Chapter 8
Controlling Flow

Abstract As you study the capabilities of programming for GIS, applications to your own work may come to mind. For example, you may need to run a geoprocessing tool on distributed batches of datasets or tweak multiple tables before they can be imported into ArcGIS. Before writing scripts for tasks like these, it's helpful to outline the main steps without belaboring syntax details, but instead using *pseudocode*, a generic format for outlining workflow. You also need to be familiar with the building blocks of workflow. How will the code make decisions based on the input? How will it repeat a process for each file? Chapters 9 and 10 describe the Python syntax for 'branching' and 'looping', but that part will seem easy, compared to the logic involved in designing the workflow. This chapter uses pseudocode examples to introduce 'branching' and 'looping'.

Chapter Objectives
After reading this chapter, you'll be able to do the following:

- Define the terms algorithm and pseudocode.
- Translate a GIS problem into a set of succinct steps.
- Use the terms looping, and branching to a describe workflow.
- Identify the three key components in a looping structure.
- Use indentation to group blocks of pseudocode.
- Interpret pseudocode.

8.1 Outlining Workflow

From a high-level perspective, all workflows are made up of three basic structures: *sequential* steps, *decision-making*, and *repetition*. Directions from point A to point B are given in sequential steps. Checking if data is zipped, tarred, or uncompressed is done with decision-making. Clipping every file in a geodatabase is performed with repetition. Decision-making and repetition are also referred to, respectively, as *branching* and *looping*.

Table 8.1 shows simple examples of these structures expressed with pseudocode. The examples include two common types of repetition. Workflows usually involve some combination of sequence, repetition, and decision-making steps.

L. Tateosian, *Python For ArcGIS*, DOI 10.1007/978-3-319-18398-5_8

Table 8.1 Pseudocode examples of the three basic flow structures.

Flow structure	Example

Sequence	`Hitch horse to sleigh.`
	`Go over the river.`
	`Go through the woods.`
	`Arrive at grandmother's house on the right.`

Repetition (looping)	`FOR each file in state layers geodatabase`
	`  Clip the file on the county boundary.`
	`ENDFOR`

	`SET buffer distance to 1 mile.`
	`WHILE buffer distance < 6 miles`
	`  Buffer the shapefile with the buffer distance.`
	`  INCREMENT buffer distance by 1 mile.`
	`ENDWHILE`

Decision-making (branching)	`IF data is compressed THEN`
	`  Uncompress data.`
	`ENDIF`

Examples 8.1–8.3 demonstrate these structures with pseudocode for three applications (clipping batches of data, tweaking tabular data, and downloading data). The pseudocode in Example 8.1 outlines steps for finding the weighted sum of a set of rasters to analyze dumping patterns for sediment dredged from canals. To create the rasters, the workflow repeats a sequence of steps (steps 3–5) for each input table. These steps convert tabular data to a spatial format, clip the spatial data using the disposal region, and convert the vector results to raster format.

Example 8.1: Pseudocode for processing dredging sediment data repeats steps 3–5 for each table.

```
1 GET a list of the weekly data base tables.
2 FOR each table in the list
3     Make a spatial layer from the table.
4     Clip the new layer on the disposal site extent.
5     Convert clipped vector data to raster format.
6 ENDFOR
7 Find weighted sum of output rasters.
```

Pseudocode uses capitalized keywords such as `GET` and `FOR...ENDFOR` to denote workflow structures and actions. Table 8.2 has a list of common pseudocode keywords. Keywords with openings and closings such as `FOR...ENDFOR`, `WHILE...ENDWHILE`, and `IF...ELSE...ENDIF`, are *block structures*. Items within a block structure are indented the same amount to indicate that they are related. Since steps 3–5 are surrounded by the `FOR...ENDFOR` structure, this block of pseudocode is indented, reinforcing that these three steps are repeated for each table. This also emphasizes that step 7 is not repeated for each table. Step 7 needs to find the

Table 8.2 A short list of common pseudocode keywords.

Action	Pseudocode keywords
Input	`READ, GET`
Output	`PRINT, SHOW`
Initialize	`SET`
Add one or subtract one	`INCREMENT or DECREMENT`
Repetition (looping)	`WHILE...ENDWHILE, FOR...ENDFOR`
Decision-making (branching)	`IF, THEN, ELSE IF, ELSE, ... ENDIF`
Function (procedure or subroutine)	`FUNC, ENDFUNC, CALL, RETURN`

weighted sum of all the output rasters, so this must only occur one time—after all the output rasters have been created. The `FOR...ENDFOR` structure is usually referred to as a `FOR`-loop because the workflow *loops* back to repeat steps. For example, if there are three tables in the list, the line number workflow will run 1-2-3-4-5--2-3-4-5--2-3-4-5--2-3-4-5--6-7.

Though each line in the pseudocode corresponds to a step in the process, the steps are defined broadly enough so that details do not obscure the overall flow. For example, making a spatial layer from a table might involve finding the latitude and longitude field names and choosing a projection but the pseudocode doesn't need to include these details.

Example 8.2 outlines steps for automatically reading data links from a data hosting Web page, downloading the data from those links, and uncompressing the data. The workflow contains decision-making structures as well as a repetition structure because each data file is only uncompressed if needed. The paired steps 8–9 and steps 11–12 handle two distinct kinds of compression files. These lines are indented three levels because the `IF...ELSE IF...ENDIF` structure is inside the `FOR... ENDFOR` structure. Since it is inside of the `FOR...ENDFOR` structure, decision-making occurs for each dataset linked to the page.

Example 8.2: Pseudocode for automatically fetching and uncompressing data files listed on a web site.

```
1  GET the data center web page URL.
2  READ the data center web page contents.
3  GET a list of links to data in the page contents.
4  FOR each data link
5      Fetch the data from the specified link.
6      Save the data.
7      IF the data is zip compressed THEN
8          Unzip data.
9          Delete zip file.
10     ELSE IF the data is tar compressed THEN
11         Untar data.
12         Delete tar file.
13     ENDIF
14 ENDFOR
```

Example 8.3 specifies the steps to prepare some tabular data with spatial fields to be imported into ArcGIS. In this example, the input tabular spatial data needs several modifications to fulfill GIS file import formatting requirements: the field names are not in the first row, the table has unnecessary trailing columns, and the field names have unacceptable special characters. This example uses both `WHILE`-loops and `FOR`-loops for repetition.

- `FOR`-loops are used to repeat steps *for each* item in a list of items such as, tables, data links, or field names. In this kind of `FOR`-loop, a *looping variable* successively gets the value of (or *iterates* through) each item in the list.
- `WHILE`-loops are used to repeat steps for a set of numbers (for example, row numbers 1–5 or buffer distances of 2–10 miles). A looping variable iterates through the numerical values. `WHILE`-loops can also be used to stop after an unknown number of steps based on the fulfillment of some condition (e.g., loop while the desired field header has not been found).
- There are three basic components of the `WHILE`-loop, initializing, checking, and updating the looping variable. In Example 8.3, these `WHILE`-loop components are as follows:

  1. Initialize looping variable: `SET row number to 1`
  2. Test looping variable: `row number < field name row number`
  3. Update looping variable: `INCREMENT row number`

Initializing the `WHILE`-loop variable sets the starting value, so that it can be tested with a condition which is either true or false. This controls the flow of the steps (if the condition is true, go to step 5; if not, go to step 8). In other words, when this condition is false, the loop terminates. If the condition is true, the indented steps, such as 'delete row' and 'increment row number' in Example 8.3 are performed. In most cases, the looping variable should be updated at the very end of the loop, after all of the other indented steps. Omitting the looping variable update results in never-ending repetition, since the loop condition never becomes false.

When a `FOR`-loop is used on a list of items, the three loop components are usually implied with the `FOR each` statement. For example, for lines 14–16 in Example 8.3, these components could be described as follows:

1. Start at the beginning of the list: Initialize the looping variable to the first field name in the list.
2. Check if there are still more items in the list: If the looping variable is still set to a field name, go to step 15. If not, go to step 17.
3. When the indented steps have been performed, point the looping variable to the next field name in the list or an 'end of list' flag.

Notice that in all of the `FOR`-loop and `WHILE`-loop examples, the looping variable is used inside the loop. In Example 8.1, the table is made into an X Y layer. In Example 8.2, the data is fetched from the specified link. In Example 8.3, the row is deleted (line 5), the column is deleted (line 10), and the field name is inspected for special characters. In general, the looping variable should be explicitly used inside the loop in some step other than the update. Otherwise, an important step may be missing.

Example 8.3: Pseudocode for preparing a table for ArcGIS import by deleting unneeded rows and columns and repairing field names.

```
1  FUNC preprocessTable (data table, field name row number, first
   invalid column number)
2      OPEN data table.
3      SET row number to 1.
4      WHILE row number < field name row number
5          Delete row.
6          INCREMENT row number.
7      ENDWHILE
8      SET column number to first invalid column number.
9      WHILE column number <= number of columns
10         Delete column.
11         INCREMENT column number.
12     ENDWHILE
13     GET a list of field names in the table.
14     FOR each field name in the list
15         Replace special characters in field name with underscore.
16     ENDFOR
17     Save the data table.
18     Close the data table.
19 ENDFUNC
```

In addition to sequences, decision-making, and repetition structures, code reuse structures, such as functions are often represented in pseudocode. Functions (also known as procedures), group a set of related steps so that they can be re-used by referring to them by name. The first line of Example 8.3 uses the keyword `FUNC`, short for function. This indicates that lines 2–18 make up a function named `preprocessTable`. If your workflow involves modifying multiple tables, you can `CALL` the `preprocessTable` function for each one as shown in Example 8.4.

Example 8.4: Pseudocode for preparing a set of tables for ArcGIS import; For each table, it deletes the first 5 rows and the columns beyond column 50, and repairs the field names.

```
1 GET a list of table names.
2 FOR each table name in the list
3     CALL preprocessTable(table name, 6, 51).
4 ENDFOR
```

Indentation is important in pseudocode to show the relationship of steps to each other and reinforce the block structures. It's also important in Python. Examples 8.1 and 8.4 only have two levels of indentation. The second level is due to their

FOR-loop blocks. Example 8.2 has three levels because the IF...ELSE IF...ENDIF decision-making blocks are nested inside the FOR-loop block. Example 8.3 has three levels of indentation as well; Code is indented inside the FUNC block and indented again inside the loop blocks. A *nested structure* is a block structure that occurs inside of another block structure and requires an additional level of indentation.

Example 8.5 has four levels of indentation. This pseudocode is designed to handle a set of tables each containing the same misspelled field name ('vlue' should be 'value'). The tables are distributed throughout a set of subdirectories, so we need to look in each directory, at each table, and each field name. There are three FOR-loops: one for subdirectories, one for tables, and one for field names. The contents of each loop is indented one more time than the previous one. These three loops are nested because the entities have a hierarchical relationship: each subdirectory has a set of tables and each table has a set of field names. The fourth level of indentation is due to the decision-making block (IF...ENDIF) to check the name of each field and replace it if necessary.

> **Note** Any block structure nested within another block structure needs to be indented. Steps that are outside of a block structure need to be correctly dedented (moved back a notch).

Example 8.5: Pseudocode for correcting a misspelled field name (vlue to value) in each of a set of tables that are distributed through a set of subdirectories.

```
FOR each subdirectory of the current directory
    FOR each table in this subdirectory
        FOR each field in this table
            GET the field name.
            IF the field name is 'vlue' THEN
                SET field name to 'value'.
            ENDIF
        ENDFOR
    ENDFOR
ENDFOR
```

Summing up, here are some rules-of-thumb for describing workflow with pseudocode:

- Decide carefully about what to put inside, outside, before, and after the repetition structure. Should it happen many times or just once?
- Use WHILE-loops for numerical looping variables or when the number of repetitions is unknown ahead of time vs. FOR-loops for a list of items.

- Update the looping variable in the last step of a `WHILE`-loop.
- Use the looping variable at least once inside a `FOR`-loop.
- Use the looping variable at least twice inside a `WHILE`-loop, once before the update.
- Indent steps within a single block structure the same amount to indicate that the steps are related.
- Indent steps within a nested block structure one more time than the previous level of indentation.

Pseudocode provides a convenient tool for sketching the outline of a workflow. Considering the sequential, repetition, decision-making structures provides a frame of reference for breaking the workflow into steps. Exposure to these concepts in general terms should make it easier to pick up the corresponding Python components discussed in the upcoming chapters.

8.2 Key Terms

Workflow structures: sequential steps, repetition, decision-making	`WHILE`-loop
Looping	`FOR`-loop
Branching	Looping variable
Pseudocode	Iterate
Block structures	Functions (procedures, subroutines)
	Nested structures

8.3 Exercises

1. Rewrite the workflows described below in terms of pseudocode. Use appropriate **structural components** (sequences, decision-making, repetition), **keywords** and **indentation**. Answers will vary.

   (a) Get from your living room to your bathroom.
   (b) Check the geometry type of a given shapefile. If the geometry type is polygon, find the area of each polygon and print it. If the geometry type is line, find the length of each line and print it. Otherwise do nothing.
   (c) For all the rasters in a database, perform reclassification.
   (d) The ArcGIS polygon aggregation tool combines polygons that are within the specified distance of each other. For a given shapefile, perform polygon aggregation for aggregation distances of 100 feet, 200 feet, 300 feet, and so forth, up to and including 1000 feet. Use a `WHILE`-loop.
   (e) A KMZ compression format file contains multiple KML files. Unzip a KMZ file and then convert each KML file into a shapefile. Next, find the intersection

of *all* the shapefiles. The final output will be one shapefile containing the intersections of all the shapefiles.

(f) Some of the data in a set of dBASE tables was entered inaccurately. That is, in some records, the park ID is missing. You want to delete these corrupted records. All the tables are sitting together in a single directory. Loop through each record of each table in the directory. Use a `FOR`-loop inside a `FOR`-loop for this nested looping and use two levels of indentation.

2. Step through this pseudocode by hand. Hypothesize what it will print for the input `x` values of 100, 5, 0, and −5. (Zero is an even number, but it is neither positive nor negative.)

```
GET x
WHILE x is less than 100
    IF x is even THEN
        SET x to x + 50
        PRINT x
    ELSE
        IF x is positive THEN
            PRINT x
            SET x to x + 1
        ELSE
            PRINT x
        ENDIF
    ENDIF
ENDWHILE
```

Input	Hypothesized output	Script output
100		
5		
0		
−5		

Next test your answers by running the sample script ‘numGames.py’, which encodes the pseudocode as Python. How did you do? Report if your answers were correct or not and explain any mistakes in your predictions.