# Programmation Python pour Arcgis

## Un exemple d'amélioration de l'outil « champs de vision » (*viewshed*)

**Eric Bailly**,
*université de Nice*
*eric.bailly@unice.fr*

**Une étude de visibilité peut être réalisée par le logiciel *Arcgis* à partir de l'outil « *champs de vision* » (viewshed). Il produit une carte binaire présentant les espaces visibles et invisibles à partir de points ou d'un tracé. Il ne propose toutefois pas comme « *origine de la visualisation* » une surface. Cet article présente un script *Python* permettant la création d'une carte de visualisation à partir d'une zone. Il se veut essentiellement pédagogique et expose une programmation itérative classique. L'objectif n'est pas de générer un outil de production ni l'optimisation d'un code, mais de montrer comment on peut interagir simplement avec *Arcgis* et créer ses propres outils.**

## Les champs de vision

### Aspects techniques

La littérature propose plusieurs termes synonymes pour définir une zone visible à partir d'un point (ou d'un ensemble de points formant une ligne ou une surface). On retrouve ainsi le bassin de vision, champ de vision ou de visibilité, en anglais *viewshed*. La définition mathématique en est la suivante : « *Étant donné un point de vue P sur un terrain (domaine D), la surface visible Sv correspond à l'ensemble des points de la surface qui sont visibles de P : Sv (P) = {W ∈ D \ W est visible de P}* » (1). Dans la pratique, l'opération *viewshed* nécessite plusieurs données, notamment le point d'origine de la visualisation et une matrice d'altitude du terrain (modèle numérique de terrain). D'autres options peuvent être ajoutées (taille de la personne qui regarde, courbure de la terre (2), portée du regard, angles, etc.).

À partir de ces informations, le logiciel calcule l'ensemble des zones visibles depuis le point d'origine. Le principe est le même pour une ligne (elle est découpée en points, la carte représentant l'union des visibilités élémentaires).

### Exemples d'utilisation et problématique

Les études de visibilité sont souvent utilisées dans l'analyse des paysages pour, par exemple, préparer la mise en place d'un belvédère, étudier l'impact d'un futur aménagement ou définir les espaces visibles à partir d'un sentier touristique (3).

Prenons l'exemple d'une commune qui envisage d'installer un terrain dédié à l'accueil de camping-cars. Deux problématiques liées au paysage se posent à elle : tout d'abord, quels impacts visuels cet aménagement va-t-il engendrer ? Ensuite, quelle partie du territoire pourra voir un touriste lorsqu'il sera sur cet emplacement ? Afin de répondre à ces questions, il faut créer une carte présentant les zones visibles à partir de ce terrain. Si *Arcgis* offre un outil d'analyse visuelle (*viewshed* - champs de vision), ce dernier ne permet de définir, en tant que « *terrain d'observation* » qu'un point ou une ligne, mais pas un polygone. Si le fichier de points contient plusieurs éléments ponctuels, c'est celui ou

ceux que l'utilisateur sélectionne qui serviront d'origine de la visualisation. Si aucun n'est choisi, l'outil *viewshed* fusionne tous les *viewsheds* pour chaque point du fichier. Ainsi il suffit de transformer notre zone en semis de points pour réaliser notre « *viewshed zonal* ».

Ce travail est organisé de la façon suivante :

Nous allons tout d'abord découper le MNT selon notre zone d'étude. Nous obtenons une matrice de points d'altitude de la forme de notre zone.

Figure 1 : le raster : chaque pixel contient une valeur correspondant à une altitude.

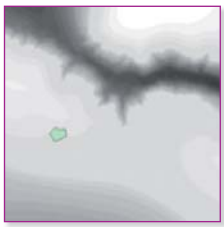


Figure 2 : la zone d'étude positionnée sur le *raster*.

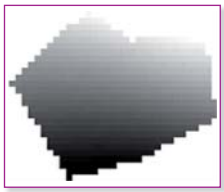


Figure 3 : le découpage du *raster* par la zone d'étude.

Ensuite nous transformons cette matrice en points. *Arcgis* permet de passer d'une matrice à un ensemble de points centrés sur chaque pixel comme le montre la figure 4.

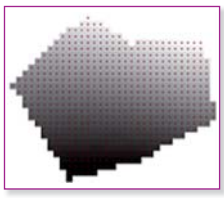


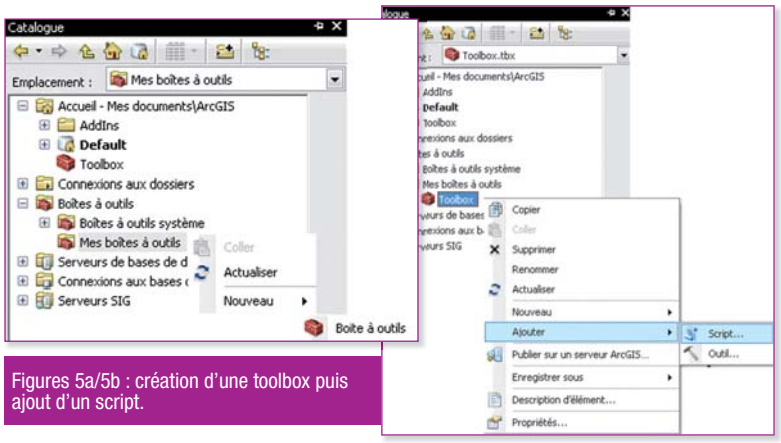
Figure 4 : les points positionnés au centre de chaque cellule.

Enfin, l'outil *viewshed* va utiliser ces points comme origine du traitement. Même s'il est possible de réaliser ces opérations directement dans *ArcMap*, nous allons créer un script en *Python* et l'intégrer dans la *toolbox* d'*Arcgis* afin d'en comprendre le fonctionnement.

Dans un deuxième temps, nous pourrions utiliser le résultat de la visualisation pour chacun des points, afin, par exemple, d'effectuer une animation (en superposant au fur et à mesure des images comme dans un « *gif animé* »). *ArcMap* permet de le faire, mais cela devient difficilement gérable lorsque les points se multiplient. Nous allons donc ajouter cette possibilité à notre *script* et ainsi présenter le fonctionnement d'une boucle. *In fine* le script rendra un *viewshed* zonal et l'ensemble des *viewsheds* pour chacun des points qui compose notre zone.

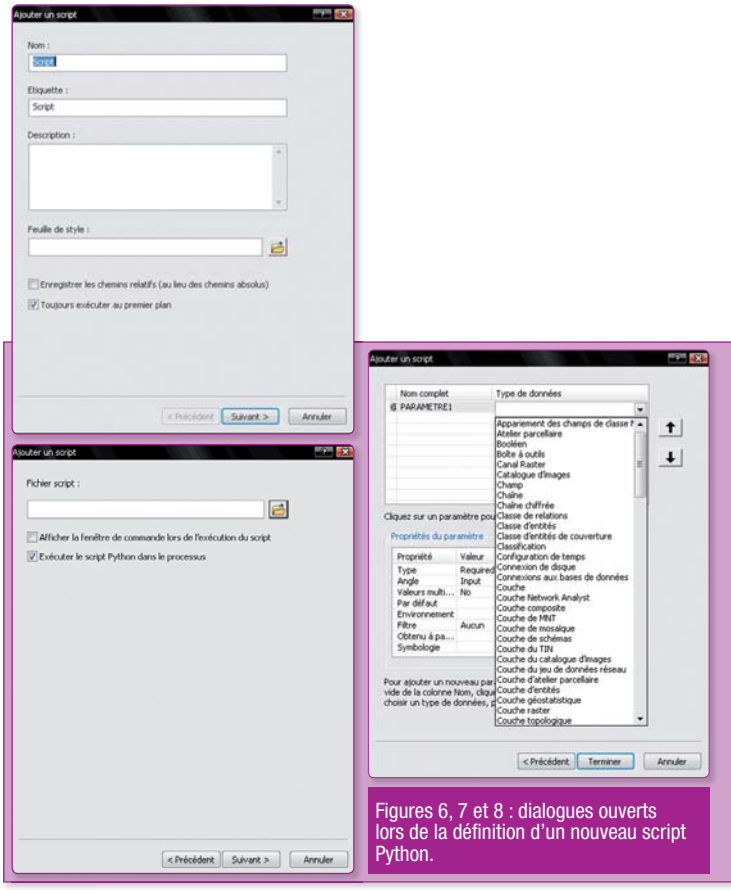
## Création d'une *toolbox*

Afin de créer un outil facile à utiliser, nous allons l'intégrer dans une toolbox et appeler une boîte de dialogue permettant à l'utilisateur de spécifier des paramètres tels que les noms et les emplacements des fichiers. Pour ce faire, il faut aller dans l'icône catalogue (à droite de la fenêtre), sélectionner « *mes boîtes à outils* », puis effectuer un clic droit. Le menu contextuel propose « *nouveau* » et « *boite à outils* » (figure 5)




Figures 5a/5b : création d'une toolbox puis ajout d'un script.

Une *toolbox* est créée ; il est possible de la renommer. Un clic droit sur cette nouvelle boîte à outils permet d'obtenir un menu contextuel offrant la possibilité d'ajouter un script. Une boîte de dialogue permet de le nommer puis d'associer un fichier *.py*. Il suffit ensuite de spécifier les paramètres du *script*.




Figures 6, 7 et 8 : dialogues ouverts lors de la définition d'un nouveau script Python.

## Algorithmes

La figure suivante propose l'algorithme informatique de la première partie, le « *viewshed* zonal ».

Nous aurons donc besoin des outils *Arcgis* suivants :

- *Clip_management*, qui découpe une matrice (*raster*) à partir d'un *shapefile* zonal ;
- *RasterToPoint_conversion*, qui passe d'une matrice (*raster*) à un fichier de points ;
- *Viewshed*, qui crée une carte de visibilité (matrice) à partir des points.

La deuxième partie est une boucle que l'on trouve classiquement en programmation : *CreateFeatureclass_management* crée une classe (Point, Multipoint, Polyligne, Polygone) dans le fichier spécifié, *insertRow* copie un point dans un autre et *Viewshed* engendre une carte de visibilité (matrice) à partir des points.

### Détails du programme en *Python*

Depuis la version 10 d'*Arcgis*, le module *Arcgisscript* a été remplacé par le site-package *arcpy*. Il contient l'ensemble des instructions nécessaires à la programmation d'*Arcgis*, dans le langage *Python* (à partir de la version 2.6, intégrée avec *Arcgis* 10) et inclut notamment les outils de géo-traitement.

La société *Esri* fournit aussi une documentation en ligne à cette adresse : *http://help.arcgis.com/fr/arcgisdesktop/10.0/help/index.html*. Une communauté française importante et très active participe sur plusieurs forums dont : *http://www.forumsig.org* et *http://georezo.net*.

Rappelons aussi qu'il est tout à fait possible d'obtenir un résultat par des moyens différents. Le

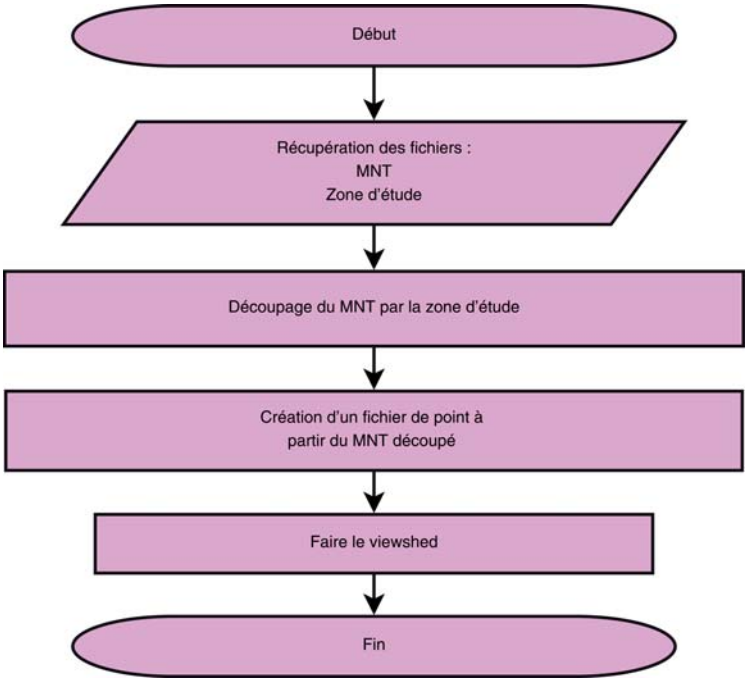



Figure 9 : organigramme d'exécution du viewshed zonal.

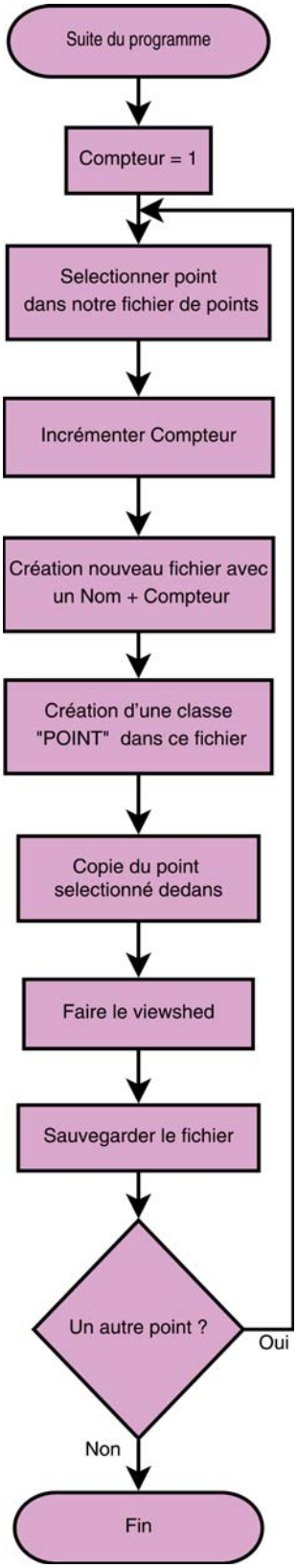



Figure 10 : organigramme de la seconde partie.

code qui va suivre n'est qu'un exemple et ne demande qu'à être complété, modifié et optimisé. Nous utilisons la version 10.1 qui corrige de nombreux problèmes.

Les outils *arcpy* sont tous appelés sous la forme : *arcpy.NomOutil_aliasBoiteOutils*. Notons que la casse (majuscules/minuscules) est prise en compte. C'est souvent une source d'erreur de type : *name 'xxx' is not defined*. Tout programme doit commencer par un entête qui spécifie son nom, sa description, ses entrées/sorties, le nom de l'auteur et sa fonction, voire ses coordonnées et la date de création du code.

On importe ensuite *arcpy* de cette façon :

```
import arcpy
```

Le plus souvent, lorsque l'on utilise des fichiers, on définit le répertoire dans lequel tous les accès se feront :

```
arcpy.env.workspace = "chemin d'accès"
```

Les instructions suivantes montrent comment produire un document Arcmap de type *.mxd*.

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

**"CURRENT"** correspond au document ouvert par défaut lors de l'initialisation du programme.

On crée ici un bloc de données dans le document, le **[0]** dans la liste et on lui donne un nom :

```
arcpy.mapping.ListDataFrames(mxd)[0].name = "Mon_bloc_de_données"
```

On sauvegarde le tout :

```
mxd.saveACopy ("monprojet.mxd")
```

L'étape suivante consiste à récupérer les arguments spécifiés par l'utilisateur à travers la boîte de dialogue. Nous avons un *raster* (0) et notre *shapefile* (1).

```
  InputRaster = arcpy.GetParameterAsText(0)

  InputShp = arcpy.GetParameterAsText(1)
```

Voici comment établir des couches :

```
  Raster_layer = arcpy.MakeRasterLayer_management (InputRaster, "monraster")

  Zone_layer = arcpy.MakeFeatureLayer_management (InputShp, "mazone")
```

Les noms fournis entre guillemets seront ceux qui apparaîtront dans la table des matières (*table of contents*). On peut sauvegarder les couches sur le disque dur, les noms entre guillemets seront suffixés avec « *.lyr* » :

```
  arcpy.SaveToLayerFile_management(Raster_layer, "Raster_layer")

  arcpy.SaveToLayerFile_management(Zone_layer, "Zone_layer")
```

Pour faire afficher les couches dans la table des matières on peut utiliser cette méthode :

```
  df = arcpy.mapping.ListDataFrames(mxd)[0]

  addLayer = Zone_layer.getOutput(0)

  arcpy.mapping.AddLayer (df, addLayer, "AUTO_ARRANGE")

  addLayer = Raster_layer.getOutput(0)

  arcpy.mapping.AddLayer (df, addLayer, "AUTO_ARRANGE")
```

Trois positions sont possibles dans le bloc de données : "**AUTO_ARRANGE**", "**BOTTOM**" et "**TOP**".

Cette première partie permet de mettre en place l'ensemble de nos données. Nous allons maintenant montrer comment utiliser les géotraitements proposés par *Arcgis*.

On effectue le découpage du *raster* (*InputRaster*) par la zone de travail (*InputShp*). Le résultat est mis dans la variable *Clip_Raster. zoneclipee.img* est le nom de sauvegarde sur le disque dur. L'option *ClippingGeometry* demande à l'outil de découper le raster par le shapefile.

```
Clip_Raster = arcpy.Clip_management (InputRaster, "0 0 0 0", "zoneclipee.img", InputShp, 255, "ClippingGeometry")
```

La transformation du *raster* en un fichier de point se réalise avec cet outil :

```
Points_zone_clipee = arcpy.RasterToPoint_conversion (Clip_Raster, "Image_points.shp")
```

*Image_points.shp* est le nom de sauvegarde sur le disque dur.

Enfin on peut réaliser le traitement « *champs de vision* » et le sauvegarder sous la forme d'une couche.

```
Viewshed_zone = arcpy.Viewshed_3d (InputRaster, Points_zone_clipee, "Viewshed_zonal")

  Viewshed_layer = arcpy.MakeRasterLayer_management (Viewshed_zone, "Viewshed_zonal.lyr)

  arcpy.SaveToLayerFile_management (Viewshed_layer, "Viewshed _layer", "ABSOLUTE")
```

La première partie du programme est terminée, le *viewshed* zonal est finalisé. Passons maintenant à l'étape suivante qui consiste à produire et sauvegarder un *viewshed* pour chaque point.

On commence par créer un dossier dans lequel seront sauvegardées toutes les images :

```
arcpy.CreateFolder_management
(arcpy.env.workspace,"viewshed_
images")
```

La fonction qui suit forme un nom de chemin d'accès correct :

```
Dossier_images = os.path.join
(arcpy.env.workspace,"viewshed_
images")
```

qui devient notre workspace :

```
arcpy.env.workspace = Dossier_
images
```

On déclare ensuite un curseur pointant sur les éléments de notre fichier de points :

```
curseur = arcpy.SearchCursor
(Points_zone_clipee_local)
```

Une boucle permet d'itérer tous les éléments contenus dans le fichier de points :

```
for element in curseur :
```

Ensuite, un compteur permet de générer un nom de fichier différent pour chaque image :

```
  fichier_un_point = "fichier_
point" + str (cpt)
   cpt += 1
```

On construit dans ce fichier une classe d'entité point et un point (*InsertCursor*).

```
  monfichier = arcpy.
CreateFeatureclass_
management(arcpy.env.
workspace,fichier_un_
point,"POINT")
  rows = arcpy.InsertCursor
(monfichier)
```

Puis on copie dans le point produit le point du fichier de point

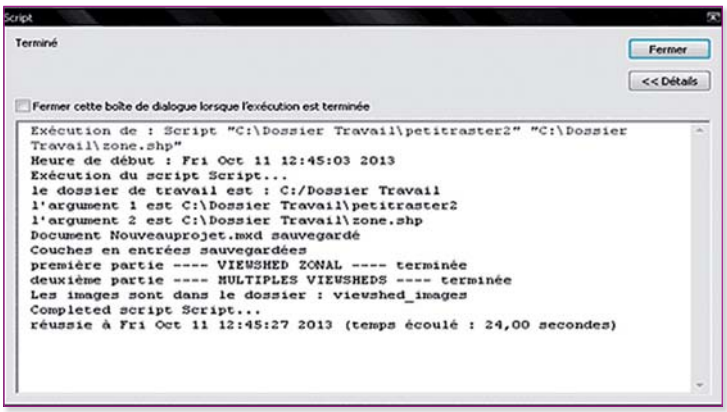



Figure 11 : fenêtre de script, dans laquelle on peut suivre le bon déroulement du programme *Python*.

```
rows.insertRow (element)
```

Comme précédemment on construit le *viewshed* en ayant pris soin d'itérer un nom.

```
   viewshed_S = "viewshed_
unpoint_"+str (cpt)+".tif"
   outViewshed = Viewshed(Input
Raster,monfichier)
Et on le sauvegarde :
  outViewshed.save(viewshed_S)
```

Lorsque tous les points ont été parcourus, la boucle est terminée ainsi que le *script*. L'exécution est tracée dans la fenêtre de *script* (figure 11).

Notre table des matières affiche la zone, le *viewshed* zonal et le raster, tandis que l'ensemble des *viewsheds* est sauvegardé dans le dossier *viewshed_images*.

## Conclusion

Cet article a présenté un ensemble de techniques permettant la création de scripts pour *Arcgis*. Avec un peu d'habitude il est très simple de réaliser ses propres routines. Sans oublier l'aide qu'apporte la lecture des nombreux *scripts* déjà réalisés et accessibles sur *Internet*.


Figure 12 : la table des matières.

## Bibliographie


[1] Karine Debavelaere Leroux, 2005. Réduction de MNA sous contraintes géomorphologiques ; impact sur la détermination du champ de vision. Mémoire de travail de fin d'études en vue de l'obtention du Diplôme d'Ingénieur de l'ESGT.

[2] P. Guth, 2005. Shortcuts in the line-of-sight and viewshed algorithms with gridded geographic dems. ASPRS 2005 annual conference, 7-11 mars 2005.

[3] Éric Bailly, Mohamed Ben Jeddou, Jean Marie Castex, Eric Gilli, Davtian Gourgen, Isabelle Mor, 2013. Cartographie des châtaigneraies, massif du Mercantour, Projet Intégré Transfrontalier (Alcotra 2010-2012), Axe 4 Planification « gestion durable des territoires agro-pastoraux », Parc National du Mercantour et Parco Alpi Maritime, Nice Métropole Côte d'Azur, AFA de la Tinée. Géomatique Expert 94, septembre 2013.

# Listing du programme

```
# Viewshed_zonal.py
# Description
# Entrées : un raster et une zone (shape)
# Sorties : un raster et un dossier cotenant des images
# Author : Bailly Eric, CMMC, Université Nice Sophia Antipolis
# Date : Septembre 2013
#################################################################################


# Importation des modules
import arcpy
from arcpy import *
from arcpy.sa import *

# Déclaration du dossier de travail dans lequel on trouvera les entrées et résultats
arcpy.env.overwriteOutput = True # permet l'écriture sur des fichiers déjà existants
arcpy.env.workspace = "C:/Dossier Travail"
arcpy.AddMessage ("le dossier de travail est : " + arcpy.env.workspace)

# On récupère les entrées de la boite de dialogue
InputRaster = arcpy.GetParameterAsText(0)
InputShp = arcpy.GetParameterAsText(1)

arcpy.AddMessage ("l'argument 1 est " + InputRaster)
arcpy.AddMessage ("l'argument 2 est " + InputShp)

# On crée un Document MXD
mxd = arcpy.mapping.MapDocument("CURRENT")
# On lui donne un nom
arcpy.mapping.ListDataFrames(mxd)[0].name = "Nouveauprojet"
mxd.saveACopy ("monprojet.mxd")
arcpy.AddMessage ("Document Nouveauprojet.mxd sauvegardé")

# On crée les couches
# Entre "" le nom de la couche qui sera affichée dans la TOC (table des matières)
Raster_layer = arcpy.MakeRasterLayer_management(InputRaster, "monraster")
Zone_layer = arcpy.MakeFeatureLayer_management(InputShp, "mazone")

# On sauve les couches
# Entre "" le nom sur le disque dur avec .lyr
arcpy.SaveToLayerFile_management(Raster_layer, "Raster_layer", "ABSOLUTE")
arcpy.SaveToLayerFile_management(Zone_layer, "Zone_layer", "ABSOLUTE")
arcpy.AddMessage ("Couches en entrées sauvegardées")

# On place les cartes dans la tables des matières
df = arcpy.mapping.ListDataFrames(mxd)[0]

addLayer = Zone_layer.getOutput(0)
arcpy.mapping.AddLayer(df, addLayer, "AUTO_ARRANGE")

addLayer = Raster_layer.getOutput(0)
arcpy.mapping.AddLayer(df, addLayer, "AUTO_ARRANGE")

# Première partie du programme : le viewshed zonal

# On déclare le fichier dans lequel sera mis la zone clippée
Clip_Raster = "zoneclipee.img"

# Découpage du raster par la zone de travail
arcpy.Clip_management(InputRaster, "0 0 0 0",Clip_Raster, InputShp, 255, "ClippingGeometry")

# On déclare le fichier dans lequel sera mis la zone clippée
```

```
Points_zone_clipee = “Image_points.shp”

# Du raster vers un fichier de points

Points_zone_clipee_local = arcpy.RasterToPoint_conversion(Clip_Raster, Points_zone_clipee)

# le viewshed zonal

viewshed_zonal = Viewshed (InputRaster,Points_zone_clipee)

# on le reclassify pour obtenir une image binaire
valeur_reclass = “1 10000 1” # les valeur de 1 à 10000 prennent la valeur 1
viewshed_zonal_fin = arcpy.Reclassify_3d (viewshed_zonal, “value”,valeur_reclass)

# on en fait une couche .lyr, on la sauve, on l’affiche dans la table des matières
viewshed_zonal_layer = arcpy.MakeRasterLayer_management(viewshed_zonal_fin, “viewshed_zonal”)
arcpy.SaveToLayerFile_management(viewshed_zonal_layer, “viewshed_zonal_layer”, “ABSOLUTE”)
df = arcpy.mapping.ListDataFrames(mxd)[0]
addLayer = viewshed_zonal_layer.getOutput(0)
arcpy.mapping.AddLayer(df, addLayer, “AUTO_ARRANGE”)

arcpy.AddMessage (“première partie ---- VIEWSHED ZONAL ---- terminée “)

# Deuxième partie du programme : les multiples viewshed

# on créer un dossier contenant les viewshed
arcpy.CreateFolder_management (arcpy.env.workspace,”viewshed_images”)
# on sauvegardera dedans
Dossier_images = os.path.join (arcpy.env.workspace,”viewshed_images”)
arcpy.env.workspace = Dossier_images

# Debut des traitements

# création d’un curseur qui va pointer les enregistrements c’est à dire sur chaque points de notre
zone_
curseur = arcpy.SearchCursor (Points_zone_clipee_local)

# un compteur pour incrémenter les noms des fichiers
cpt = 1

# La boucle qui passe en revue tous les points du fichier Points_zone_clipee_local
for element in curseur :

    fichier_un_point = “fichier_point” + str (cpt) #incrémentation des noms de fichiers
    #création de la classe “point”
    monfichier = arcpy.CreateFeatureclass_management(arcpy.env.workspace,fichier_un_point,”POINT”)
    #création du point
    rows = arcpy.InsertCursor (monfichier)
    #on copie dans le point créer le point du fichier de point
    rows.insertRow (element)
    #incrémentation des noms des viewshed
    viewshed_S = “viewshed_unpoint_”+str (cpt)+”.tif”
    #le viewshed
    outViewshed = Viewshed(InputRaster,monfichier)
    #on le sauvegarde
    outViewshed.save(viewshed_S)
    #itération du compteur
    cpt += 1

arcpy.AddMessage (“deuxième partie ---- MULTIPLES VIEWSHEDS ---- terminée “)

arcpy.AddMessage (“Les images sont dans le dossier : viewshed_images”)
```