Dobesova, Z.(2012): Visual programming for novice programmers in geoinformatics.
SGEM 2012 12th International Multidisciplinary Scientific GeoConfrenceConference,
Proceedings Volume III, STEF92 Technology Ltd., Sofia, Bulgaria, 433-440p.
ISSN 1341-2704, DOI: 10.5593/sgem2012

VISUAL PROGRAMMING FOR NOVICE PROGRAMMERS

IN GEOINFORMATICS

Ing. Zdena Dobesova, Ph.D.

Palacký University, Olomouc, **Czech Republic**

ABSTRACT

Geographic information systems are used by geographers, engineers, surveyors, clerks that are not familiar with text programming. Visual programming brings to these users the opportunity to construct the batch data processing program in a graphical way.

GIS software is frequently supplemented by components for visual programming. These components are aimed mainly for easy programming of the data batch processing. Advantage is storing the program of batch processing for the repetitive running on various spatial data sets. Furthermore, exchanging program between users is valuable.

From my perspective as a teacher I am most interested in the question how novices learn to program. Visual programming is a fresh start point for a novice programmer. Visual program is more comprehensible than textual programming. Evaluation of possibilities of the component ModelBuilder in ArcGIS software is mentioned. Model Builder produce program called “model” than can be converted to Python scripting language. This moment changes common users to programmers. Comparison of the same graphic version of model with converted textual Python script helps to understand commands. Novice programmer learn how call the geoprocessor module, check licensees of extensions, how to set input parameters and how call tolls from Arc Toolbox. Visual programming is easy in ModelBuilder. Subsequently, users are able to comprehend and write the Python scripts quickly.

Keywords: geoinformatics, visual programming, ModelBuilder, data flow, Python.

INTRODUCTION

Graphical construction of data flow processing is called visual programming. Some GIS products have the component for visual programming. The graphical symbols and functionality of some GIS products are described and assessed in previous articles [4, 5, 8]. The assessment can be from the point of the amount of functionality and the cognitive dimensions [1, 2, 3, 11].

ArcGIS software has component for construction of batch processing data that is called ModelBuilder. The data flow can be constructed graphically by basic set of boxes and arrows (Fig. 1). ArcGIS functions-tools are mainly accessible in Arc Toolbox. Tools are arranged in toolboxes. Tool can be called interactively or can be called in models or by scripting languages. ArcGIS support several scripting languages [7, 10]. Scripting language Python is one of them. Scripting in Python is very well supported by manual and help pages [10]. Moreover, other additional set of methods of geoprocessor can be called in Python script directly. These methods can not accessible in ModelBuilder. The offer of possibilities for programming in Python is wider than offer of tools for design

model in ModelBuilder. Useful feature of ModelBuilder is the ability to convert graphical model constructed only by visual programming to the scripting language Python. It was used in the teaching process.

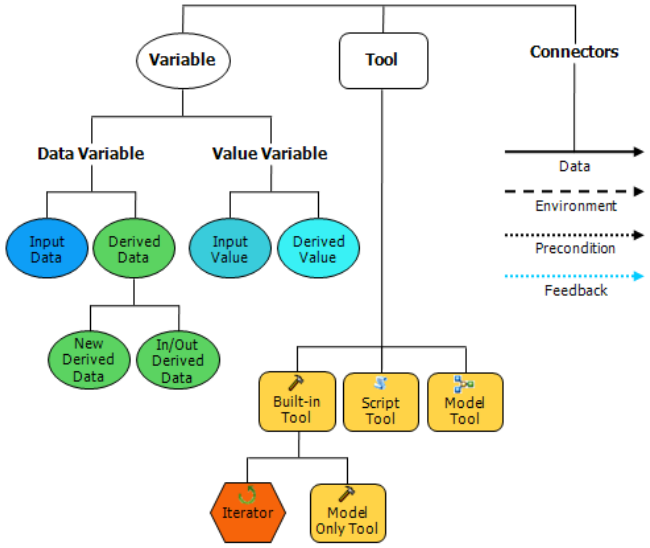


Fig. 1: Graphical symbols in ModelBuilder [10]

Visual programming versus scripting in Python

The question is: What is better for the user?

- Start directly with learning scripting in Python for ArcGIS to utilize wider possibilities.
- Start with learning design of model in ModelBuilder and following by learning textual programming in Python.

The second option appeared better for novice programmers than the first. The experience of teaching supports that choice. Starting with ModelBuilder is better because visual programming is very easy for users. Data flow is designer simply by dragging and dropping tools and connection with arrows. Design is quick. Design has not been accurate in syntax as textual programming. Model is well graphical arranged, transparent and comprehensible. The amount of functions is some cases enough for user needs in uncomplicated models. The important extension of ModelBuilder is a supplement by Iterators in version ArcGIS 10. Iterators are constructors for construction cycles.

Steps of learning scripting in Python

Experiences from learning group of students at Palacký University bring following order of steps.

1. Design a simple model with one tool from the toolbox.
2. Design model with more tools.
3. Design model with a tool from extension.
4. Change model to the parametric model.

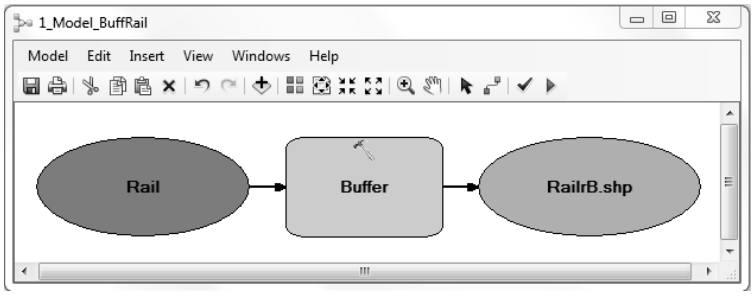
5. Design model with iterators.
6. Design model with other nested model.

Firstly, the model in ModelBuilder is constructed in each step (1 to 6). Secondly, the exported model in Python script is explored in each step. The output scripts content more and more commands. The user is inspired how to write a script.

Students are not without any knowledge of textual programming. They know the theory of object programming. All ArcGIS tools (Clip, Buffer, Simplify, etc.) are called as object methods with arguments. They have basic knowledge of construction programing cycles and condition in Python. Base rules of Python syntax are mentioned previously. Case sensitivity is sometimes threat and source of errors for novice programmers in Python. Python has not explicit declaration of variables. It is an advantage and disadvantage. User has not to declare variables, in the other side it is a source of mistakes causing type errors.

Design a simple model with one tool from the toolbox

The first simply model only contains one tool - rounded box (Fig. 2). Input and output data are expressed by ellipses. The converted script contains the header with commentary line. It is the inspiration for user. Notes with date of creation, name and description as comments are relevant. Command "import arcpy" imports module. It is a python file that generally includes classes and functions - geoprocessing tools. Students remember that import arcpy is the first command in all scripts. Tool Buffer_analysis is called as a method with argument in rounded brackets. Using local variables is also beneficial inspiration. The second part of name is alias on the toolbox in method. This fact must be explained by the teacher or help.



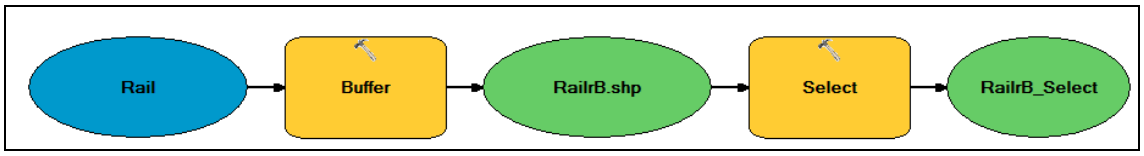

```
# ---------------------------------------------------------------------------
# 1_BuffR.py
# Created on: 2012-05-04 17:01:45.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy


# Local variables:
Rail = "D:\\Railr.shp"
RailrB = "D:\\RailrdB.shp"

# Process: Buffer
arcpy.Buffer_analysis(Rail, RailrB, "500 Meters", "FULL", "ROUND", "NONE", "")
```

Fig. 2: Model and corresponding script with one tool

Next examples of scripts are only cuts out parts of scripts. In fact, commands and parts mentioned previously are also present.

Design model with more tools

There are two tools - Buffer and Select in example model (Fig. 3). In visual programming, there is simply one tool follows second tool in the process line. Ellipse express feature class that is the output data from first tool and the same data is the input data for next tool. The sequence of tools is comprehensible. Commands that call tools as methods are in separate rows that follow in the same order as in the model. The name of feature classes in case output data is the second argument of the first method (RailrB_shp). Programmer must notice it and remember it. Shortly, ellipses are arguments of methods. There is no equivalence command for ellipse in script code. Good inspiration for novice programmers is that each calling of the method is introduced by commentary line # Process: Buffer etc.

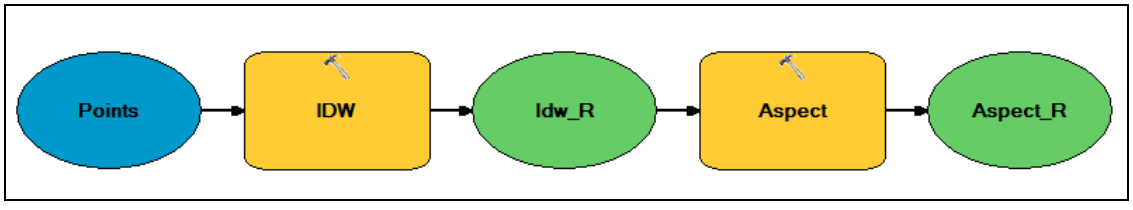



```
# Process: Buffer
arcpy.Buffer_analysis(Rail, RailrB_shp, "500 Meters", "FULL", "ROUND", "NONE", "")

# Process: Select
arcpy.Select_analysis(RailrB_shp, RailrB_Select, "ICC = 17")
```

Fig. 3: Model and corresponding script with two tools

Design model with a tool from extension

IDW and Aspect tools are used from toolbox “3D Analyst Tools”. This toolbox belongs to the extension 3D Analyst. Method “arcpy.CheckOutExtension” retrieves the license from the License Manager in Python script (Fig. 4). No graphic symbol corresponds to this method in the model. The necessity of calling the method “CheckOutExtension” is the new information for programmer when some tool from extension is used.



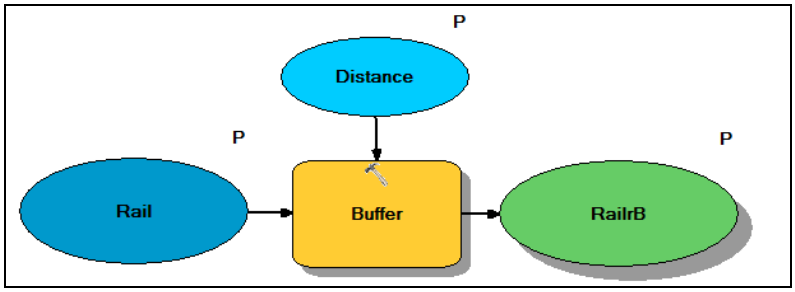

```
# Check out any necessary licenses
arcpy.CheckOutExtension("3D")
```

Fig. 4: Model and corresponding script where is necessary to retrieve extension

Converted script does not solve if the extension is not available. To write functional script is necessary to test if the extension is available by method ”CheckExtension”. Condition is “if arcpy.CheckExtension("3D") == "Available": ….” . Inspiration for full script code can be found in ArcGIS Help.

Change model to the parametric model

Parametric model is universal. Different data can be processed by the same model. Letter “P” expresses that that data are parameters in the model (Fig. 5). Method GetParameterrrAsText gets the specified parameter by its index position from the list of parameters. Script has the solution for the situation when parameter is unspecified. Default value is to set in condition “if ..”. The prevent solution of errors is also inspiriting.



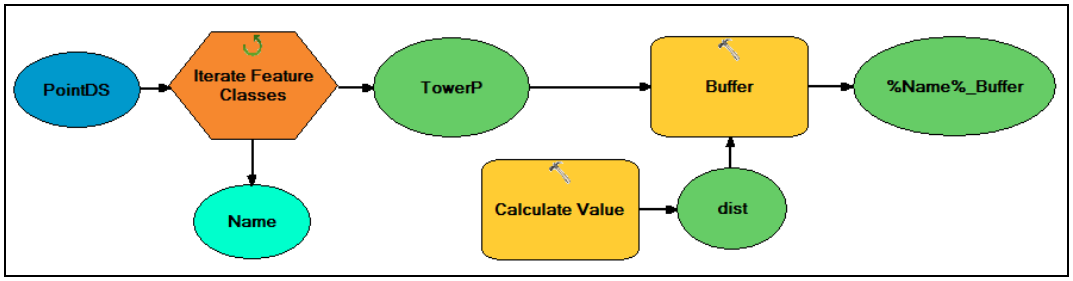

```
# Script arguments
Rail = arcpy.GetParameterAsText(0)
if Rail == '#' or not Rail:
    Rail = "D:\\Railr.shp" # provide a default value if unspecified

RailrB = arcpy.GetParameterAsText(1)
if RailrB == '#' or not RailrB:
    RailrB = "D:\\RailrdB.shp" # provide a default value if unspecified

Distance = arcpy.GetParameterAsText(2)
if Distance == '#' or not Distance:
    Distance = "500 Meters" # provide a default value if unspecified

# Local variables:
# Process: Buffer
arcpy.Buffer_analysis(Rail, RailrB, Distance, "FULL", "ROUND", "NONE", "")
```

Fig. 5: Parametric model and corresponding script

Design model with iterators

The Iterator toolset has twelve iterators that help repeat a process or set of processes on a set of inputs. Hexagon is a symbol of iterator in model (Fig. 6). Exported Python script calls the ModelBuilder toolset “mb” for calling iterators. Better construction is a combination of Enumeration methods and cycle “for” or “while”. Conversion of iterator to Python shows only one possible way**;** nevertheless construction of cycles is

better. In case of iterators, it is better learning the geoprocessor enumeration methods for scripting directly. There is not useful inspiration from exported script.



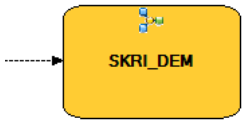

```
# Process: Iterate Feature Classes
arcpy.IterateFeatureClasses_mb(PointDS, "", "POINT", "NOT_RECURSIVE")
```

Fig. 6: Model and corresponding script with iterators

Internal variable that are set in the model as %Name% (ellipse on the right side in Fig. 6) used for composition of the name of output data is not exported to Python correctly. This construction must be realized manually by several commands directly in Python. Example for construction the output feature name based on the input feature name to name data automatically is described in the article „Automatic generation of digital elevation models using Python scripts" [6].

Nested model

It is possible input to the model another model as a nested model. ModelBuilder is limited to use only one iterator in model. Nested model is a solution for using more iterators in one model. Commonly is possible to design model with nested models. Nested model has in ModelBuilder yellow box with the small icon in (Fig. 7). This idea is didactic. It prepares novice programmer to the idea about division program to more part – subprograms. The advantage of subprogram is quick, repetitive use of the same set of commands when they can be called as subprogram.




```
# Process: Morpho_Spline
arcpy.gp.toolbox = "D:/SKRI_DEM";

arcpy.gp.Model_AliasSKDEM(SP_InputPoints)
```

Fig. 7: Model and corresponding script with nested model

Writing scripts in Python

Students are able to write scripts directly in Python after experiences with converted scripts. The phase of direct writing of scripts in Python can be started. New other constructions, methods students can adopt to scripts to improve them. The quality and functionality of scripts is increased.

E.g. the process of model exporting to Python script do not add “try: except:” construction to the exported script. Error-handling statement “try: except:” is used to handle any unexpected errors during execution of a script. This construction has to be written directly in Python by student.

CONCLUSION

The article observes one way how to start the learning of scripting for ArcGIS. The novice programmers are considered. The recommendation is firstly to start with construction data flows in ModelBuilder and after that continue with scripting. The starting point is the conversion of a simple model to Python scripts. Robinson mentioned that programming courses are generally regarded as difficult [12]. Practical experience with teaching and learning confirms that functionality of converted model is more comprehensible to students. Finally, student evaluate this way as quick introducing to scripting. Presented way demonstrates how to better teach novice programmers.

Set of examples in ModelBuilder and correspondent script is demonstrated. Some constructions are inspiring: import arcpy model, calling tool as method, using of variables, getting input parameters to script, check extension, commentary lines. A few functions are converted from model to script in a different way or incorrectly. Deep study of scripting is recommended to the students after this initial phase.

There is relationship between the ability of comprehend the script and the ability to generate new script. Running the model and script with the same tasks supports the comprehension. As a result, the students are able to generate new own scripts.

ACKNOWLEDGEMENT

The research was supported by the project of Internal Grant Agency of Palacký University in Olomouc No. 2012_007 "The small format aerial photography in the study of the effect of surface heterogeneity on the habitats".

REFERENCES

[1] Blackwell, A. Green. T. A Cognitive Dimensions Questionnaire, Available: http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf

[2] Blackwell A. F. Cognitive Dimensions of Notations Resource Site. Available: http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/index.html

[3] Blackwell A. F. and Green. T. R. G. National systems-the Cognitive Dimensions of Notations framework. J.M. Carrol (Ed.) HCI Models, Theories, and Frameworks, Toward a Multidisciplinary Science, San Francisco. Morgan Kaufman, 2003, pp. 103-1034.

[4] Dobesova, Z.: Visual programming language in geographic information systems, Recent Researches in Applied Informatics, Proceedings of the 2nd International Conference on Applied Informatics and Computing Theory, AICT `11, Prague, 2011, NAUN/IEEE. AM, WSEAS Press, pp. 276-280, ISBN 978-1-61804-034-3.

[5] Dobesova, Z. : Programming Language Python for Data Processing, Proceedings of International Conference on Electrical and Control Engineering (ICECE), Yichang, China, 2011, IEEE, CFP 1173J-PRT, Volume 6, pp. 4866-4869. ISBN 978-1-4244-8163-7.

[6] Dobesova, Z.: Automatic generation of digital elevation models using Python scripts Conference Proceedings SGEM 2011, 11th International Multidisciplinary Scientific GeoConference. STEF92 Technology Ltd., Sofia, Bulgaria, 2011, ISSN 1314-2704, pp. 599-604.

[7] Esri. Geoprocessing Model and Script Tool Gallery. Available: <http://resources.arcgis.com/gallery/file/geoprocessing>

[8] Dobesova, Z.: Assessment of Visual Languages in Geographic Information Systems, IEEE Symposium on Visual Languages and Human-Centric Computing, Innsbruck, 2012 [in print]

[9] Esri. Geoprocessor Programming Model ArcGIS 9.3, Available: <http://webhelp.esri.com/arcgisdesktop/9.3/pdf/Geoprocessor_93.pdf>

[10] Esri. ArcGIS Desktop Help 10, What is ModelBuilder? Available: <http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/What_is_ModelBuilder/002w00000001000000/>.

[11] Green T. R. G., Petre, M. Usability analysis of visual programming environments: A Cognitive Dimensions Framework. Journal of Visual Languages and Computing. 7. pp. 131–174, 1996.

[12] Robins, A., Rountree, J., Rountree, N.: Learning and Teaching Programming: A Review and Discussion, Computer Science Education, Vol.13, No.2, Swets & Zeitlinger, pp. 137-172, 2003.