Why Python for Data Analysis?

School of Information Studies
Syracuse University

Data Analysis

Nuts and Bolts of Data Analysis

- Manipulating
- Processing
- Cleaning
- Crunching

Types of Data

Multidimensional arrays

Tabular or spreadsheet data

Multiple tables related by key columns

Evenly or unevenly spaced time series

Why Python?

Easy to use

Most popular dynamic languages

Distinguished by active scientific computing community

Significant growth since 2000

Excellent choice for data-centric applications

Libraries Used

NumPy—used for scientific computing

Pandas—makes working with structured data very fast

Matplotlib—helps produce 2D data visualizations

Ipython—ties everything together

SciPy—collection of packages

- Scipy.integrate—numerical integration routines
- Scipy.linalg—linear algebra routines
- Scipy.optimize —function optimizers (minimizers)
- Scipy.stats—probability distributions

So Why Python?

General-purpose programming language

- Beautifully designed
- Intuitive
- Exceedingly powerful

Python Is…

Real

A general-purpose programming language

Used by Google and Dropbox in core applications

Building Blocks of Python

School of Information Studies
Syracuse University

Program Components

Input: outside world

Output: What are the results we want?

Sequential execution: one step at a time

Conditional execution: What conditions?

Looping: repetitive operations

Subroutines: reusing code

Python Variables

Numbers

Strings

Values

Using variables

Variables: Numbers

```
>>>number = 7
>>>number                          (displays its value)
7
>>>average = (number +9)/2         (standard math ops)
>>>average                         (displays its value)
8
```

Variables: Strings

```
>>>string1 = ‘Monty’
>>>string2 = ‘Python’
>>>string1 + string2        (‘MontyPython’)
```

Variable Types

Types can matter.

- Type function – type (number1)

Name variables to be meaningful.

- Letters, numbers, underscore
- No keywords

Print Function

Python displays value

Explicit PRINT function

Print (balance)	Result – 125
Print (“The balance is “, balance)	Result – The balance is 125.

User Input

Getting data from the user—type is a string

```
>>>username = input ('What is your name? ')
What is your name?  Tom
>>>print ('Hello, ',username)
Hello, Tom
>>>year = input ('What year were you born? ')
What year were you born?  1989
>>>print ('BirthYear:  ',year)
Birthyear:  1989
```

Using Input in Calculations

Must convert to numeric type

```
>>>type (year)
<class 'str'>
>>>age=2018-int (year)
>>>print ('Age: ',age)
```

Booleans and Conditional Statements

School of Information Studies
Syracuse University

Boolean Expressions

Either TRUE or FALSE

Comparison operator (==)

>>>5==5 OR >>> 5 is 5

True

>>>5==6 OR >>> 5 is 6

False

Other Comparisons

Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Not equal to	!=

Logical Operations

And, or, not

```
>>>num1 = 5
>>>(num1 >= 0) and (num1 < 10)      Response – True
>>>(num1 >= 0) and (num1 < 4)       Response – False

>>>(num1 >= 0) and not(num1 <10) Response – False
```

Conditional Statements

If-then statements

OR

If-then-else statements

Allow us to execute code based on conditions

Conditional Code

If statement ending in :

- Indent the code to be executed
- End with an extra ‘enter’

```
>>>num1 = 5
>>>if num1<10:
…        print (‘Less than 10’)
…
Less than 10
>>> (next line of code)
```

Conditional Statement With Else

```
>>>If num1 < 10:
…       num2 = num1 *10
…    else:
…        num2 = num1
…
>>>print (num2)
50
```

Multiple Conditions

Multiple if–else–else statements

```
>>>if num1 <5:
…       print (‘Less than 5’)
…    else:
…       if num1 == 5:
…          print (‘Equal 5’)
…       else:
…          print (‘Greater than 5’)
…
Equal 5
```

Multiple Conditions

If the conditions have an else followed by if, abbreviate it as ‘elif’

```
>>>if num1 < 5:
...         print (‘Less than 5’)
...     elif num1 == 5:
...         print (‘Equal 5’)
...     else:
...         print (‘Greater than 5’)
```

Lists and Simple Loops

School of Information Studies
Syracuse University

What Are Lists?

A way to group data

Alist = [‘ant’, ‘bee’, ‘car’, ‘dog’]

Nlist = [5, 7, 9]

Emptylist = []

Mixlist = [‘ant’, 7, 9, 6]

Len(alist) number of items

Indexing a List

Len(alist) is 4

Refer to an individual entry:

Alist[0] = ‘ant’

Alist[1] = ‘bee’

Alist[2] = ‘car’

Alist[3] = ‘dog’

Combining Lists

>>>blist = [‘egg’, ‘fox’]

>>>alist + blist

[‘ant’, ‘bee’, ‘car’, ‘dog’, ‘egg’, ‘fox’]

>>>alist and blist are still the same

>>>alist.extend (blist)

Append one item

>>>alist.append (‘goo’)

Simple Loops

For loop – iteration over elements

```
>>>for elem in alist:
…       print (elem)
…
```

OR

```
>>>for elem in alist:
…       print (‘Label: ‘,elem)
…
```

Building a List

Use one list to build another list

```
>>>clist = []
>>>for elem in blist:
…      newstring = ‘c:’ + elem
…      clist.append (newstring)
…
```

Range Function

Produces numbers from 0 to N-1

```
>>>dlist = [ ]
>>>for index in range (10):
…      dlist.append (index)
…
```

List Slices

Slice is a portion of the list

>>>alist[1:4]

Gives the index numbers of 1, 2, and 3

>>>alist[:4]

Gives the index numbers of 0, 1, 2, and 3

>>>alist[4:]

Gives the index numbers starting with 3 and going to end

Another Boolean Comparison

To test if an item in a list

>>> ‘egg’ in alist

>>> ‘foo’ in alist

List Comprehensions

Uses outer brackets with a list element

```
>>>blist
>>>elist = [‘e:’+elem for elem in blist]
>>>elist
>>>flist = [index for index in range(5)]
>>>flist
```