python

Programming with Python 2

for Non-programmers

Babar Ali

Topics

- The python interpreter
- Data & Variables
  - Types and format of variables.
- Data & Variables in numpy
  - Basic concepts
  - Creating and manipulating variables in numpy
- Math operations & functions

Introducing

THE PYTHON INTERPRETER

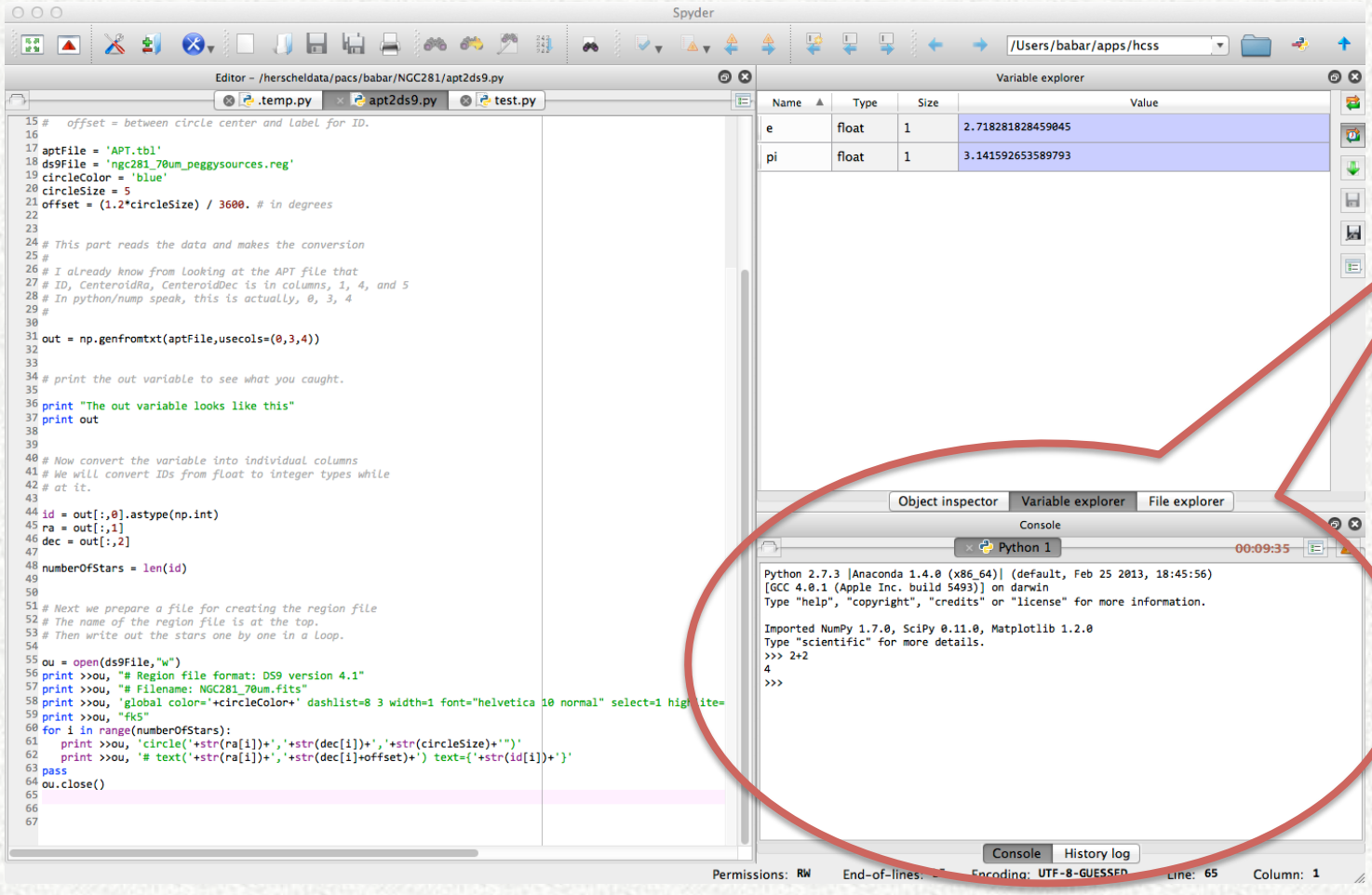
Recall

- Python is a scripting language
  - The compiler is always 'ON'
- Python uses an interpreter to parse python commands
- The interpreter is what 'runs' (executes) a python script
- The interpreter can be invoked by many different routes.

```
>>>
~]
~] python
Python 2.7.3 |Anaconda 1.4.0 (x86_64)| (default, Feb 25 2013, 18:45:56)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The python interpreter as started using a command-line command on MacOS terminal.

Anaconda python distribution



python interpreter in spyder. Look for the three arrows >>>

DEMO

spyder Development Environment

Spyder Demo

- Use **Spyder** for:

1. Writing code
2. Executing code
3. Checking things in the interpreter
4. Finding help

Spyder Demo

- Use **Spyder** for:

1. Writing code
2. Executing code
3. Checking things in the interpreter
4. Finding help

There are many, many things a development environment does. We will keep it simple, but feel free to try things on your own.

Spyder Demo

- Where do I write code?
- What are the buttons for?
- Executing single lines and blocks.
- Indentation.
- Spyder helps, parentheses, quotes, ...
- Using the interpreter
- Finding help

python on SHIPs

- We will use the interpreter to demonstrate python language.
  - Use spyder to follow the exercises and demos.
  - Invoke your version now and find the interpreter window.
- To issue an example command, simply type it in the interpreter window.
- We will learn how to run multiple commands (scripting) later.

Using the interpreter

Example: Python as a calculator. Enter the examples given below.

* stands for multiplication
+ stands for addition
/ stands for dividion
- stands for subtraction
** stands for power

```
>>> 2048+2048
>>> 2*3
>>> 19.99 * 1.095
>>> (203.4 + 203.8 + 202.4 + 202.8) / 4
>>> 10**2
>>> (103. – 4*3 ) /2
```

Comments

- In python the hash/pound character '#' denotes a comment.
- Comments are meant for humans.
- The interpreter will ignore everything to the right of the # sign.

```
>>> 2048+2048
>>> # 2048+2048
>>> # This is a comment.  I can put anything I like here.
>>> 10**2  # This demos 10 squared.
>>> # Author: Babar Ali
>>>
```

Comments will be used frequently to explain the commands.

More than a calculator

- The real power of python is in the commands and functions – which we will learn later.
- During the examples, you may encounter the following syntax:

```
>>> import numpy as np
>>> np.abs(-10)
```

- Simply enter the commands as shown for now. We will discuss their use in later sections.

VARIABLES AND DATA

Data Concepts

For our purposes …

- Data is anything held in memory in a **variable** and/or stored on disk in a **file**.
- Variable types and formats are limited by the programming environment.
  - Though, any complicated type may be defined with functions.
- Files can have any format or structure supported by the operating system.

Variables in python

- Simply, names that point to data items.
- Rules for variable names in python:
  - Must begin with letters a-z, or A-Z, or an underscore '_'
  - Subsequent characters can be number or letters, or underscore
  - Must not use one of the reserved words.
  - Must also avoid conflicts between function names with variables

Variables Names in python

Examples:

```
a
b10sp
cdelt1
statingPosition
starting_position
index
isFITStype
npts
```

Names are case sensitive. The following are two different variables to python.

```
fileName
filename
```

Types of data

Most common types	Description
Integers	Simply any positive or negative integer (whole) number: 0, 1, -20, …
Floating-point	A real number with a decimal point that also uses an exponent: -0.2, 3.1415, 1.8e7, -2.00 Different types of floating points allow different amount of precision (significant digits) for the stored the data value. Inf is used to represent infinity NaN is used to represent 'Not a Number'.
Boolean	A single byte number that holds a 'Yes' (1) or 'No' (0) value.
String	A sequence of ASCII characters.

More data types exist, but are not covered here:

complex, hexadecimal, character, …

Scalars, Vectors and Arrays

Data of all types may be:

Scalar: A single value.

10, -1.3e23, 'Ursula'

Or,

Arrays: Multi-dimensional array of values.

A **vector** is a limiting case of an array with only 1 dimension.

[1,2,6,10]

3000 x 3000 index array of floating-point values. E.g. a camera image.

Programming Concepts

- A variable's name has nothing to do with its type. All of the following are valid and different contents represented by variable 'a':

  a = 0

  a = [-1.0, 2.3, 9.8, 1.e10]

  a = "Hello strings"

- Variable names are chosen by humans for humans.
- Use a name that makes sense.

  pi = 3.141592654

- Use a naming convention.

Variables in python

To define a variable, simply assign it to a data type.

```
>>> a = 10   # Defines a scalar integer value and sets a to it.
>>> b = 19.99 * 1.095   # Compute the operation and store the result in 'b'.
>>> s = "A string"   # Strings must be defined with either a single or double quote.
```

You can also define multiple items at once.

```
>>> a = b = c = 10.2
>>> a, b, c = 1.1, 2.2, 3.3
```

Vectors and arrays use [and] to mark the boundaries, and a comma to distinguish data

```
>>> a = [1,2,3,6,7,10]  # Defines an array of integer values.
>>> s2 = ["yes", "you can have", "a vector of strings"]
>>> img = [ [0.1,0.2], [1.1,2.3], [3.2,1.2], [4.3,3.2] ] # A 2x4 Array of floats.
```

Calculations with variables

Variables are used like algebraic expressions in computations

```
>>> a = 19.99
>>> b = a / 1.095
>>> length = 23.4 # meters
>>> width = 12.8 # meters
>>> area = length * width  # defines a new variable area
>>> z = a*a - (a/b)**2
```

- Its not a good idea to mix data types.
- Python will forgive mixing integer and floats – the results will always be a float.
- Strings cannot be used in math computations and will give an error.

Converting Types

Python provides a way to change between data types

```
>>> a = 19.99
>>> int(a) # prints integer of 19.99.
>>> float(int(a))  # Converts integer back to float with loss of precision!
>>> str(a) # Now it's a string.
>>> s = '10'
>>> int(s) # This is also allowed.
>>> int( "my string" )  # This is NOT allowed because it is nonsensical.
```

Note the value of 'a' itself did not change in the examples from 19.99
To store the changed value, set it equal to a variable name.

```
>>> a_float = 19.99
>>> a_int = int( a_float )
>>> a_string = str( a_float )
>>> a_float = a_float * 2. # Setting it equal to itself is allowed!
```

Built-in python types and functions

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	(floored) quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`long(x)`	*x* converted to long integer
`float(x)`	*x* converted to floating point
`complex(re,im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*. (Identity on real numbers)
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

From
http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

Programming help

How can I tell what type of data I have?

```
>>> a = 19.99
>>> a      # will print the value of a.
>>> a.__class__      # prints the type for the data contained in the variable.
```

Where to find more help.

http://docs.python.org/2/library/

http://www.python.org

VARIABLES IN NUMPY

Beyond Built-in functions

What is the result of?

```
>>> a = [10,20,30]
>>> a*2
>>> a.__class__
```

For most science and computation needs, we prefer a*2 operation to multiply each element of ‘a’ by the scalar 2.
And, the result of a.__class__ does not tell us what types of data are held by ‘a’.

In fact, vector and array support is seriously lacking in basic python.

Enter numpy

- A set of python functions and classes that extend the usability of python for numerical computations
  - ➢ Used extensively in the scientific community.
- numpy is bundled with anaconda distribution.
- numpy documentation reference:

  http://docs.scipy.org/doc/numpy/contents.html

numpy data types

Data type	Description
bool	Boolean (True or False) stored as a byte
int	Platform integer (normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float	Shorthand for `float64`.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex	Shorthand for `complex128`.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Using numpy

Invoke numpy

```
>>> import numpy as np   # The common method.
```

Try previous example again in numpy

```
>>> a = np.int16( [10,20,30] )
>>> a*2
>>> a.dtype
```

Other ways to create arrays in numpy

zeros

```
>>> a = np.zeros(10)    # Create a vector of floats.  Length of vector is 10.
>>> a
>>> a = np.zeros(10, dtype=np.int16)  # Same as above, except now it is integers.
>>> a = np.zeros([2,8])  # Now it is a 2x8 floating-point array.
>>> a = np.zeros([2,8], dtype=np.int16)  # Now it is a 2x8 integer array.
```

arange

```
>>> a = np.arange(2,10)    # Create a vector of integer values starting at 2 and end at 9
>>> a
>>> a = np.arange(2,10, dtype=np.float)  # Same as above, except now it is floats.
>>> a = np.arange(2,10,2)  # Step by 2 instead of 1
>>> a
```

Manipulating array values

Can access individually

```
>>> a = np.zeros(10)    # Create a vector of floats.  Length of vector is 10.
>>> a[3] = 428.34
>>> a
```

Indexing always starts at 0.
a[3] is actually the fourth element of the array.

Can access a range of values.

```
>>> a = np.zeros(10)    # Create a vector of floats.  Length of vector is 10.
>>> a[3:8] = 428.34
>>> a
```

References to range of indices ends one before the last one. Here, elements 4, 5, 6, 7, and 7 are set to 428.34, but not the 9th.

More (but not all) on referencing

- Caution: python stops 1 index value short of the specified range.
  - Up to, but not including.

```
>>> a = np.arange(1,10)  # Create a vector of integer values starting at 1 and end at 9
>>> a[0]        # access the first element.
>>> a[2:4]      # access the 3rd-4th element (index 2, 3).
>>> a[2:]       # access all elements starting with the 3rd.
>>> a[:4]       # access all elements up to but not including the 4th element.
>>> a[-2]       # access the 2nd to last element.
>>> a[-5:]      # access all elements starting from the 5th from last.
```

Other commonly used data functions

```
>>> len(a)      # Print the length, number of elements, of 'a'.  Gives an error for scalars.
>>> a.size      # NUMPY only.  Similar to len(a) for numpy arrays.
>>> a.ndim      # NUMPY only.  Returns the number of dimensions for 'a'.
```

numpy

MORE MATH OPERATORS

Common math and trig functions

A rather complete listing:
http://docs.scipy.org/doc/numpy/reference/routines.math.html

```
>>> np.abs(a)        # Absolute value of 'a'.
>>> np.sqrt(a)       # Square root of 'a'.
>>> np.sum(a)        # Sum of all elements of 'a'.
>>> np.max(a)        # Maximum value of 'a'.
>>> np.min(a)        # Minimum value of 'a'.
>>> np.sin(a)        # Trig. Sin of 'a'.  The assumed value of 'a' is in radians.
>>> np.deg2rad(a)    # Converts 'a' from radians to degrees.
>>> np.rad2deg(a)    # Converts 'a' from degrees to radians.
```

Calls may be nested:

```
>>> np.sin( np.deg2rad([0.,30.,45.,90.]) )
```