le rectangle d'emprise ; en lecture et en écriture), mapUnits(en lecture seule), name (en lecture et en écriture), scale (en lecture et en écriture),

spatialReference (en lecture et en écriture). La méthode zoomToSelectedFeatures () permet de zoomer sur les entités sélectionnées.

2.3.1.3. La vue active.

Il y a 2 modes d'affichage dans ArcMap : le mode Données et le mode Page. La propriété correspondante du document est activeView. C'est une chaîne de caractères. Elle est en lecture et en écriture.
En mode Page, activeView = "PAGE_LAYOUT".
En mode données, activeView est égale au nom du bloc de données actif. Pour activer un bloc de données, il suffit d'affecter son nom à la propriété activeView.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste_blocs = arcpy.mapping.ListDataFrames(mxd)
>>> premier_bloc = liste_blocs[0]
>>> nb_blocs = 0
>>> for i in liste_blocs:# Calcul du nombre de blocs
...     nb_blocs += 1
...
>>> dernier_bloc = liste_blocs[nb_blocs - 1]
>>> mxd.activeView = premier_bloc.name
>>> mxd.activeView = dernier_bloc.name
>>> mxd.activeView = "PAGE_LAYOUT"
```

La fonction arcpy.RefreshActiveView() rafraîchit la vue active.

2.3.1.4. Les couches.

Elles sont appelées layers. Leur liste est retournée par la fonction arcpy.mapping.ListLayers(document, {filtre}, {dataframe}) . Le filtre (facultatif) est une chaîne de caractères à retrouver dans le nom des couches. L'étoile (*) est le caractère générique. Si dataframe n'est pas précisé, la fonction retourne la liste éventuellement filtrée de toutes les couches du document.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste_blocs = arcpy.mapping.ListDataFrames(mxd)
>>> premier_bloc = liste_blocs[0]
>>> liste_couches_0 = arcpy.mapping.ListLayers(mxd,"*",premier_bloc)
>>> for couche in liste_couches_0:
...     print couche.name
...
REGIONS
>>> liste_couches = arcpy.mapping.ListLayers(mxd)
>>> for couche in liste_couches:
...     print couche.name
...
REGIONS

LANGUEDEP
```

Voici quelques propriétés de la classe layer.

Propriété	Lecture/écriture	Type	Commentaire
dataSetName	Lecture seule	String	Nom simple du fichier source
dataSource	Lecture seule	String	Nom complet du fichier source
isFeatureLayer	Lecture seule	Boolean	True si la couche est une couche d'entités
isRasterLayer	Lecture seule	Boolean	True si la couche est une couche raster
name	Lecture/écriture	String	Nom
showLabels	Lecture/écriture	Boolean	Etiquettes visibles ?
transparency	Lecture/écriture	Integer	Transparence (de 0 à 100)
visible	Lecture/écriture	Boolean	Couche visible ?
workspacePath	Lecture seule	String	« Espace de travail » du fichier source (chemin d'accès ou nom complet de la géodatabase)

Voici quelques méthodes de la classe layer.

Méthode	Commentaire
save()	Enregistre dans un fichier .lyr
saveACopy (file_name, {version})	Enregistre un copie dans un fichier .lyr
supports (layer_property)	Retourne un booléen.

2.3.1.5. Les tables.

Leur liste est retournée par la fonction arcpy.mapping.ListTableViews(document, {filtre}, {dataframe}) . Le filtre (facultatif) est une chaîne de caractères à retrouver dans le nom des couches. (*) est le caractère générique. Si dataframe n'est pas précisé, la fonction retourne la liste éventuellement filtrée de toutes les tables du document.

```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste_tables = arcpy.mapping.ListTableViews(mxd)
>>> for table in liste_tables:
...     print table.name
...
Ma_table
```

La classe tableView possède en particulier les propriétés datasetName, dataSource, name et workspacePath, comme la classe layer.

2.3.1.6. Les éléments graphiques de la vue active.

Leur liste est retournée par la fonction arcpy.mapping.ListLayoutElements (document, {element_type}, {filtre}). Les types d'éléments sont les chaînes de caractères suivantes : "DATAFRAME_ELEMENT", " GRAPHIC_ELEMENT", "LEGEND_ELEMENT", "MAPSURROUND_ELEMENT", " PICTURE_ELEMENT", " TEXT_ELEMENT". Si le type d'élément est omis, la liste, éventuellement fitrée, est complète.

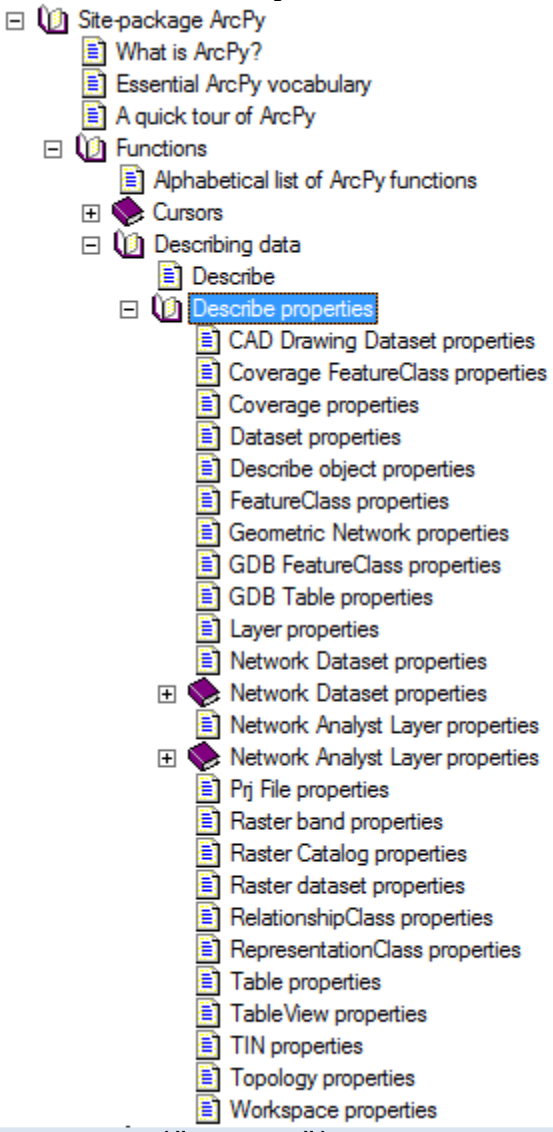
```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste = arcpy.mapping.ListLayoutElements(mxd)
>>> len(liste)
12
>>> for elt in liste:
...     print elt.type
...
GRAPHIC_ELEMENT

GRAPHIC_ELEMENT

MAPSURROUND_ELEMENT

MAPSURROUND_ELEMENT

TEXT_ELEMENT

TEXT_ELEMENT

GRAPHIC_ELEMENT

TEXT_ELEMENT

TEXT_ELEMENT

TEXT_ELEMENT

DATAFRAME_ELEMENT

DATAFRAME_ELEMENT
```

2.3.1.7. La fonction arcpy.Describe.

La syntaxe est : arcpy.Describe(Nom_élément). La fonction retourne un objet dont les propriétés (en lecture seule) décrivent l'élément. Il y a des propriétés toujours disponibles :

Propriété	Commentaire en anglais (aide ArcGIS)	Type de la propriété
baseName	The file base name	String
catalogPath	The path of the data	String
children	A list of sub elements	List
childrenExpanded	Indicates whether the children have been expanded	Boolean
dataType	The type of the element	String
extension	The file extension	String
file	The file name	String
fullPropsRetrieved	Indicates whether full properties have been retrieved	Boolean
metadataRetrieved	Indicates whether the metadata has been retrieved	Boolean
name	The user-assigned name for the element	String
path	The file path	String

Il y a également des propriétés spécifiques selon la nature de l'objet décrit. Elles sont détaillées dans l'aide ArcGIS. Malheureusement, l'« auto-code-completion » reste muette.




```
>>> mxd = arcpy.mapping.MapDocument("CURRENT")
>>> liste_couches = arcpy.mapping.ListLayers(mxd)
>>> Couche = liste_couches[0]
>>> descr = arcpy.Describe(Couche)
>>> descr.spatialReference.name, descr.shapeType
u'NTF_Lambert_II_Carto'   u'Polygon'
```

2.3.2. Exploration d'un espace de travail (workspace).

L'espace de travail est une chaîne de caractères. C'est le nom complet de la géodatabase ou du dossier contenant les jeux de données. On peut le lire ou le définir par la propriété workspace de la classe arcpy.env.

```
>>> arcpy.env.workspace="C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
```

Pour obtenir ce nom complet, on peut faire glisser le dossier ou la géodatabase depuis le catalogue dans la fenêtre Python.
L'espace de travail définit l'emplacement des données et l'emplacement où toutes les nouvelles données seront créées, sauf si un chemin complet est spécifié.
L'espace de travail étant choisi, on peut utiliser les fonctions de listes prévues dans ArcPy :

ListFiles({filtre})

```
>>> arcpy.env.workspace = "C:/Cours/ArcGIS/TP/TP1"
>>> arcpy.ListFiles()
[u'Aude.jgw', u'Aude.jpg', u'Aude.jpg.aux.xml', u'Ma_table.xlsx', u'tp1.doc',
u'TP1.gdb', u'TP1a.emf', u'TP1a.mxd']
>>> arcpy.ListFiles("*.mxd")
[u'TP1a.mxd']
```

ListDatasets({filtre},{ type_entité})
Retourne une liste de noms de « jeux de données ». Les rasters sont des « jeux de données ». Les tables, les classes d'entités, les géodatabases fichiers n'en sont pas. Le « type_entité » pour un raster est "Raster".

```
>>> arcpy.ListDatasets()
[u'Aude.jpg']
>>> arcpy.ListDatasets("*","Raster")
[u'Aude.jpg']
```

ListFeatureClasses({filtre},{ type_entité})
Retourne une liste de noms de classes d'entités. Les principaux types d'entités sont « Point », « Line » et « Polygon ».

```
>>> arcpy.env.workspace="C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
>>> arcpy.ListFeatureClasses("*","Point")
[u'Mes_points']
>>> arcpy.ListFeatureClasses("*","Line")
[u'Mes_lignes', u'Mes_lignes_2']
>>> arcpy.ListFeatureClasses("*","Polygon")
[u'REGIONS', u'LANGUEDEP', u'com11l2p', u'COM_CAN', u'Ma_classe', u'com11l2p_Clip',
u'com11l2p_Intersect', u'com11l2p_Union', u'Ma_classe_Union']
>>> arcpy.ListFeatureClasses("com*")
[u'com11l2p', u'COM_CAN', u'com11l2p_Clip', u'com11l2p_Intersect', u'com11l2p_Union']
>>> arcpy.ListFeatureClasses()
[u'REGIONS', u'LANGUEDEP', u'com11l2p', u'COM_CAN', u'Ma_classe', u'Mes_points',
u'Mes_lignes', u'Mes_lignes_2', u'com11l2p_Clip', u'com11l2p_Intersect',
u'com11l2p_Union', u'Ma_classe_Union']
```

ListRasters({filtre},{ type_raster})
Retourne une liste de noms de rasters. Les principaux types de rasters sont "BMP", "GIF", "IMG", "JPG", "JP2", "TIFF", "GRID".

```
>>> arcpy.env.workspace="C:\Cours\ArcGIS\TP\TP1"
>>> arcpy.ListRasters()
[u'Aude.jpg']
>>> arcpy.ListRasters("*","JPG")
[u'Aude.jpg']
```

ListTables({filtre},{type_table})
Retourne une liste de noms de tables. Les principaux types de tables sont "dBASE" et "INFO". Les tables Excel, pourtant exploitables par ArcMap, ne sont pas reconnues.

```
>>> arcpy.ListTables()
[u'Ma_table.dbf']
>>> arcpy.ListTables("*","dBASE")
[u'Ma_table.dbf']
```

ListFields(table ou classe d'entités, {filtre}, {type_de_champ})
Retourne une liste de champs. Les principaux types de champs sont "OID", "Geometry", "Date", "Single", "Double", "Integer","SmallInteger", "String".
Pour renseigner le 1er argument (table ou classe d'entités), le plus simple est de faire glisser la table ou la classe d'entités depuis le catalogue dans la fenêtre Python.
Attention : c'est un liste d'objets champ qui est retournée ; pas une liste de noms de champs.

```
>>> arcpy.env.workspace="C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
>>> liste_champs = arcpy.ListFields("REGIONS")
>>> for champ in liste_champs:
...     print champ.name, champ.type
...
OBJECTID   OID

Shape   Geometry

CODE   String

NOM   String

SUP_KM2   Double

POP90   Double

Shape_Length   Double

Shape_Area   Double
```

2.3.3. Parcours d'une table ou d'une classe d'entités. Les curseurs.

Un curseur est un objet de la classe arcpy.cursor. Il permet de parcourir une table ou une classe d'entités. Il y a trois catégories de curseurs : search (pour une lecture), insert (pour une insertion) et update (pour une modification ou une suppression).
Un curseur supporte l'instruction « for *ligne* in *curseur*: » pour un parcours des lignes correspondantes.
Le tableau suivant décrit les méthodes de la classe arcpy.cursor. L'argument « ligne » peut être une ligne d'une table simple ou une entité d'une classe d'entités.

Méthode	Effet
deleteRow(ligne)	Supprime la ligne ou l'entité.
insertRow(ligne)	Insère la ligne ou l'entité
newRow()	Crée une ligne vide ou une entité vide. Il faudra ensuite l'insérer.
next()	Retourne la ligne suivante ou l'entité suivante.
reset()	Le prochain next() retournera la 1ère ligne ou la 1ère entité
updateRow(ligne)	Remplace la ligne ou l'entité actuelle par la ligne ou l'entité passée en argument.

Le tableau suivant décrit les méthodes de la classe arcpy.row.

Méthode	Effet
getValue(nom_du_champ)	Retourne la valeur du champ
isNull(nom_du_champ)	La valeur du champ est « null »
setNull(nom_du_champ)	le champ prend la valeur « null »
setValue(nom_du_champ, objet)	Le champ prend la valeur référencée par le paramètre « objet »

Raccourcis :
ligne.getValue(nom_du_champ) peut être remplacé par ligne.nom_du_champ
ligne.setValue(nom_du_champ, objet) peut être remplacé par ligne.nom_du_champ = objet

La fonction arcpy.SearchCursor(jeu_de_données,{where_clause},{référence_spatiale},{champs},{champs_de_tri}) retourne un curseur de lecture.
« jeu_de_données » est un nom de table ou de classe d'entités. On peut l'obtenir en faisant glisser la table ou la classe d'entités depuis le catalogue dans la fenêtre Python.
« where_clause » est une clause de filtrage SQL. C'est une chaîne de caractères. Pour l'obtenir, on peut construire une clause analogue dans Menu, Sélection, Sélectionner par attribut.
« référence_spatiale » est la référence spatiale de la classe d'entités.
« champs » est une chaîne de caractères constituée des noms des champs à traiter ; les noms sont séparés par des points-virgules (;) ; par défaut, tous les champs sont pris.
« champs de tri » est une chaîne de caractères constituée des noms des champs à traiter ; les noms sont séparés par des points-virgules (;) ; le nom d'un champ à tri croissant est suivi de la lettre A (comme « Ascending ») ; le nom d'un champ à tri décroissant est suivi de la lettre D (comme « Descending »).

```
>>> arcpy.env.workspace = "C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
>>> curseur = arcpy.SearchCursor("com11l2p")
>>> nb, pop = 0, 0
>>> for ligne in curseur:
...     nb += 1
...     pop += ligne.POP
...
>>> print "Nombre de communes dans l'Aude :",nb
Nombre de communes dans l'Aude :   438

>>> print "Population de l'Aude :",pop, "habitants"
Population de l'Aude :   298898.0   habitants

>>> del curseur, ligne
```

```
>>> arcpy.env.workspace = "C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
>>> curseur = arcpy.SearchCursor("com11l2p","",None,"NOM ; POP","POP D")
>>> ligne = curseur.next()
>>> nom = ligne.getValue("NOM")
>>> population = ligne.POP
>>> print "Commune la plus peuplée de l'Aude :",nom,population,"hbts"
Commune la plus peuplée de l'Aude :   NARBONNE   45866.0   hbts

>>> del ligne, curseur
```

La fonction arcpy.UpdateCursor(jeu_de_données,{where_clause},{référence_spatiale},{champs},{champs_de_tri}) retourne un curseur de mise à jour ou de suppression. Les arguments sont les mêmes que ceux de arcpy.SearchCursor.

```
>>> curseur = arcpy.UpdateCursor("com11l2p","\"NOM\" = 'CARCASSONNE'",None,"POP")
>>> ligne = curseur.next()
>>> ligne.POP += 100 # On ajoute 100 habitants à Carcassonne.
>>> curseur.updateRow(ligne)
>>> del curseur, ligne
```

La fonction arcpy.InsertCursor(jeu_de_données,{référence_spatiale}) retourne un curseur d'insertion.
Pour ajouter une ligne (ou une entité), il faut :

créer une nouvelle ligne (ou entité) par la méthode newRow
renseigner les champs de la ligne (ou de l'entité)
ajouter la ligne (ou l'entité) par la méthode insertRow. L'ajout se fait en fin du jeu de données.

```
>>> curseur = arcpy.InsertCursor("Ma_table.dbf")
>>> ligne = curseur.newRow()
>>> ligne.Nom = "Nouveau_Nom"
>>> ligne.Prénom = "Nouveau_Prénom"
>>> curseur.insertRow(ligne)
>>> curseur = arcpy.UpdateCursor("Ma_table.dbf", "\"Nom\" = 'Nouveau_Nom'")
>>> ligne = curseur.next()
>>> curseur.deleteRow(ligne)
>>> del curseur, ligne
```

2.3.4. Extraction de données.

Le principal outil pour extraire des données choisies selon un critère attributaire est :

ArcToolBox > Outils d'analyse > Extraire > Sélectionner.

On peut bien sûr l'utiliser directement. On peut aussi l'inclure dans un script en le faisant glisser dans la fenêtre Python. Dans l'exemple suivant, on extrait les communes audoises plus grandes que la commune de Talairan. Les aires sont dans le champ « Area ».

Exemple 1 :

```
>>> arcpy.env.workspace = r'C:\Cours\ArcGIS\Python 2011-2012\TP\Ressources'
>>> fc = 'com11l2p.shp'
>>> curseur = arcpy.SearchCursor(fc,"\"NOM\" = 'TALAIRAN'")
>>> ligne = curseur.next()
>>> aire = str(ligne.Area)
>>> arcpy.Select_analysis(fc,"gdes_coms","\"Area\" > " + aire)
>>> del ligne, curseur
```

Pour extraire des données selon un critère spatial, il faut procéder autrement. La méthode exposée ci-dessous convient aussi pour extraire des données selon un critère attributaire. On utilise les outils :

Outils de gestion des données > Couches et vues tabulaires > Générer une couche

Outils de gestion des données > Couches et vues tabulaires > Sélectionner une couche par attributs

Outils de gestion des données > Couches et vues tabulaires > Sélectionner une couche par emplacement

Outils de gestion des données > Entités > Copier des entités

On peut appliquer la méthode suivante.

1. Générer une couche à partir de la classe d'entités source : arcpy.MakeFeatureLayer_management
2. Sélectionner dans cette couche les entités à extraire :
arcpy.SelectLayerByAttribute_management ou bien arcpy.SelectLayerByLocation_management
3. Copier dans la classe cible les entités sélectionnées : arcpy.CopyFeatures_management

Exemple2 : extraction, parmi les communes du Loir-et-Cher, des communes de l'arrondissement de Blois (de code-arrondissement égal à 1).

```
>>> arcpy.env.workspace = "C:/Python 2011-2012/TP/TP4_Extraction/TP4_Extraction.mdb"
>>> source = "C:/Python 2011-2012/TP/TP4_Extraction/Com41.shp"
>>> couche = "Com41"
>>> whereClause = " \"CODE_ARR\" = '1' "
>>> cible = "Com_Blois"
>>> arcpy.MakeFeatureLayer_management(source, couche)
>>> arcpy.SelectLayerByAttribute_management(couche, "NEW_SELECTION", whereClause)
>>> arcpy.CopyFeatures_management(couche, cible)
```

Exemple 3 : extraction, parmi les villes du Loir-et-Cher, des villes de l'arrondissement de Blois (situées dans les communes de cet arrondissement).

```
>>> arcpy.env.workspace = "C:/Python 2011-2012/TP/TP4_Extraction/TP4_Extraction.mdb"
>>> source = "Villes41"
>>> couche = "Villes41_"
>>> selecteur = ″Com_Blois″
>>> distance = 0
>>> cible = ″Villes_Blois″
>>> arcpy.MakeFeatureLayer_management(source, couche)
>>> arcpy.SelectLayerByLocation_management(couche,″WITHIN″, selecteur, distance,
"NEW_SELECTION")
>>> arcpy.CopyFeatures_management(couche, cible)
```

Exemple 4 : obtention de la liste des géométries d'une classe d'entités.

Cette liste est retournée par l'outil : arcpy.CopyFeatures_management quand on choisit pour cible un objet de la classe arcpy.Geometry. Toute liste non vide de géométries peut être utilisée comme source de chacun des outils arcpy.MakeFeatureLayer_management et arcpy.CopyFeatures_management.

```
>>> source = "C:/Python 2011-2012/TP/TP4_Extraction/Com41.shp"
>>> copie = "C:/Python 2011-2012/TP/TP4_Extraction/copie_Com41.shp"
>>> cible = arcpy.Geometry()
>>> liste_geoms = arcpy.CopyFeatures_management(source, cible)
>>> arcpy.CopyFeatures_management(liste_geoms, copie)
>>> couche = "Com41_"
>>> arcpy.MakeFeatureLayer_management(liste_geoms, couche)
```

Dans la copie, il n'y a que la géométrie ; pas les attributs.

Exemple 5 : script d'un outil de script pour générer la classe des centroïdes d'une classe d'entités donnée. En paramètres, on donne le nom complet de la classe d'entités source et le nom complet de la classe des centroïdes. Attention ! Pour que la classe d'entités cible ait une référence spatiale, il importe d'attribuer cette référence spatiale aux items de la liste de géométries qui alimentera l'outil CopyFeature_Management.

```
import arcpy
source = arcpy.GetParameterAsText(0)
spatialref = arcpy.Describe(source).SpatialReference
liste_pts = []
curseur = arcpy.SearchCursor(source)
for ligne in curseur:
    pt = arcpy.PointGeometry(ligne.Shape.centroid,spatialref)
    liste_pts.append(pt)
cible = arcpy.GetParameterAsText(1)
arcpy.CopyFeatures_management(liste_pts,cible)
```

2.3.5. Exploration d'un objet géométrique (point, ligne, polygone).

2.3.5.1. La propriété shapeType.

Lorsque la fonction arcpy.Describe() est appliquée à une couche ou à une classe d'entités, elle retourne un objet qui possède la propriété shapeType. Cette propriété retourne le type de la géométrie : Point, Polyline ou Polygon.

```
>>> arcpy.env.workspace = r'C:\Cours\ArcGIS\TP\TP1_Document\TP1 Document.mdb'
>>> fcs = arcpy.ListFeatureClasses()
>>> for fc in fcs:
...     descr = arcpy.Describe(fc)
...     print fc,descr.shapeType
...
VILLE_10   Point
REGIONS   Polygon
DEPARTMT   Polygon
LANGDEP   Polygon
autor11l2p   Polyline
can11l2p   Polygon
com11l2p   Polygon
>>> del fcs, fc, descr
```

2.3.5.2. La classe arcpy.Point.

Les objets de cette classe décrivent les points « purs ». Ils sont définis par leurs coordonnées. Ils sont dépourvus, par exemple, de référence spatiale. Les points en tant qu'entités relèvent plutôt de la classe arcpy.PointGeometry.

Syntaxe : arcpy.Point ({X}, {Y}, {Z}, {M}, {ID})

Paramètre	Signification	Type	Valeur par défaut
X	Abscisse	Double	0.0
Y	Ordonnée	Double	0.0
Z	3ème coordonnée	Double	None

M	Valeur M	Double	None
ID	Identifiant	Integer	0

```
>>> pt = arcpy.Point(3,5)
>>> print pt.X, pt.Y
3.0   5.0
```

2.3.5.3. La classe arcpy.Array.

Toute géométrie peut être considérée comme une collection de points. Les objets de la classe arcpy.Array décrivent les collections de points.
Syntaxe : Array ({items})
Les items peuvent être des listes, des points ou d'autres objets Array.
Unique Propriété : count. Elle retourne le nombre d'items.
Méthodes :

Méthode	Effet
add (value)	Ajoute un objet Array ou Point dans le tableau en dernière position.
append (value)	Ajoute un objet dans le tableau en dernière position.
clone (point_object)	Clone l'objet point.
extend (items)	Etend le tableau en ajoutant des éléments.
getObject (index)	Retourne un objet spécifique du tableau.
insert (index, value)	Ajoute un objet dans le tableau à une position spécifique.
next ()	Retourne l'objet suivant dans le tableau.
remove (index)	Supprime un objet spécifique du tableau.
removeAll ()	Supprime tous les objets et crée un tableau vide.
replace (index, value)	Remplace un objet à l'aide de la position d'index.
reset ()	Réinitialise le tableau au premier objet.

Un objet arcpy.Array peut être parcouru par une boucle for.

```
>>> liste = [[1,2],[3,4],[5,6]]
>>> pt = arcpy.Point()
>>> tableau = arcpy.Array()
>>> for paire in liste:
...     pt.X = paire[0]
...     pt.Y = paire[1]
...     tableau.add(pt)
...
>>> tableau.count
3
>>> for pt in tableau:
...     print pt.X,pt.Y
...
1.0   2.0
3.0   4.0
5.0   6.0
>>> del liste, pt, tableau
```

2.3.5.4. Les classes arcpy Geometry, PointGeometry, Multipoint, Polygon, Polyline.

Syntaxe : arcpy.***nom_classe*** (inputs, {spatialReference}, {hasZ}, {hasM})

Paramètre	Signification	Type	Valeur par défaut
inputs	Point ou tableau	Point ou Array	
spatialReference	Référence spatiale	SpatialReference	None
hasZ	Possède coordonnée Z	Boolean	False
hasM	Possède une valeur M	Boolean	False

Les propriétés sont en lecture seule.

Propriété	Explication en anglais (aide ArcGIS)	Type
area	The area of the geometry	Double
centroid	The true centroid if it is within or on the feature; otherwise, the label point is returned.	Point
extent	The extent of the geometry.	Extent
firstPoint	The first coordinate point of the geometry.	Point
hullRectangle	A space-delimited string of the coordinate pairs of the convex hull rectangle.	String
isMultipart	True, if the number of parts for this geometry is more than one.	Boolean
labelPoint	The point at which the label is located. The labelPoint is always located within or on a feature.	Point
lastPoint	The last coordinate of the feature.	Point
length	The length of the linear feature. Zero for point, multipoint feature types.	Double
partCount	The number of geometry parts for the feature.	Integer
pointCount	The total number of points for the feature.	Integer
trueCentroid	The center of gravity for a feature.	Point
type	The geometry type: polygon, polyline, point, multipoint, multipatch, dimension, annotation.	String

Méthodes.

Méthode	Explication en anglais, tirée de l'aide ArcGIS
contains (second_geometry)	Indicates if this geometry contains the other geometry.
crosses (second_geometry)	Indicates if the two geometries intersect in a geometry of lesser dimension.
disjoint (second_geometry)	Indicates if the two geometries share no points in common.
equals (second_geometry)	Indicates if the two geometries are of the same type and define the same set of points in the plane.
getPart ({index})	Returns an array of point objects for a particular part of geometry or an array containing a number of arrays, one for each part.
overlaps (second_geometry)	Indicates if the intersection of the two geometries has the same dimension as one of the input geometries.
touches (second_geometry)	Indicates if the boundaries of the geometries intersect.
within (second_geometry)	Indicates if this geometry is contained within another geometry.

L'exemple suivant présente le parcours d'une entité polygone.

```
>>> arcpy.env.workspace = "C:\Cours\ArcGIS\TP\TP1\TP1.gdb"
>>> curseur = arcpy.SearchCursor ("REGIONS")
>>> ligne = curseur.next()
>>> polygone = ligne.Shape #Polygone ILE-DE-FRANCE
>>> polygone.type
u'polygon'
>>> polygone.partCount
1
>>> polygone.pointCount
405
>>> tableau = polygone.getPart(0)
>>> type(tableau)
<class 'arcpy.arcobjects.arcobjects.Array'>
>>> tableau.count
405
>>> coords = []
>>> for pt in tableau:
...     coords.append([pt.X,pt.Y])
>>> print coords[0][0],cords[0][1] # Coordonnées du premier sommet
605600.052802   2367900.04789
>>> del curseur, ligne, polygone, tableau, coords, pt
```