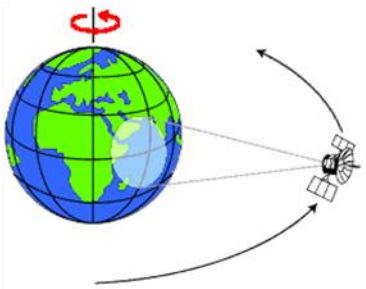
NICHOLAS SCHOOL OF THE
ENVIRONMENT AND EARTH SCIENCES
DUKE UNIVERSITY

Introduction to Scripting: Python 101 – Part 2

ENV 859 – Advanced GIS

Section 2 – Tutorial 2

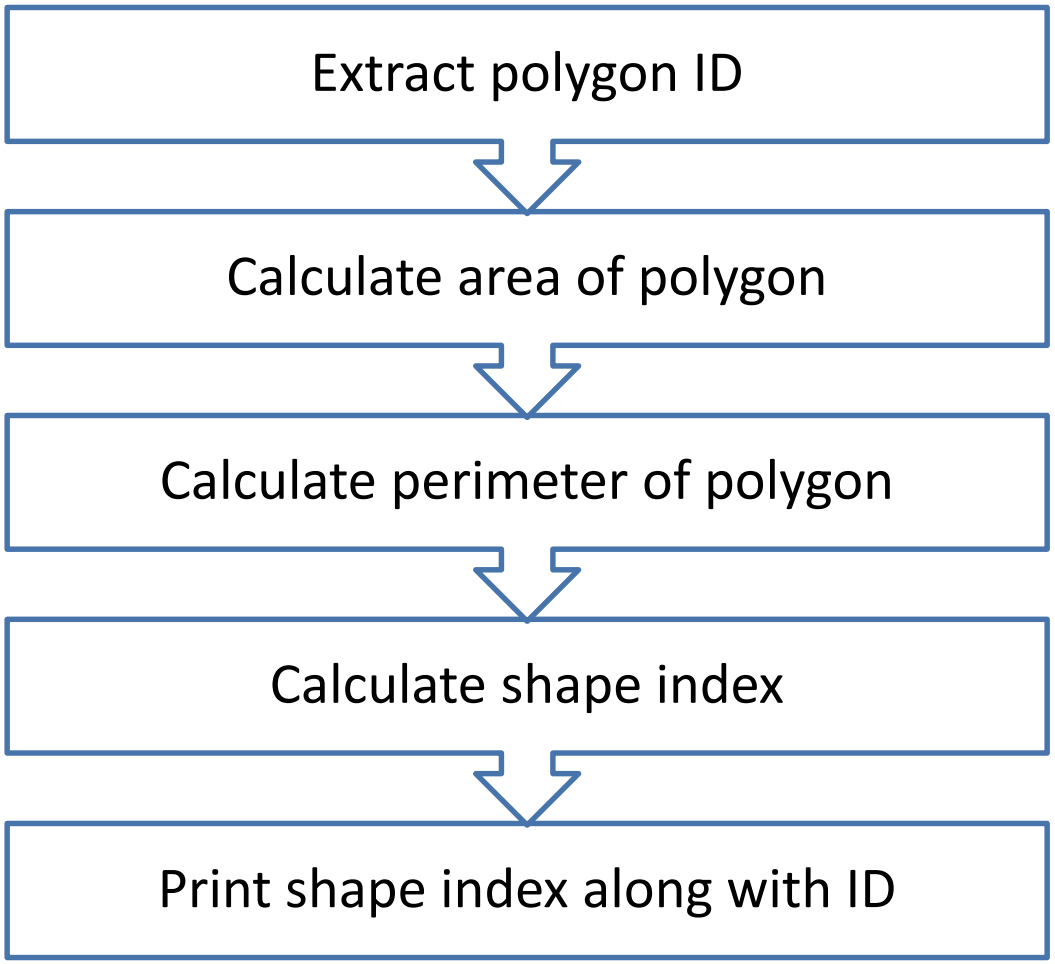


Fay 2015

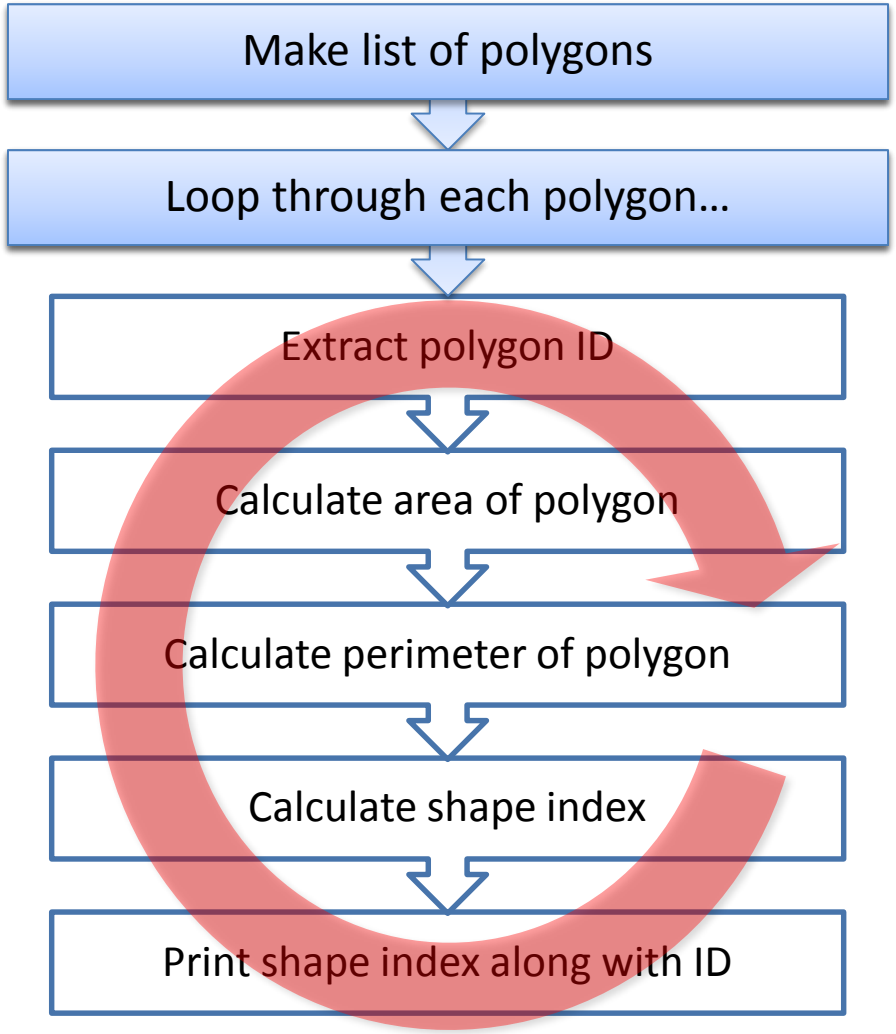
Overview

- What is a script?
- Controlling flow in a script:
  - Iteration with loops
  - Conditional statements
- Simple script inputs and outputs
- Handling script errors

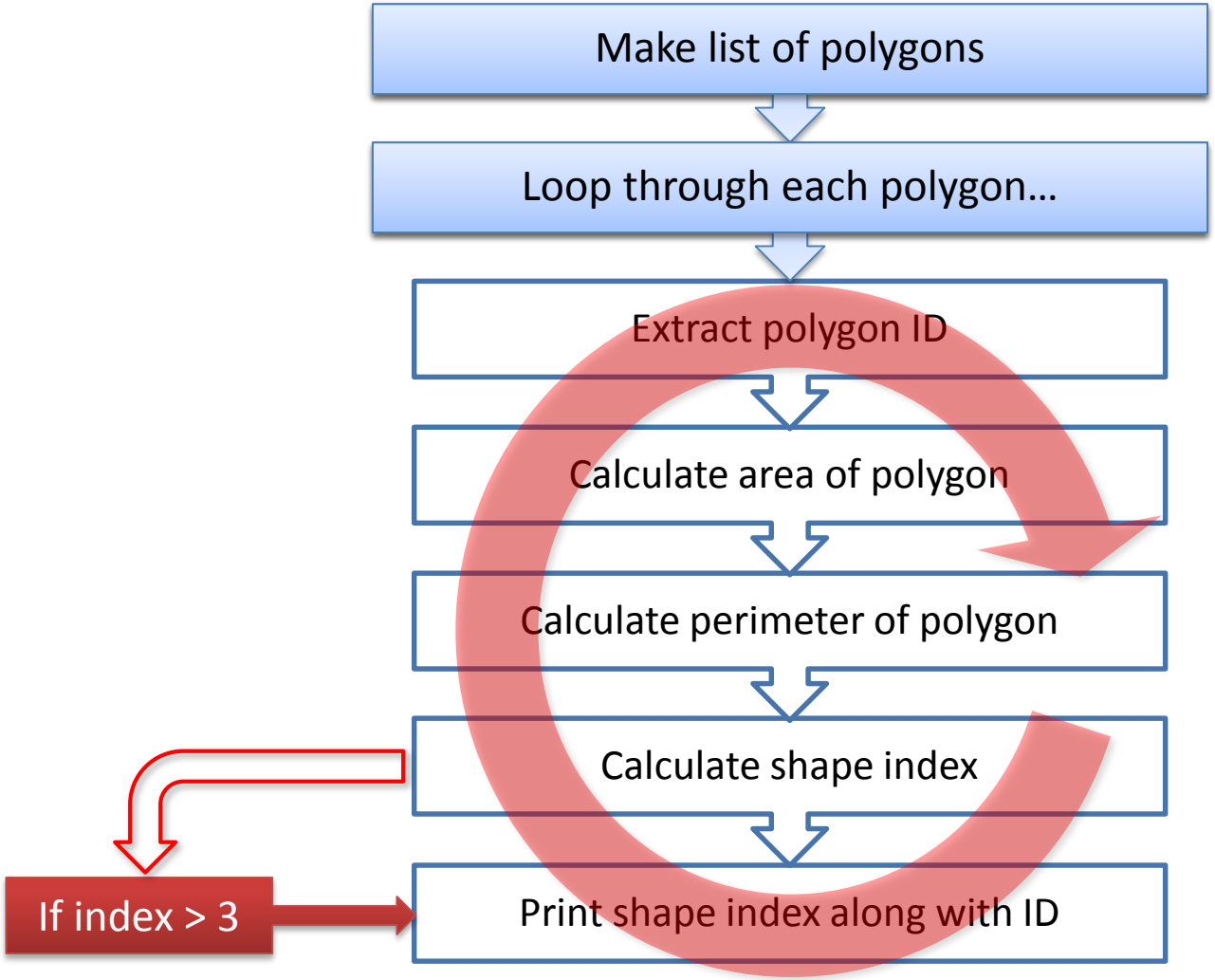
What is a script?



Looping

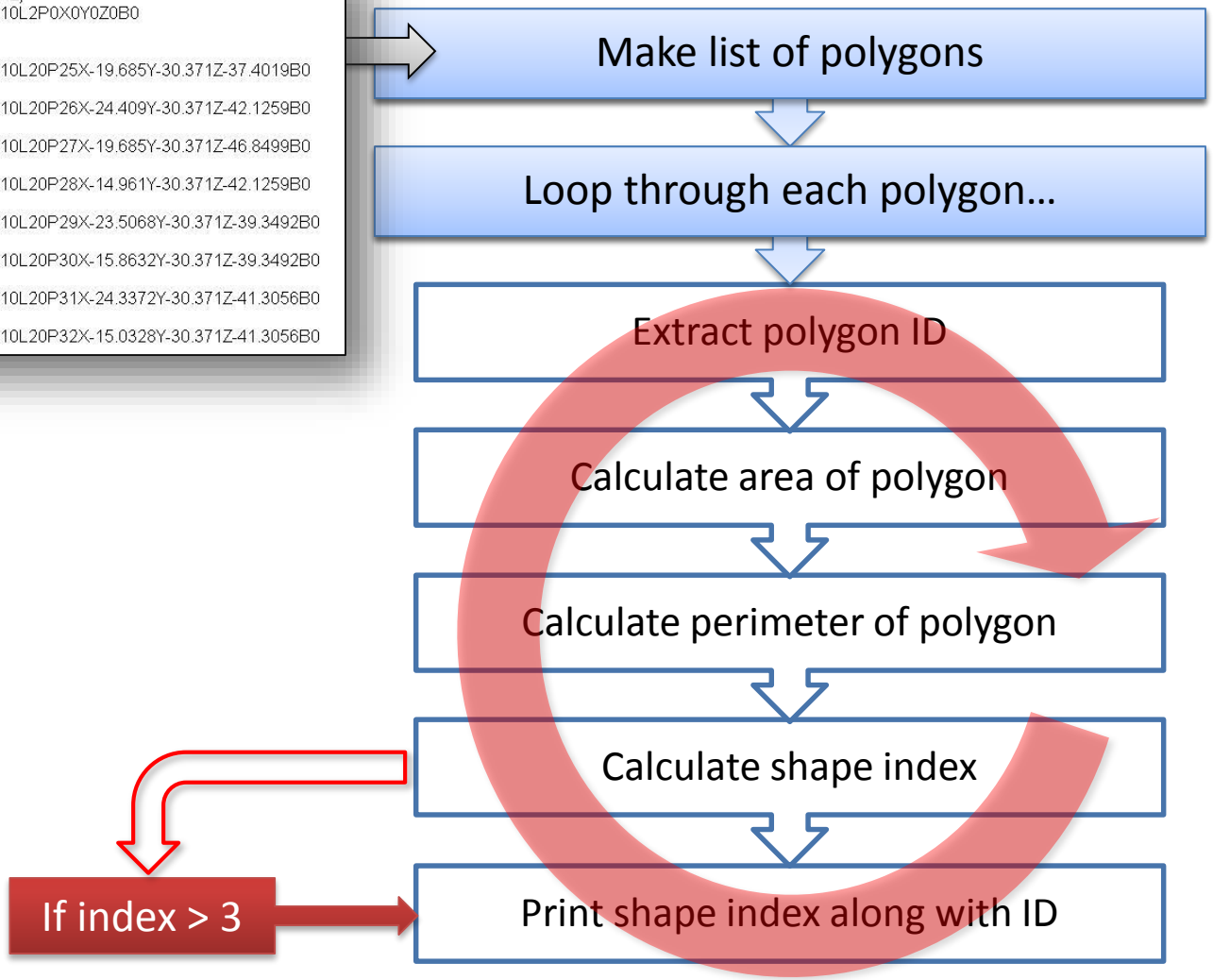

Make list of polygons
Loop through each polygon…
Extract polygon ID
Calculate area of polygon
Calculate perimeter of polygon
Calculate shape index
Print shape index along with ID

Conditional



Accepting input

```
(EXTERNAL)
N10G90G10L2P0X0Y0Z0B0

(B0)
N20G90G10L20P25X-19.685Y-30.371Z-37.4019B0
(B90)
N30G90G10L20P26X-24.409Y-30.371Z-42.1259B0
(B180)
N40G90G10L20P27X-19.685Y-30.371Z-46.8499B0
(B270)
N50G90G10L20P28X-14.961Y-30.371Z-42.1259B0
(B54)
N60G90G10L20P29X-23.5068Y-30.371Z-39.3492B0
(B306)
N70G90G10L20P30X-15.8632Y-30.371Z-39.3492B0
(B80)
N80G90G10L20P31X-24.3372Y-30.371Z-41.3056B0
(B280)
N90G90G10L20P32X-15.0328Y-30.371Z-41.3056B0
```



Writing output

```
(EXTERNAL)
N10G90G10L2P0X0Y0Z0B0

(B0)
N20G90G10L20P25X-19.685Y-30.371Z-37.4019B0
(B90)
N30G90G10L20P26X-24.409Y-30.371Z-42.1259B0
(B180)
N40G90G10L20P27X-19.685Y-30.371Z-46.8499B0
(B270)
N50G90G10L20P28X-14.961Y-30.371Z-42.1259B0
(B54)
N60G90G10L20P29X-23.5068Y-30.371Z-39.3492B0
(B306)
N70G90G10L20P30X-15.8632Y-30.371Z-39.3492B0
(B80)
N80G90G10L20P31X-24.3372Y-30.371Z-41.3056B0
(B280)
N90G90G10L20P32X-15.0328Y-30.371Z-41.3056B0
```

Make list of polygons

Loop through each polygon…

Extract polygon ID

Calculate area of polygon

Calculate perimeter of polygon

Calculate shape index

If index > 3

Print shape index along with ID

```
(EXTERNAL)
N10G90G10L2P0X0Y0Z0B0

(B0)
N20G90G10L20P25X-19.685Y-30.371Z-37.4019B0
(B90)
N30G90G10L20P26X-24.409Y-30.371Z-42.1259B0
(B180)
N40G90G10L20P27X-19.685Y-30.371Z-46.8499B0
(B270)
N50G90G10L20P28X-14.961Y-30.371Z-42.1259B0
(B54)
N60G90G10L20P29X-23.5068Y-30.371Z-39.3492B0
(B306)
N70G90G10L20P30X-15.8632Y-30.371Z-39.3492B0
(B80)
N80G90G10L20P31X-24.3372Y-30.371Z-41.3056B0
(B280)
N90G90G10L20P32X-15.0328Y-30.371Z-41.3056B0
```

Handling errors

```
(EXTERNAL)
N10G90G10L2P0X0Y0Z0B0

(B0)
N20G90G10L20P25X-19.685Y-30.371Z-37.4019B0
(B90)
N30G90G10L20P26X-24.409Y-30.371Z-42.1259B0
(B180)
N40G90G10L20P27X-19.685Y-30.371Z-46.8499B0
(B270)
N50G90G10L20P28X-14.961Y-30.371Z-42.1259B0
(B54)
N60G90G10L20P29X-23.5068Y-30.371Z-39.3492B0
(B306)
N70G90G10L20P30X-15.8632Y-30.371Z-39.3492B0
(B80)
N80G90G10L20P31X-24.3372Y-30.371Z-41.3056B0
(B280)
N90G90G10L20P32X-15.0328Y-30.371Z-41.3056B0
```

Make list of polygons

Loop through each polygon…

Extract polygon ID

Calculate area of polygon

Calculate perimeter of polyg

Calculate shape index

If index > 3

Print shape index along with ID

```
TERNAL)
G90G10L2P0X0Y0Z0B0

)
20G90G10L20P25X-19.685Y-30.371Z-37.4019B0
(B90)
N30G90G10L20P26X-24.409Y-30.371Z-42.1259B0
(B180)
N40G90G10L20P27X-19.685Y-30.371Z-46.8499B0
(B270)
N50G90G10L20P28X-14.961Y-30.371Z-42.1259B0
(B54)
90G10L20P29X-23.5068Y-30.371Z-39.3492B0
N70G90G10L20P30X-15.8632Y-30.371Z-39.3492B0
(B80)
N80G90G10L20P31X-24.3372Y-30.371Z-41.3056B0
(B280)
N90G90G10L20P32X-15.0328Y-30.371Z-41.3056B0
```

Iteration: *For* loops

01_ForLoopExample.py

```
#ForLoopExample.py

# This example uses a for loop to iterate through each item in
# the "fruit" list, updating the value of the "fruit" variable and
# executing whatever lines are indented under the for statement

#Create a list of fruit
fruitList = ("apples","oranges","kiwi","grapes","blueberries")

# Loop through each item in the tuple and execute
# each line that is indented under the for loop
for fruit in fruitList:
    print "I like to eat " + fruit

# Dedented lines are run after the loop completes
print "\nI like ice cream too..."
```

For loops iterate through each item in a collection [list or tuple]

- …lines **indented** under for loop are run for each item
- …the value of the current item is held in the variable specified in the for loop statement (e.g. *fruit* in the above example)

Iteration: *While* loops

01_WhileLoopExample.py

```
#WhileLoopExample.py

# This example demonstrates how a while loop is used. Here, we
# calculate the area of several circle with a radius 'r'. We loop
# through gradually larger values of r until the area of the circle
# exceeds 1000.

pi = 3.1415
r = 1
area = (r ** 2) * pi

while area < 1000:
    print r, area          # Indentation indicates what's run in the loop
    r = r + 1          # The variable that gets evaluated must change in the
    area = (r ** 2) * pi    # loop otherwise you'll create an infinite loop!

print "The while loop is done"  # Dedented lines run after the loop completes
```

Condition

While loops evaluate a condition…

- …the loop begins with the condition followed by a semicolon (:).
- …lines **indented** under while loop are run as long as condition is true
- …the variable evaluated in the condition (area)
  - …must exist before the while loop starts
  - …must change within the loop or else you get an infinite loop

The *range()* function

03_RangeFunctionExample.py

```
#RangeFunctionExample.py

# This example demonstrates the range function for generating
# a sequence of values that can be used in a for loop.

pi = 3.1415

for r in range(0,100,20):
    area = (r ** 2) * pi
    print "The area of a circle with radius ", r, "is ", area
```

The **range()** function creates a list of sequential values.

- …the values in this list can be used within a for loop to iterate some script commands a set number of times.

Controlling execution with *if*

04_IfExample_1.py

```
1   #IfExample.py
2
3   # This example demonstrates how an 'if' statement is used to
4   #  control execution of certain commands. It builds off the for
5   #  loop example, adding a condition so that the message prints
6   #  only when the fruit is a kiwi or blueberries
7
8   #Create a list of fruit
9   fruitList = ("apples","oranges","kiwi","grapes","blueberries")
10
11  # Loop through each item in the tuple and execute
12  # each line that is indented under the for loop
13  for fruit in fruitList:
14      if fruit == "kiwi" or fruit == "blueberries":
15          print "I like to eat " + fruit
16
17
18  # Dedented lines are run after the loop completes
19  print "\nI like ice cream too..."
```

The **if** statement evaluates a specific condition and executes the statements indented underneath only if that statement is true…

Controlling execution with *if* and *else*

04_IfExample_2.py

```
#IfExample.py

# This example demonstrates how an 'if' statement is used to
#  control execution of certain commands. It builds off the for
#  loop example, adding a condition so that the message prints
#  only when the fruit is a kiwi or blueberries

#Create a list of fruit
fruitList = ("apples","oranges","kiwi","grapes","blueberries")

# Loop through each item in the tuple and execute
# each line that is indented under the for loop
for fruit in fruitList:
    if fruit == "kiwi" or fruit == "blueberries":
        print "I like to eat " + fruit
    else:
        print "I don't like " + fruit


# Dedented lines are run after the loop completes
print "\nI like ice cream too..."
```

Adding **else** after the **if** statement allows you to add a group of statements that are run *only* when the condition in the if statement is *false*…

Controlling execution with *if*, elif, and *else*

04_IfExample_3.py

```
#IfExample.py

# This example demonstrates how an 'if' statement is used to
#  control execution of certain commands. It builds off the for
#  loop example, adding a condition so that the message prints
#  only when the fruit is a kiwi or blueberries

#Create a list of fruit
fruitList = ("apples","oranges","kiwi","grapes","blueberries")

# Loop through each item in the tuple and execute
# each line that is indented under the for loop
for fruit in fruitList:
    if fruit == "kiwi" or fruit == "blueberries":
        print "I like to eat " + fruit
    elif fruit == "grapes":
        print "I don't mind " + fruit
    else:
        print "I don't like " + fruit


# Dedented lines are run after the loop completes
print "\nI like ice cream too..."
```

The **elif** clause, short for "*else if*" is used to evaluate an additional condition if the previous one is not met. You can have as many **elif** statements as you wish…

Controlling iteration with ***break***

05_BreakExample.py

```
#BreakExample.py

# This example evaluates a list 10 randomly generated numbers, calculating the
#  square root of each. The break statement is used to catch numbers less than
#  zero, as an error would ensue.

randomNumbers = [81, 8, -17, 2, 9, 0, 9, 4, 16, 64]

# Loop through each day in the tuple of days
for x in randomNumbers:
    # Evaluate the value
    if x < 0:
        print x , "is not a positive number"
        break
    print "The square root of ",x, "is", x ** 0.5

# Dedent lines run after the loop completes
print "Finished processing"
```

The **break** statement halts execution of a loop. When executed, Python skips all lines indented within the loop and proceeds to the next dedented line.

Controlling iteration with ***continue***

05_ContinueExample.py

```
#BreakExample.py

# This example evaluates a list 10 randomly generated numbers, calculating the
#  square root of each. The break statement is used to catch numbers less than
#  zero, as an error would ensue.

randomNumbers = [81, 8, -17, 2, 9, 0, 9, 4, 16, 64]

# Loop through each day in the tuple of days
for x in randomNumbers:
    # Evaluate the value
    if x < 0:
        print x , "is not a positive number"
        continue
    print "The square root of ",x, "is", x ** 0.5

# Dedent lines run after the loop completes
print "Finished processing"
```

The **continue** statement skips all remaining statements *within a loop* and moves to the next iteration…

Reading data into scripts

EnoNearDurham.txt - Notepad2

File Edit View Settings ?

```
# ---------------------------------- WARNING ----------------------------------------
# The data you have obtained from this automated U.S. Geological Survey database
# have not received Direct
# revision.  The data are
# United States Government
# Additional info: http://
#
# File-format description:
# Automated-retrieval info
#
# Contact:   gs-w_support_
# retrieved: 2012-09-21 09
#
# Data for the following 1
#    USGS 02085070 ENO RIV
# ------------------------
#
# Data provided for site 0
#    DD parameter   Descri
#    02   00060     Discha
#
# Data-value qualification
#     P  Provisional data
#
agency_cd site_no datetime
5s  15s 20d 6s  14n 10s
USGS  02085070  2012-08-31
USGS  02085070  2012-08-31
USGS  02085070  2012-08-31
```

06_ReadDataExample.py

```python
#ReadDataExample.py
#
# This example illustrates the use of the file object tor ead in data from a
# text file, extract lines of text, and print them to the screen

# Set the gageDataFile to point to the text file containing the data
#  Note: if no path is specified, Python will look for the file in the same
#  folder as the script. If the file is in a different location, you can
#  specify the entire path, e.g. "S://Scripting1//EnoNearDurham.txt"
gageDataFile = "EnoNearDurham.txt"

# Create a file object by opening the file -- in read only ('r') mode
fileObj = open(gageDataFile,'r')

# Read the first line, setting its contents into the "lineString" variable, then print it
lineString = fileObj.readline()
print lineString

# Or, use a while loop to read each line sequentially and print to the screen
while lineString:
    print lineString
    lineString = fileObj.readline()

# Send a note that that all lines have now been read and printed to the screen
print "Finished!"

#Be sure to close the fileObject so that other applications can access it
fileObj.close()
```

Reading text files in Python

1. *Open* the file in read mode

```
fileObj = open(gageDataFile,'r')
```

"EnoNearDurham.txt"

2. *readlines()* - reads all the lines into a list

```
lineList = fileObj.readlines()
```

3. *readline()* – reads a single line at a time; useful within *while* loops

```
lineString = fileObj.readline()
```

4. *Close* the file to release it for other programs

```
fileObj.close()
```

Writing output to text files

07_WriteDataExample.py

```python
#WriteDataExample.py

# This script demonstrates how a file object is used to both write to a new text file and
#  append to an existing text file.In it we will read the contents of the EnoNearDurham.txt
#  file and write selected data records to the output file.

# First, create some variables. "dataFileName" will point to the EnoNearDurham.txt file,
#  and "outputFileName" will that will contain the name of the file we will create.
dataFileName = "S://Scripting2//EnoNearDurham.txt"
outputFileName = "S://Scripting2//SelectedNWISRecords.txt"

# Next, open the data file and read its contents into a list object. This is similar
#  to the previous tutorial, but here we read the entire contents into a single list
dataFileObj = open(dataFileName, 'r')  ## Opens the data file as read only
lineList = dataFileObj.readlines()     ## Creates a list of lines from the data file
dataFileObj.close()                    ## We have our list, so we can close the file
print len(lineList), " lines read in"  ## This is a quick statement to show how many lines were rea

# Here, we create a file object in 'w' or write mode. This step creates a new file at
#  the location specified. IF A FILE EXISTS ALREADY THIS WILL OVERWRITE IT!
newFileObj = open(outputFileName,'w')

# Next, we loop through the lines in the dataList. If it's a comment, we'll skip it.
#  Otherwise, we'll split the items in the line into a list.
for dataLine in lineList:
    # If the line is a comment, skip further processing and go to the next line
    if dataLine[:4] <> "USGS":
        continue
    newFileObj.write(dataLine)
newFileObj.close()
```

Writing text files in Python

1. *Open* the file in write (or append) mode

```
newFileObj = open(outputFileName,'w')
```

– *Overwrites* any existing file

```
newFileObj = open(outputFileName,'a')
```

– *Adds to* the existing file

```
"S://Scripting2//SelectedNWISRecords.txt"
```

2. *write(*string*)* – writes the string to the file

```
newFileObj.write(dataLine)
```

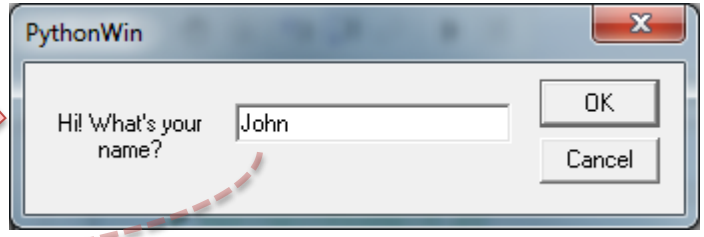
3. Close the file to release it for other programs

```
newFileObj.close()
```

Simple input using **raw_input()**

08_RawInputExample.py

```
# RawInputExample.py

# This script demonstrates the use of the raw_input() function
#  for retrieving user input via a simple popup window. The script
#  asks the user for his or her name and date of birth, and then
#  replies with a simple message.

# Ask for the user's name
userName = raw_input("Hi! What's your name?")

# Generate a prompt for the follow up question
promptString = "Hello, "+userName+". How old are you?"

# Get the birth year, using the prompt string created above
userAge = raw_input(promptString)

# Calculate the birth year of the user. All replies using raw_input() are
#  passed as strings, so we would need to convert then to numbers
birthYear = 2012 - int(userAge)

# Send the reply
reply =  "I'm guessing you were born in " + str(birthYear)
raw_input(reply)
```



The argument supplied with the **raw_input()** function becomes the prompt, and the user reply is stored as a variable…

Handling errors with **try** and **except**

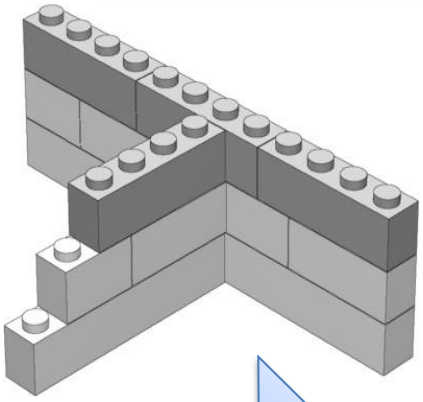
09_HandlingErrorsExample.py

```
# HandlingErrorsExample.py

# This example demonstrates error handling using try-except
#  statements. If an error occurs within the lines indented
#  under the try: statement, Python jumps to the except: statement
#  and exits more gracefully than spewing red text

userNumber = raw_input("Enter a number")

try:
    # Convert the raw input from string to a number
    x = float(userNumber)

    # Calculate the square root of the number
    sqroot_x = x ** 0.5

    # Print the result
    print "The square root of ", x, " is ", sqroot_x

except Exception as e:
    print "An error has occurred"
    print e
```

If any type of error occurs within the **try** section (e.g. if x < 0), then Python skips to the **except** section.

The **try** and except **statements** work together to trap and handle errors

Review



1. Building blocks
2. Building techniques
3. Building actual machines

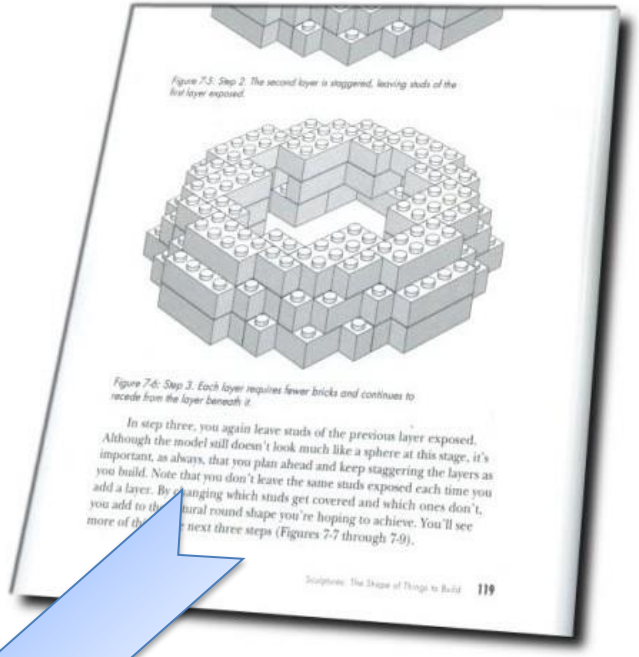


Figure 7-5: Step 2. The second layer is staggered, leaving studs of the first layer exposed.

Figure 7-6: Step 3. Each layer requires fewer bricks and continues to recede from the layer beneath it.

In step three, you again leave studs of the previous layer exposed. Although the model still doesn't look much like a sphere at this stage, it's important, as always, that you plan ahead and keep staggering the layers as you build. Note that you don't leave the same studs exposed each time you add a layer. By changing which studs get covered and which ones don't, you add to the natural round shape you're hoping to achieve. You'll see more of the next three steps (Figures 7-7 through 7-9).

Sculptures: The Shape of Things to Build 119