Python: A Brief Introduction

Basics

- Getting started:
  - Recommended: Setup Anaconda/Miniconda as your Python installation
  - Running a script: `python <filename>` (e.g., `python script.py`)
  - Interactive Python environment: `ipython`
  - Python notebooks provide an elegant way to present your code and output. Start a python notebook server with: `jupyter-lab`. In order to make use of the in-line plotting function in python notebooks, include `%matplotlib inline` at the beginning.
  - Install python packages from the Conda repository (if you are using Anaconda/Miniconda) or Python Package Index (PyPI) or GitHub using `pip`:

    ```
    conda install [--no-update-dependencies] <packagename>
    pip install [--user] [--no-deps] <packagename>
    pip install [--user] [--no-deps]
    git+https://github.com/<user>/<repo>.git
    ```

    The `-user` flag installs the package just for current user in the user's local home directory. The `-no-deps` flag forces pip to ignore package dependencies.

- Operators, Logical Operators, Variables, and Containers

  ```
  +, -, *, /, %, **, =, ==, ∼, &, |, in, and, or, [], {}
  ```

- Reserved Keywords

  ```
  and, assert, break, class, continue, def, del, elif, else, except, exec,
  finally, for, from, global, if, import, in, is, lambda, not, or, pass,
  print, raise, return, try, while
  ```

- Printing
  - Few quick examples of printing to highlight how to print multiple variables, join and print strings, and using string formatting to further customize printing of variables.

    ```
    print(<var1>)
    print(<var2>, <var3>)
    print("Hello" + "World!")
    print("%s %s %d %f" % ("hello", "world", 3, 3.14159))
    print("%10s %15s %8d %10.4f" % ("hello", "world", 5, 3.14159))
    print("{0:10s}{1:15s}{2:8d}{3:10.4f}".format("hello","world",5,3.14159))
    ```

- Comments and Whitespace/Indentation
  - Comments are typically used to describe the code. Typically the author uses comments to convey additional information about the code to the reader in a more human-friendly manner (e.g., what the code is doing, what arguments are needed, etc.). All commented lines in the code are ignored by the interpreter. It is a good practice to properly comment your code for readability. There are two ways to annotate comments in Python: *i*) Single line comments begin with the hash character (`#`) and are terminated by the end of the line (all text after # is ignored); *ii*) Multi-line comments can be inserted as a string with `"""` delimiter at each end (everything between the `"""` delimiters is ignored).

```
# This is a comment

"""
This is a
comment that can
span multiple lines
"""
```

- Whitespace is used to denote blocks. In some languages, curly brackets (`{ }`) are used to identify block, whereas in Python, whitespace is used instead. When indented, a block of code becomes a child of the previous line. In addition to the indentation, the parent also has a colon following it. Examples of parent start blocks: `def, if, elif, else, try, except, finally, with, for, while, class`. To end a block, you simply un-indent.

```
im_a_parent:
    im_a_child:
        im_a_grandchild
    im_another_child:
        im_another_grand_child
```

- Functions (and lambda functions)
  - Proper declaration and definition of a function includes the function name, arguments and keyword arguments (arguments with a specific name that is required when calling the function), followed by a block of code, and (optionally) a `return` statement. If no `return` statement is specified, the function with return a value of `None`.

```
# Syntax
def <func_name>(arg1,arg2,...,kwarg1=value1,kwarg2=value2,...):
        """
        Comment describing the function and arguments
        and whatever else you may like
        """
        # Individual comments
        <statements>
        return <value>

# Example Function
def my_quadratic_root(a,b,c):
       """
       Returns the roots of a quadratic eqn.
       Works only for real roots
       Complex roots not yet implemented
       """
       sqrt_term = (b**2 - 4*a*c)**0.5
       root1 = (-b - sqrt_term) / (2*a)
       root2 = (-b + sqrt_term) / (2*a)
       return [root1, root2]
```

  - Quick definition of a function – `lambda`. This is useful when defining functions that contain only one straightforward computation (one line of code) – e.g., computing the square, calling another function with a fixed argument, etc. Apart from being limited to just one computation, another main drawback of the `lambda` function is that it cannot be used in a code that utilizes multiprocessing. This is because the `lambda` functions cannot be pickled.

```
# Syntax
<func_name> = lambda arg1,arg2,...: <expression>

# Same example function as before, but now in lambda form
my_quadratic_root = lambda a,b,c: [(-b - (b**2 - 4*a*c)**0.5) / (2*a),
                                   (-b + (b**2 - 4*a*c)**0.5) / (2*a)]
```

- Control Flow

  -
    ```
    if <expression>:
        ...
    elif <expression>:
        ...
    else:
        ...
    ```

  -
    ```
    for <target> in <iterable>:
            ...
    ```

  -
    ```
    while <expression>:
        ...
    ```

  -
    ```
    with <object> as <var>:
        ...
    ```

  -
    ```
    try:
        ...
    except <Error>:
        ...
    finally:
        ...
    ```

  - More control flow options: `break`, `continue`, `pass` – these can be used within `for`, `while`, `if-elif-else`.

- List Comprehension

  - General definition:

    ```
    [ <expression> for <target> in <iterable> ]
    [ <expression> for <target> in <iterable> <clauses> ]
    ```

    `<clauses>` can be a series of zero or more `if` statements.

  - Some Examples:

    ```
    [ i**2 for i in range(10) ]
    [ i for i in range(100) if i%2!=0 ]
    [ i for i in range(100) if i%2!=0 and i>25 ]
    ```

- File I/O

  - `open()` is the basic python function for file I/O. It satisfies the most basic I/O needs, however, it only offers low-level utility. In most cases, you will want to prefer higher-level I/O functions (such as NumPy’s savetxt-genfromtxt, AstroPy’s tables, etc. discussed later on).
  - `’w’` option opens a new file named ’text.txt’ for writing (will overwrite existing).

    ```
    outfile = open(’test.txt’,’w’)
    outfile.write("Hello World!")
    outfile.write("\nThis should be a new line")
    outfile.close()
    ```

  - `’a’` option will append to the ’test.txt’ file. Using a with statement eliminates the need to have a close() statement.

    ```
    with open(’test.txt’, ’a’) as f:
        f.write("\nThis is an appended line")
    ```

  - `’r’` option will open ’test.txt’ file in read-only mode.

    ```
    infile = open(’test.txt’, ’r’)
    for line in infile:
        print(line)
    infile.close()

    with open(’test.txt’, ’r’) as f:
        x = f.readline()     # Reads one line
        y = f.readlines()    # Reads all lines
    ```

- Modules and Packages (e.g., NumPy, SciPy, Matplotlib, Scikit-Learn, AstroML, AstroPy, PyFITS, CosmoloPy, etc.)

  ```
  import module
  module.function()

  import module as mod
  mod.function()

  from module import function
  function()
  ```

- Basic Functions and Built-in modules
  - Mathematical operations: `abs()`, `min()`, `max()`, `sum()`, `sorted()`, `sqrt()`
  - Casting functions: `set()`, `list()`, `len()`, `int()`, `str()`, `float()`
  - Iteration tools: `zip()`, `enumerate()`, `range()`, `xrange()`
  - Datatype verification tools: `type()`, `isinstance()`
  - `math` includes basic mathematical operations like `sin()`, `cos()`, `tan()`, `fabs()`, `exp()`, `log()`, `log10()`, `pow()`, `sqrt()` as well as constants π (pi) and e (e).
- Classes
  - Quick example of a class:

```
class Rectangle:
    def __init__(self,x,y):
        """
        Initializes the Rectangle class with dimensions x and y
        """
        self.x = x
        self.y = y
    def dimensions(self):
        """
        Returns the dimensions
        """
        return self.x, self.y
    def area(self):
        """
        Function to compute the area
        """
        self.area = self.x * self.y
        return self.area
    def scale(self,s):
        """
        Function to scale the dimensions by a factor s
        """
        self.x = self.x * s
        self.y = self.y * s
    def __repr__(self):
        """
        Overloads the print() function for this class to
        provide additional information
        """
        return "This variable is a Rectangle Class. \n" \
               "It has dimensions: "+str(self.x)+" x "+str(self.y)
```

- Further Learning: Classes, Decorators, Objects, Inheritance, Overloading, Variable Scope, Exceptions, Optimization, Multiprocessing, Modularization, etc.

NumPy

Full Documentation: *http://docs.scipy.org/doc/numpy/reference/*

- Import: `import numpy as np`
- Array Creation

```
np.arange(10)                              np.array([1,2,3,4,5])
np.arange(0, 1, 0.2)                       np.logspace(0,2,10)
np.linspace(0, 10, 5)
np.zeros(5)
```

- Array Operations

```
x = np.arange(10)                          np.sin(x)
x + 3.14                                   x * x.T
x**2
np.power(x, 2)
```

- Array Manipulation/Slicing

```
x[:3]                                      np.insert(x, 5, 100)
x[5:]                                      np.delete(x, 2)
x[:-3]                                     np.sort(x)
x[3:-2]                                    np.concatenate((x,y))
x[::-1]
np.append(x, 49)
```

- Structured Arrays

```
dtype = [(’ID’, int), (’name’, (str, 8)), (’value’, float)]
np.zeros(3, dtype=dtype)
```

- Masked Arrays:

```
a = np.random.rand(100)
masked_a = np.ma.masked_array(a, mask=(a<0.5), fill_value=-99.)
```

  Most NumPy array operations for masked arrays can be found in `np.ma`. E.g., `np.ma.sum()`, `np.ma.sqrt()`, `np.ma.power()`, etc.

- Saving and Reading Arrays from file

```
# Generate a dummy array
dtype = [(’ID’, int), (’name’, (str, 8)), (’value’, float)]
x = np.array( [(1,’Pi’,3.14159),(2,’e’,2.71828),(3,’sqrt(2)’,1.41429)],
               dtype=dtype)

# Save the array to ’test.txt’ file
np.savetxt(’test.txt’, x, fmt=’%8i %10s %10.4f’,
               header=’%6s %10s %10s’ % (’ID’,’name’,’value’))
```

```
# Read from the 'test.txt' file into a structured array
y = np.genfromtxt('test.txt', dtype=[('ID',int), ('name',(str,8)),
               ('value', float)], usecols=(0,1,2))

# Read from file into a individual arrays
# Only use when all columns have the same data type
x,y = np.genfromtxt('mydata.dat', usecols=(0,1), unpack=True)
```

- Random Number Generator

```
np.random.seed(2)
np.random.rand(N)
np.random.randn(N)
np.random.randint(x)
```

- Polynomial Fitting

```
coeff = np.polyfit(x,y,deg=1)
fit_func = np.poly1d(coeff)
```

- Statistics

```
np.mean(x)                                  np.percentile(x, q)
np.median(x)                                np.histogram(x, bins)
np.std(x)
```

SciPy

Full Documentation: *http://docs.scipy.org/doc/scipy/reference/*

- Integration

```
import scipy.integrate
scipy.integrate.quad(func, lolim, hilim)[0]
scipy.integrate.simps(y, x)
```

- Interpolation

```
import scipy.interpolate
scipy.interpolate.interp1d(x,y)
scipy.interpolate.UnivariateInterpolatedSpline(x,y)
```

- Optimization and Root Finding

```
import scipy.optimize
scipy.optimize.brentq(func, a, b)
scipy.optimize.curve_fit(func, x, y)
```

- Distribution Functions

```
from scipy.stats import distributions

distributions.norm.rvs(loc, scale, size)  # Generate random variables
distributions.norm.pdf(x, loc, scale)     # Compute the PDF
distributions.norm.cdf(x, loc, scale)     # Compute the CDF
distributions.norm.mean(loc, scale)       # Compute the mean of the PDF
distributions.norm.median(loc, scale)     # Compute the median of the PDF
distributions.norm.std(loc, scale)        # Compute the standard deviation of the PDF

# There are various distributions available
distributions.poisson.rvs(loc, size)
distributions.poisson.pdf(loc, size)
```

Matplotlib

Full Documentation: *http://matplotlib.org/api/pyplot_api.html*

- Basic Usage

```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.scatter(a,b)
plt.show()
plt.savefig(filename)
```

- Incorporating figures in a script

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x,y)
ax.scatter(a,b)
```

- Making Subplots: You can use the `add_subplot()` routine to define multiple subplots. The first digit = # of rows, second digit = # of columns, third digit = subplot# (going left-to-right and top-to-bottom)

```
fig = plt.figure()
ax  = fig.add_subplot(111)         # Makes a single subplot
ax1 = fig.add_subplot(211)         # Makes two subplots (one in each row)
ax2 = fig.add_subplot(212)
```

  A cleaner way is to use `subplots()` which lets you define the figure as well as all subplots at the same time.

```
fig, ax = plt.subplots(1,1)     # Can also be used for a single subplot
fig, ax = plt.subplots(2,2)     # Makes 4 subplots (two in each row)
```

  In the case of multiple subplots, the resulting `ax` variable is a N-dimension array containing all the subplots. One can also include fancier arguments

```
fig, ax = plt.subplots(2, 2, figsize=(10,8), dpi=75, tight_layout=True)
ax[0,0].plot(x,y)
ax[1,0].scatter(a,b)
```

- Making your plots presentable

```
plt.xlim(0,1)                              ax.set_ylim(-10,-1)
plt.ylim(10,100)                           ax.set_xlabel(’Axis label goes here’)
plt.xlabel(’Axis label goes here’)         ax.set_ylabel(’Axis label goes here’)
plt.ylabel(’Axis label goes here’)         ax.set_title(’Title goes here’)
plt.title(’Title goes here’)
ax.set_xlim(5,10)
```

- Plot types:
  - Line-plot: `plt.plot(x,y,c=’k’,lw=1.0,ls=’-’)`
  - Scatter-plot: `plt.scatter(x,y,marker=’+’,s=10)`
  - Errorbar-plot: `plt.errorbar(x,y,xerr,yerr)`
    E.g.,

```
x = np.random.randn(100)
y = np.random.randn(100)
dx = np.random.rand(100) * 0.5
dy = np.random.rand(100) * 0.5

upcond = (y > 0.9)
locond = (y < -0.9)
nocond = ~(upcond | locond)

plt.errorbar(x[nocond],y[nocond],xerr=dx[nocond],
    yerr=dy[nocond], ls=’’, c=’k’, capsize=0)
plt.errorbar(x[upcond],y[upcond],xerr=dx[upcond],
    yerr=0.5, ls=’’, c=’r’, uplims=True, capsize=0)
plt.errorbar(x[locond],y[locond],xerr=dx[locond],
    yerr=0.5, ls=’’, c=’r’, lolims=True, capsize=0)
```

  - Histogram: `plt.hist(x,bins,histtype=’step’)`
  - 2D-Histogram: `plt.hist2d(x,y,bins=[binsx,binsy],cmap=plt.cm.cmap)`
    E.g.,

```
hist2d, binsx, binsy = np.histogram2d(x,y,bins=[binsx,binsy])
xcenter = 0.5*(binsx[1:]+binsx[:-1])
ycenter = 0.5*(binsy[1:]+binsy[:-1])
plt.pcolormesh(xcenter,ycenter,hist2D,cmap=cmap)
```

  - Image Plotting: `plt.imshow(x,cmap,vmin,vmax)`

- Saving your figures (supported file formats include PDF, PNG, JPEG, PS, EPS, TIFF – all may not be available depending on your machine)

```
plt.savefig(filename)
fig.savefig(filename)
```

AstroPy

Full Documentation: *http://docs.astropy.org/*

- FITS file I/O (*http://docs.astropy.org/en/stable/io/fits/*).

```
import astropy.io.fits as fitsio

hdulist = fitsio.open(filename)
hdulist.info()
hdr = hdulist[1].header
data = hdulist[1].data

data = fitsio.getdata(filename, 1)
data, hdr = fitsio.getdata(filename, 1, header=True)

fitsio.writeto(filename, data)
```

- Table I/O (*http://docs.astropy.org/en/stable/table/*)

```
from astropy.table import Table

# For tables in the FITS format, read() will detect the correct formatting
table1 = Table.read('filename.fits')

# For ascii files, a format must be provided
table2 = Table.read('filename.csv', format='ascii.csv')
table3 = Table.read('filename.txt', format='ascii')

# You can also read a number of other formats
table4 = Table.read('filename.txt', format='ipac')

# List table columns
print(table1.colnames)

col = table1['column_name']
row = table1[0]

# Update a specific entry (column, row 0)
table1['column_name'][0] = 42

table1.write('newfilename.fits')
table2.write('newfilename.txt', format='ascii.commented_header')
```

- Cosmology (*http://docs.astropy.org/en/stable/cosmology/*)

```
from astropy.cosmology import Planck15

Planck15.age(z=2)
Planck15.luminosity_distance(z=2)
Planck15.differential_comoving_volume(z=2)

# Converting units
```

```
import astropy.units as u

age = Planck15.age(z=2)
age.cgs          # to CGS
age.si           # to SI
age.to(u.s)      # to seconds
age.to(u.hr)     # to hours
age.to(u.yr)     # to years
age.to(u.Myr)    # to Myr
```