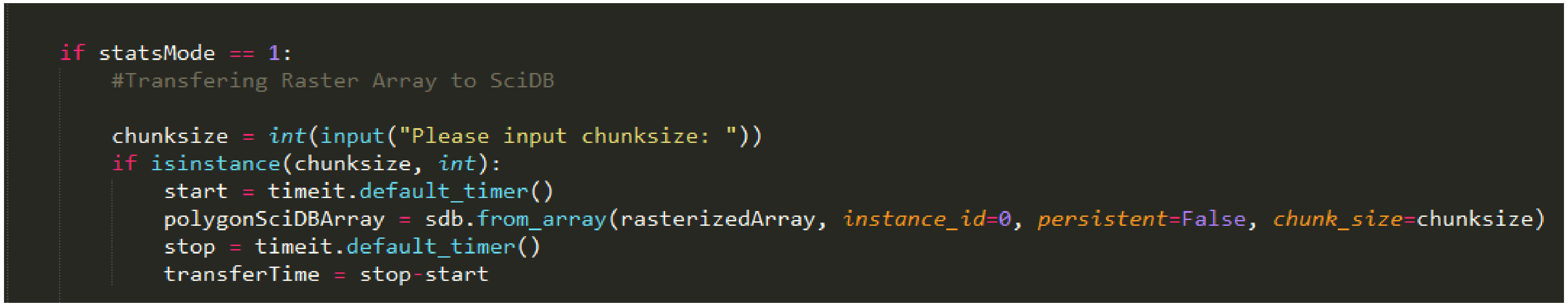
WEEK 3

Control Structures

Files (Open & Close)

Python

- White space sensitive
- Indentation sensitive



```
if statsMode == 1:
    #Transfering Raster Array to SciDB

    chunksize = int(input("Please input chunksize: "))
    if isinstance(chunksize, int):
        start = timeit.default_timer()
        polygonSciDBArray = sdb.from_array(rasterizedArray, instance_id=0, persistent=False, chunk_size=chunksize)
        stop = timeit.default_timer()
        transferTime = stop-start
```

CONTROL STRUCTURES

Learning control structures

- If – Then - Else
  - Very few programs are straight forward
  - Control structures allow you to define options

- Loops
  - for loops
  - while loops

If – Then - Else

- #states that b must equal the value for the following statements to be executed
- if b == 0:
  - pass
  - #Do something

- #States that if c is true, then do something. Any value that is not zero will evaluate as true
- if c:
  - pass
  - print(“hello class”)

Code It!

- Words to test = garage, python
- Ask the user to input some text
- If the text the user inputs has an ‘a’ then upper case
- If the text the user inputs has a ‘e’ then lower case

- If the text the user inputs has an a and the last character is an ‘e’, replace it with ‘ers’

Loops

- While loops
  - Start with an initial true condition
  - Run this loop (potentially forever until the condition changes)

```
b = 0
while b < 100:
    print(b)
```

While loops

- Potentially problematic
  - If the condition starts as false, then it never runs.
  - You might want this to run.
  - You never ever evaluate to false so it never leaves the code block

For loops

- A safer alternative to while loops.
- Execute at least once
- Always terminate
- Can be combined with iterators and sequences

```
for b in range(100):
    print(b)
```

Code It

- Multiplication table 0-12

Comprehension

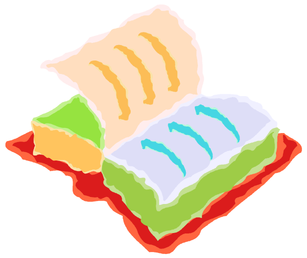
```
catchlist = []
for s in stuff:
        catchlist)
```

- `Catchlist = [s for s in stuff]`

BREAK

FILES

File Management in Python



- Write data to a file to store it permanently

- Files are digital books:
  - a book is a permanent repository of data
  - to read a book, you have to open it
  - either read from it or write in it
  - when done, close it

open -> read / write -> close

Opening & Closing a file

- Open a file: open
- Close a file: close
- There are a few things you can perform in between
- Always close the file when done.
    - Otherwise you may lose data
    - Other programs will also not be able to access the file.

Opening & Closing a file

- Open a file:

```
newfile = open(filepath,r/w)
```

- Close a file:

```
newfile.close()
```

- Always close the file when done.

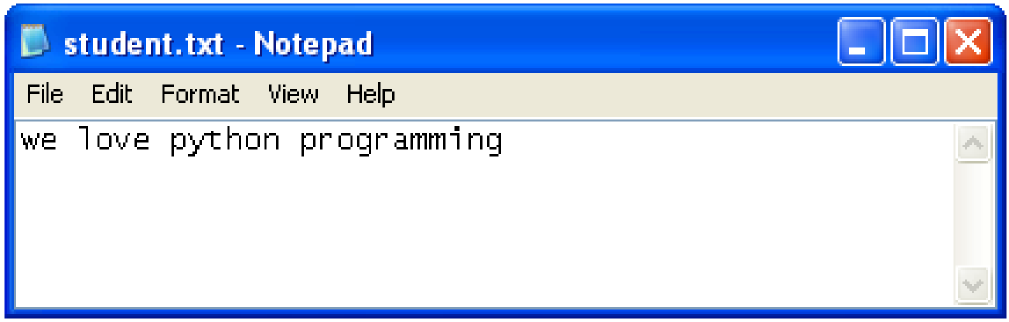
- Use ‘WITH’

The advantage of using a with statement is that it is guaranteed to close the file no matter how the nested block exits. If an exception occurs before the end of the block, it will close the file before the exception is caught by an outer exception handler. If the nested block were to contain a return statement, or a continue or break statement, the with statement would automatically close the file in those cases, too.

Opening a file for writing

- If file does not exist, it will be automatically created

```
myFile = open(r“C:\work\student.txt", 'w')
#The default method when opening a file is “r”
myFile.write("we love python programming”)
myFile.close()
#If file exists, it will be overwritten
```



```
we love python programming
```

Python 2

We had a string object that consisted of strings and bytes

Python 3

The string object has been redefined, removing the bytes.

There is now a true bytes object.

If you want to write in binary you need to convert the information to the appropriate format.

CODE IT

```
f = open(r"C:\3D.txt", "w")
#your strings must have the new line character
present
f.write("length\nheight\nwidth\n")
f.close()
# now open the file in windows...
# and now read back the content:
```

Code IT

- Exercise 1 (*.txt)
  - a string : ‘your name’
  - A variable : Address = ‘your address’
  - Close the file
- Exercise 2 (*.txt)
  - Write a list of numbers (1,10001) with each number on a new line
  - Write a list of numbers (1,10001) all on the same line
- Exercise 3 (*.csv)
  - Write a multiplication table (12x12)

Python 2

```
outFilePath = r"C:\work\outdata.csv"
dataset = [[1,1,1,1,1], [5,5,5,5,5], [3,3,3,3,3]]

with open(outFilePath, 'w') as outCSV:
    for d in dataset:
        print("This is the element %s" % (d) )

        elementAsList = str(d)[1:-1]
        print(elementAsList)
        #This function only writes things that
are strings.
        #Format your string perfectly
        outCSV.write( "%s\n"  % (str(d)[1:-1]) )
```

Python 3

```
outFilePath = r"C:\work\outdata4.csv"
dataset = [[1,1,1,1,1], [5,5,5,5,5], [3,3,3,3,3]]

with open(outFilePath, 'w', newline="\n") as
outCSV:
    for d in dataset:
        print("This is the element %s" % (d) )

        elementAsList = str(d)[1:-1]
        print(elementAsList)
        #This function only writes things that
are strings.
        #Format your string perfectly
        outCSV.write( "%s\n"  % (str(d)[1:-1]) )
```

CSV MODULE

Writing

Python 2

```
#importing the CSV module
import csv

outFilePath = r"C:\work\outdata3.csv"
dataset = [[1,1,1,1,1], [5,5,5,5,5], [3,3,3,3,3]]

#On windows you need to specify the mode= 'wb'
#Or you will get irregular spacing


with open(outFilePath, 'wb') as outCSV:
    theWriter = csv.writer(outCSV, delimiter=",")
    for d in dataset:
        print("This is the element %s" % (d) )

        #This CSV module will write all of the
elements in the list
        #It will do all of your formatting, just
get your data in a list format
        theWriter.writerow(d)
```

Python 3

```
#importing the CSV module
import csv

outFilePath = r"C:\work\outdata5.csv"
dataset = [[1,1,1,1,1], [5,5,5,5,5], [3,3,3,3,3]]

#If you choose the mode = 'wb' you must write a bytes
object
#If you don't specify the newline option in Windows you
get irregular spaces
with open(outFilePath, 'w', newline="\n" ) as outCSV:
    theWriter = csv.writer(outCSV, delimiter=',')
    for d in dataset:
        print("This is the element %s" % (d) )

        #This CSV module will write all of the elements
in the list
        #It will do all of your formatting, just get
your data in a list format
        theWriter.writerow(d)
```

Opening a file for reading

- File for reading opens only if it exists, otherwise error

```
myFile = open("student.dxt", 'r')
Traceback (most recent call last):
  File "<interactive input>", line 1, in ?
IOError: [Errno 2] No such file or directory:
  'student.dxt'
```

- There are several ways of reading a file
- *.read()   reads entire file
- *.read(5) reads first five characters
- *.readline()   reads one line
- *.readlines()   reads all lines; puts them into a list of strings

Opening a file for reading

```
>>> myFile = open("student.txt", 'r')
>>> myFile.read()
'we love python‘
>>> myFile.read()
'‘  #??
```

Python “remembers” where you finished reading;
if end of file reached, returns an empty string

Standard

```
f = open(r"C:\3D.txt", "r")
#data is lost
f.readlines()
['length\n', 'height\n',
'width\n']
f.close()
```

Alternative

```
f = open(r"C:\3D.txt", "r")
#data is saved to var(content)
content = f.read()

f.close()


print(content)
length
height
width
```

Standardizd Reading

Python 2

```
f = open(r"c:\work\outdata.csv", "r", 
newline='\n')

for line in f:
    print(line)
```

Python 3

```
f = open(r"c:\work\outdata.csv", "r", 
newline='\n')

for line in f:
    print(line)
```

Reading data in a loop

Python 2

```
theFilePath = r"c:\work\outdata.csv"
f = open(theFilePath)

dataset = f.read()
for d in f.read().splitlines():
            print(d)
```

Python 3

```
theFilePath = r"c:\work\outdata.csv"
f = open(theFilePath)

dataset = f.read()
for d in dataset.splitlines():
            print(d)
```

CSV MODULE

Reading

Python 2

```
#importing the CSV module
import csv

with open(r"c:\work\outdata5.csv", "rb") as fin:
    #The mode "r" or "rb" doesn't matter
    theCSV = csv.reader(fin, delimiter=',')
    for line in theCSV:
        print(line)
```

Python 3

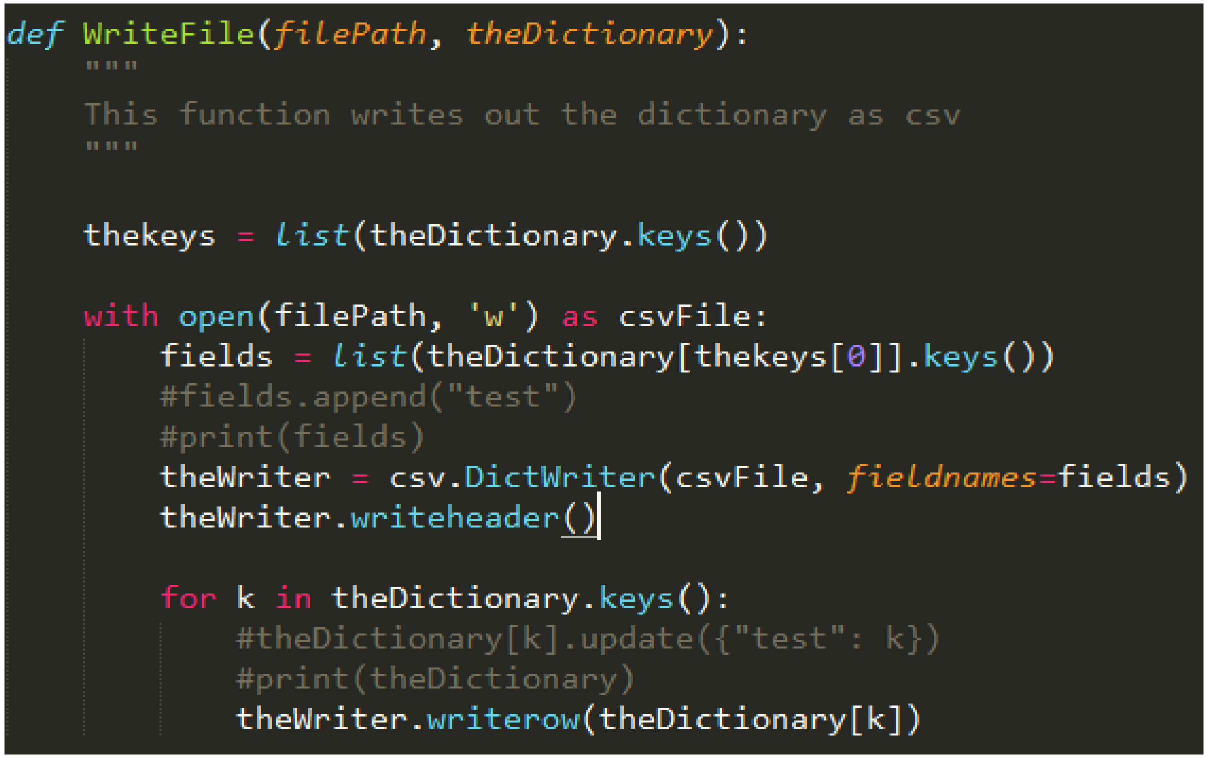
```
#importing the CSV module
import csv

outFilePath = r"C:\work\outdata5.csv"
with open(outFilePath "r") as fin:
    theCSV = csv.reader(fin, delimiter=',')
    for line in theCSV:
        print(line)
```

CSV Module

- List
  - Writer
  - Reader
- Dictionary
  - Writer
  - Reader

CSV Module

- Very good module that is incredibly and flexible and powerful
- Tricky at times but you can write very generalizable code



```
def WriteFile(filePath, theDictionary):
    """
    This function writes out the dictionary as csv
    """

    thekeys = list(theDictionary.keys())

    with open(filePath, 'w') as csvFile:
        fields = list(theDictionary[thekeys[0]].keys())
        #fields.append("test")
        #print(fields)
        theWriter = csv.DictWriter(csvFile, fieldnames=fields)
        theWriter.writeheader()

        for k in theDictionary.keys():
            #theDictionary[k].update({"test": k})
            #print(theDictionary)
            theWriter.writerow(theDictionary[k])
```

Spatial Data Management Nightmare

- A client gives you a geodatabase with important information on it. However, they don’t know the which shapefile or field has the important information. They think they might know the name of it if they saw it. How could you handle this?

- Geodatabase:
  - Shapefiles
    - Each has different attributes