Introduction to Python

- Interpreter (www.python.org)
- Editor (idle – run programs within editor)
- Need **NumPy** for COSC 370/377
- Lectures based on Python 3
- Script files begin with lines such as

```
#!/opt/local/bin/python3.3
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

§1.2 Core Python

- Variables are dynamically typed.

```
>>> a=1              # a is an integer
>>> print(a)
>>> a=a+1.5          # what is a now?
>>> print(a)
```

Strings

- Sequence of chars enclosed by single or double quotes; concatentate using "+" and slice with ":"

```
>>> str1 = 'Fred'
>>> str2 = 'Wilma'
>>> print(str1 + ' and ' + str2)

>>> print(str1[0:2])
```

Strings

- Strings are immutable objects; have fixed length.

```
>>> s = ‘My name is John’
>>> s[0] = ‘T’ # What happens?
```

Tuples

- Sequence of arbitrary (but immutable) objects; supported by similar string operations.

```
>>> record = ('Doe','John',(5,15,75))
>>> last,first,bdate = record
>>> print(first)

>>> byear = bdate[2]
>>> print(byear)
```

Tuples

```
>>> record = ('Doe','John',(5,15,75))
>>> name = record[1] + ' ' + record[0]
>>> print(name)

>>> print(record[0])

>>> print(record[0:1])

>>> print(record[0:2])
```

Lists

- Similar to tuples but are mutable; elements and length can be changed.

```
>>> list = [1.0,2.0.3.0]
>>> list.append(4.0)
>>> print(list)

>>> list.insert(0,0.0)
>>> print(list)
```

Lists

```
>>> list = [0.0,1.0,2.0,3.0,4.0]
>>> print(len(list))

>>> list[1:2]=[3.0,3.0]
>>> print(list)
```

- For a mutable object, `y = x` creates a new reference to `x` (not a copy).

Lists

- To get a new copy, use `z = x[:]`

```
>>> x = [1.0,2.0,3.0]
>>> y = x
>>> y[0]=2.0
>>> print(x)

>>> z=x[:]
>>> z[0]=4.0
>>> print(x)
```

Matrix Representation

```
>>> matrix = [ [1,2,3], \
               [4,5,6], \
               [7,8,9] ]

>>> print(matrix[1])
>>> print(matrix[1][2])
```

Arithmetic Operations

- Possibilities: `+,-,*,/,**,%`

```
>>> str = ‘yada ‘
>>> print(3*str)

>>> a =[3,4,5]
>>> print(3*a)

>>> print(a+[6,7])
```

Arithmetic Operations

```
>>> str  = ‘yada ‘
>>> str2 = ‘for you’
>>> print(str + str2)

>>> print(4 + str2)  # What happens?
```

Augmented Assignments

- Shorthand (similar to C):

```
a+=b   # a=a+b
a-=b   # a=a-b
a*=b   # a=a*b
a/=b   # a=a/b
a**=b  # a=a**b
a%=b   # a=a%b
```

Comparison Operators

- Possibilities: `<,>,<=,>=,==,!=`

```
>>> a=3
>>> b=2.99
>>> c='3'
>>> print(a>b)
>>> print(a==c)
>>> print( (a>b) and (a!=c))
>>> print( (a>b) or (a==b))
```

Conditionals

```
def sign_of_a(a):
    if a < 0.0:
        sign = 'neg'
    elif a > 0.0:
        sign = 'pos'
    else:
        sign = 'zero'
    return sign

>>> a = 2.0
>>> print('a is ' + sign_of_a(a))
```

Loops

```
nMax=4
n=1
a=[] # empty list
while n < nMax:
        a.append(1.0/n)
        n=n+1
print(a)
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Loops

```
nMax=4
a=[] # empty list
for n in range(1,nMax):
        a.append(1.0/n)
print(a)
```

- Any loop can be terminated by a `break`.

Loops

```
list=['a','b','c','d']
bound=len(list)
char_input=eval(input('Type a char: '))
for i in range(bound):        # What does range return?
    if list[i] == char_input:
       print(char_input,' is in
            position ',i+1,' in list')
       break
    else:
       if i == bound-1:
          print(char_input,' is not in list')

>>>Type a char: 'a'

>>>Type a char: 'e'
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Type Conversions

- Built-in functions: `int(a),long(a),float(a), complex(a),complex(a,b)`

```
>>> a=6
>>> b=-2.5
>>> c='4.0'

>>> print(a+b)
>>> print(int(b))          # truncation
>>> print(complex(a,b))
>>> print(float(c))
>>> print(int(c))
```

Useful Mathematical Functions

- Absolute value: `abs(a)`
- Max/min: `max(seq), min(seq)`
- Rounding: `round(a,n)`
- Comparison: ~~`cmp(a,b)`~~ removed in Python 3

Reading Input

- Use **`input()`** function to get data from user that is converted to a string; **`eval()`** converts to numerical value.

```
>>> a = input('Input a: ')
>>> print(a,type(a))

>>> b = eval(a)
>>> print(b,type(b))
```

Printing Output

- The **`print()`** function coverts all inputs to strings for output.

```
>>> a = 123.456
>>> b = [1,2,3,4]
>>> print(a,b)

>>> print('a=',a,'\nb=',b)
```

Formatted Output

- Common formats: **`wd`**, **`w.df`**, **`w.de`**
  `w`=width; **`d`**=no. of digits; **`f`**/**`e`**= floating pt. or exponential notation.

```
>>> a = 123.456
>>> n = 5789
>>> print('{:6.2f}'.format(a))

>>> print('n={:7d}'.format(n))
```

Formatted Output

```
>>> a = 123.456
>>> n = 5789
>>> print('n={:07d}'.format(n))


>>> print('{:11.3e}{:7d}'.format(a,n))
```

Error Control

- Not specifying a specific **error** causes all exceptions to be caught.

```
try:
    [code]
except error:
    [code]
```

```
try:
    a=0
    c=b/a
except ZeroDivisionError:
    print('Divide by zero')
```

§1.3 Functions

- Function syntax:

```
def fname(parm1,parm2,…):
    [statements]
    return value(s)
```

```
from math import arctan
def finite_diff(f,x,h=0.0001):
  df=(f(x+h)-f(x-h))/(2.0*h)
  ddf=(f(x+h)-2.0*f(x)+f(x-h))/h**2
  return df,ddf
```

Functions

```
>>> x = 0.5        # What about h?
>>> df,ddf=finite_diff(arctan,x)

>>> print('First deriv. =',df)
>>> print('Second deriv. =',ddf)
```

Functions

- What happens when a *mutable* object is passed to a function?

```
def squares(a):
    for i in range(len(a)):
        a[i]=a[i]**2

>>> a = [1,2,3,4]
>>> print(squares(a))
```

Lambda Functions

- Take an expressions and create a function using it; enables *lists* of actions.

```
>>> c = lambda x,y: x**2 + y**2
>>> print(c(3,4))
```

§1.4 Modules

- Modules specify the files containing functions.

```
from mod_name import *
```

- Use **import** to get the functions of a module loaded.

```
>>> import math
>>> dir(math)
```

- Need **cmath** for functions of complex numbers; **numpy** for most functions needed in this course.

§1.5 Numpy and Arrays

- Useful functions in **numpy**:

```
zeros((dim1,dim2),dtype=spec)
 ones((dim1,dim2),dtype=spec)
arange(from,to,increment) # returns array
```

```
>>> from numpy import array,float
>>> a=array([[2.0,-1.0],[-1.0,3.0]])
>>> print(a)

>>> b=array([[2,-1],[-1,3]],dtype=float)
>>> print(b)
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Arrays

```
>>> from numpy import arange,zeros,ones
>>> a=arange(2,10,2)
>>> print(a)

>>> b=ones((3,3),dtype=float)
>>> print(b)
```

Arrays

- Accessing/changing array elements:

```
>>> from numpy import *
>>> a=zeros((3,3),dtype=float)
>>> a[0]=[2.0,1.1,1.5]   # change row
>>> a[1,1]=5.2           # change element
>>> a[2,0:2]=[8.0,-3.3]  # change slice
>>> print(a)
```

Array Operations

- **`array`** function provided by Numpy.

```
>>> from numpy import array,sqrt
>>> a=array([0.0,4.0,9.0,16.0]
>>> print(a/16.0)

>>> print(a - 4.0)

>>> print(sqrt(a))  # Not sqrt from math
```

Array Functions

- Linear algebra made easy:

```
>>> from numpy import *
>>> a=array([[ 4.0,-2.0, 1.0],\
             [-2.0, 4.0,-2.0],\
             [ 1.0,-2.0, 3.0]])
>>> b=array([1.0,4.0,2.0])

>>> print(dot(b,b)) # dot product

>>> print(dot(a,b)) # matrix-vector product
```

Array Functions

```
>>> from numpy import *
>>> a=array([[ 4.0,-2.0, 1.0],\
             [-2.0, 4.0,-2.0],\
             [ 1.0,-2.0, 3.0]])
>>> b=array([1.0,4.0,2.0])

>>> print(multiply(a,b))   # element-wise product


>>> print(diagonal(a))     # principal diagonal

>>> print(diagonal(a,1))   # first subdiagonal

>>> print(trace(a))        # sum of diagonal elements
```

Array Functions

```
>>> from numpy import *
>>> a=array([[ 4.0,-2.0, 1.0],\
             [-2.0, 4.0,-2.0],\
             [ 1.0,-2.0, 3.0]])
>>> b=array([1.0,4.0,2.0])

>>> print(argmax(b))   # index of largest element

>>> print(identity(3)) # identity matrix


>>> c=a.copy()         # independent copy (not alias)
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

§1.6 Variable Scoping

- **Namespace** – dictionary of variable names and their values.
  Three levels:

  1) **Local** – created when a function is called (parameters are passed and variables are created within the function); deleted when the function terminates; variables created inside functions only have that function's local namespace.

  2) **Global** – created when a module is loaded; variables assigned in this namespace are visible to all functions in the module.

Variable Scoping

3) **Built-in** – created when the python interpreter starts; variables assigned are visible to all program units.

- Resolution of a variable:

  1) Check local namespace
  2) Check global namespace
  3) Check built-in namespace
  4) Issue NameError exception.

Variable Scoping

- Variables created in the global namespace **do not** have to be passed to functions as arguments – good practice anyway.

```
def divide1():
    c=a/b
    print('a/b ='c)

>>> a=100.0
>>> b=5.0
>>> divide1()

>>> divide2()
>>> print('a/b =',c)
```

```
def divide2():
    c=a/b
```

§1.7 Running Programs

- For script files (e.g., `myprog.py`), you can execute the code therein via

```
unix> python myprog.py
```

- You can drop `python` at the `unix` prompt if you have your `python` path on the first line of `myprog.py`: **`#!/opt/local/bin/python3.3`**

Running Programs

- Byte code of a module is written to a file with the `.pvc` file extension; the interpreter will always look for a module's `.pvc` file first.
- Before executing your script file, it is a good practice to add the line **`input('Press return')`** to the end of your code so that the program window (launched on execution) does not close when the program terminates – can be an issue on Windows-based machines.

Running Programs

- Error messages can be slightly confusing…

```
from numpy import array
a=array([1.0,2.0,3.0]
print(a)
Input('Press return')
```

Output:

```
File:"C:\Python22\test_module.py", Line 3
print(a)
     ^
SyntaxError: invalid syntax
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.

Module Documentation

- Another good practice is to add a **docstring** to the beginning of your module:

```
## module my_error
'''my_error(string).
   Prints 'string' and terminates.
'''
import sys
def my_error(string):
   print(string)
   input('Press return to exit')
   sys.exit()
```

Module Documentation

- How can the user print the docstring?

```
>>> import my_error
>>> print(my_error.__doc__)
    my_error(string).
    Prints 'string' and terminates.
```

File I/O

- How can the user open/close files?

```
file_object = open(filename,action)
file_object.close()
```

```
>>> data = open('sunspots.txt','r')
```

```
'r'       read
'w'       write (create if need be)
'a'       append to EOF
'r+'      read/write to existing file
'w+'      same as 'r+' but create file if
          needed
'a+'      same as 'w+' but append to EOF.
```

File I/O

- Extracting intensity data from the year/month/date/intensity columns in `sunspots.txt` (see p. 14 of textbook).

```
x=[] #empty list
data = open('sunspots.txt','r')
for line in data:
   x.append(eval(line.split()[3]))
data.close()
```

THE UNIVERSITY of TENNESSEE KNOXVILLE
Big Orange. Big Ideas.