Python Workshop

Arun Prasaad Gunasekaran

Indian Institute of Science

11 August 2015 - 15 August 2015

Outline

Preliminary Settings

Matplotlib Settings:

1. Go to Spyder>Tools>Preferences>Ipython Console>Graphics
2. Select Activate support
3. Set backend to Qt or Tkinter (Qt)
4. Got to Advanced Settings
5. Select Greedy Completer
6. Press apply and ok.
7. Go to the ipython console and type % matplotlib qt
8. Set the Working Directory in the File Explorer.
9. Copy the path of the Working Directory
10. type cd (in the ipython console) and paste the path and enter
11. Keep cursor near any command and press Ctrl+I for documentation.

A sample python program

```
1  # This is a single line comment
2  """
3      Program to find the factorial of a number.
4      This is also a docstring or a document string
5  """
6  n = input("Enter a positive integer") # To get user input data
7  f = 1                                 # To initialize the fact value
8  if n is 1 or 0:                       # To check if n is 0 or 1
9      exit                              # To exit the if statement
10 else:                                 # To proceed further
11     for i in range(2, n+1):           # To start a loop from 2 to n.
12         f = f*i                       # Also written as f *= i
13 print "The value of {}! is {}".format(n, f)
```

Breaking it down - Comments

Comment - Lines that are ignored when the program is run. Used for excluding codes and for including messages.

```
# This is a single line comment
# Comment begins with a hash # symbol
```

Docstrings - Document strings. Multi-line comments. But more useful for including help/direction messages that appear when help utilities are called.

```
"""
    This is a Docstring. It starts and ends with
    triple " or triple ' quotes. Mix and match does not work!
"""
```

Breaking it down - input

Input command - Used for getting input from user to a variable.

```
1 n = input('Enter the value for n') # Dynamic input
2 n = raw_input('Enter the value for n') # Raw input — string by default
3 n = float(raw_input('Enter the value for n')) # Type casted input
```

Dynamic input analyses the data and assumes data type automatically.
Raw input takes the data as strings. Ignores assuming data type.
Type casting is used to control data types.

Breaking it down - Simple Variables

These are the standard variables and primary data types.

```
i = 1 # Integer. Stores integers
r = 5.78 # Floats. Stores numbers with decimal parts.
c = 'h' # Characters. Stores single characters
s = 'Strings' # Strings. Stores a series of characters
l = True/False # Logical. Stores binary values
```

Strings and characters use both single and double quotes, but must end accordingly. Use slash to place quotes if needed.

```
"Hello", 'Hello', 'My name is JD', "I asked, 'what is for lunch?'"
'I exclaimed, "This tree is big!"'
"I used slash \" to type a double quote symbol"
```

Breaking it down - Variable nature

A variable in python can have any data type. It can be modified at any level to have a new data type. This is because variables are actually "objects" in python.

Code:

```
1  a = 1.5      # Float/Real
2  print a, id(a), type(a)
3  a = "hello" # String/Character
4  print a, id(a), type(a)
5  a = 5        # Integer
6  print a, id(a), type(a)
7  a = True     # Logical
8  print a, id(a), type(a)
9  a = [7, 8.9, 10] # List
10 print a, id(a), type(a)
11 a = (5.2, 4, 12) # Tuple
12 print a, id(a), type(a)
13 a = {'v1': 6, 'v2' : 10} # Dictionary
14 print a, id(a), type(a)
```

Output:

```
1  1.5 48132648 <type 'float'>
2  hello 140517035372048 <type 'str'>
3  5 17809656 <type 'int'>
4  True 140517173074128 <type 'bool'>
5  [7, 8.9, 10] 140515599545264
6  <type 'list'>
7  (5.2, 4, 12) 140515599725824
8  <type 'tuple'>
9  {'v1': 6, 'v2': 10}
10 140515599649880 <type 'dict'>
```

Breaking it down - operators

Operators:

```
a + b
a - b
a * b
a / b
a += b # a = a+b
a *= b # -=, /= exists
a ** b
()
a % b
or, and, not
is, is not
in, in, not in
```

Operations:

```
Addition
Subtraction
Multiplication
Division
Increment addition
Increment multiplication
Exponent
Parenthesis
Modulo Operation
Logical Operators
Identity Operators
Membership Operators
```

There are many more! Remember UPEMDAS REL LOG

List Variables

Mutable, multi-datatype arrays. Can be single levelled or multi-levelled. Enclosed by [].

```
x = [1, 3, 5, 7, 8]
y = [1.5, 5, 8.94, -5.78]
z = [1, 'f', True, [6.45, "six"], False ]
l = [ 1, 3.5, 'a', "hello", ['34', 3.14, ["three"], 4], 4.21 ]
```

They allow lists within lists. Indexing goes from 0.

List Variable Indices

Code:

```
print "x = ",x
print "y = ",y
print "z = ",z
print "l = ",l
print x[0]
print x[4]
print y[4]
print z[3][1]
print l[4][2][0]
```

Output:

```
x =  [1, 3, 5, 7, 8]
y =  [1.5, 5, 8.94, -5.78]
z =  [1, 'f', True, [6.45, 'six'], False]
l =  [1, 3.5, 'a', 'hello', ['34', 3.14, ['three'],
1
8
IndexError: list index out of range
six
three
```

Tuple Variables

Immutable, multi-datatype arrays. Can be single levelled or multi-levelled. Enclosed by ().

```
x = (1, 3, 5, 7, 8)
y = (1.5, 5, 8.94, -5.78)
z = (1, 'f', True, (6.45, "six"), False )
l = ( 1, 3.5, 'a', "hello", ('34', 3.14, ("three"), 4), 4.21 )
```

They allow tuples within tuples. Indexing goes from 0.

Tuple Variable Indices

Code:

```
print "x = ",x
print "y = ",y
print "z = ",z
print "l = ",l
print x[0]
print x[4]
print y[4]
print z[3][1]
print l[4][2][0]
```

Output:

```
x =  (1, 3, 5, 7, 8)
y =  (1.5, 5, 8.94, -5.78)
z =  (1, 'f', True, (6.45, 'six'), False)
l =  (1, 3.5, 'a', 'hello', ('34', 3.14, ('three'),
1
8
IndexError: tuple index out of range
six
three
```

Mutable - Immutable

Since variables are objects in python, sometimes, multiple variables point to the same memory location. Sometimes, certain memory modifications done to one variable replicates in others (Mutable). Sometimes they do not (Immutable).

Code:

```
1 x = [4.5, 6.7]
2 y = x
3 x.append(1)
4 print x, y
5
6 x = (4.5, 6.7)
7 y = x
8 x.append(1) # Produces an error
9 print x, y
```

Output:

```
1 [4.5, 6.7, 1] [4.5, 6.7, 1]
2
3 AttributeError: 'tuple' object has
4 no attribute 'append'
5 (4.5, 6.7) (4.5, 6.7)
```

Floats, strings, characters, integers, tuples are immutable. Only list is mutable.

Switching data and Mutation...

However, switching data and mutation do not cause any issue.

Code:

```
1 x = [4.5, 6.7]
2 y = x
3 y = [7.8, 9.6]
4 print x, y
5
6 x = (4.5, 6.7)
7 y = x
8 y = (7.8, 9.6)
9 print x, y
```

Output:

```
1 [4.5, 6.7] [7.8, 9.6]
2
3 (4.5, 6.7) (7.8, 9.6)
```

Still, there is a problem with list and tuples. They are heterogeneous data arrays. We need homogeneous data arrays for scientific calculations. That is where **numpy arrays** come in handy.

Dictionary

A composite collection of different primary and derived data types. **Imagine a wallet/bag having several items as an example**. Has **keys** and corresponding **values**. Keys are the names/identifiers while values are the data. (Lists, tuples, arrays, integers, floats etc.,). Used for consolidating data into files.

Code:

```
a = {'v1':6, 'v2':10, 'lst': \
    [5.8, "hello"]}
print a
print a['v1']
print a['lst']
print a['lst'][1]
```

Output:

```
{'v1': 6, 'v2': 10, 'lst':
    [5.8, 'hello']}
6
[5.8, 'hello']
hello
```

Type Casting

Used for data conversion. Also used to avoid mutation. Restriction of data entry is also possible.

Code:

```
1  import numpy as np
2  a = 5
3  b = [5, 7.9, 3.4]
4  print int(a), type(int(a))
5  print float(a), type(float(a))
6  print chr(a), type(chr(a))
7  print str(a), type(str(a))
8  print bool(a), type(bool(a))
9  print list(b), type(list(b))
10 print tuple(b), type(tuple(b))
11 print np.array(b), type(np.array(b))
```

Output:

```
1 5 <type 'int'>
2 5.0 <type 'float'>
3 CR <type 'str'> # Not actual value
4 5 <type 'str'>
5 True <type 'bool'>
6 [5, 7.9, 3.4] <type 'list'>
7 (5, 7.9, 3.4) <type 'tuple'>
8 [ 5.   7.9  3.4]
9 <type 'numpy.ndarray'>
```

Numpy Arrays

Comes from a module numpy. Homogeneous data type array. Automatically assumes data type. But can be manually set too. Needs numpy module. Inputs can be lists or tuples.

Code:

```
1  import numpy as np
2  l = [5, 7, 9]
3  t = (5, 7, 9.7)
4  a = np.array(l)
5  b = np.array(t)
6  print type(l)
7  print type(t)
8  print type(a)
9  print type(b)
10 print np.dtype(a[2])
11 print np.dtype(b[1])
```

Output:

```
1 <type 'list'>
2 <type 'tuple'>
3 <type 'numpy.ndarray'>
4 <type 'numpy.ndarray'>
5 int64   # Signed int with 64 bits
6 float64 # Signed float with 64 bits
```

As you see, the data types are allocated automatically. Use **dtype** inside **np.array** to set the data type manually.

```
1 a = np.array(l, dtype=np.float32)
```

For and While Loops

```
for i in range(0, 11, 2):
    print i
# Note the indent! Break in indent means end out statements within loop.
# range function delivers values from 0 to 10 in steps of 2
# i takes one value in each iteration
vp = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
for val in vp:
    print "Val = ", val
    for num in val:
        print num
        # continue # to go to next iteration
        # break # to break the loop
i = 0 # i has to be initialized for the while loop
while i in range(0, 11):
    print i
```

A volunteer to predict the output!

Nested loops (one within the other is possible)

If Clause with an example

```
1  f = 1
2  n = input("Enter a value for n:")
3  if isinstance(n, int): # Checks whether the value is integer or not
4      if n is 1 or 0:
5          print "Execution at if"
6          print "The value of {}! = {}".format(n, f)
7          exit
8      elif n<0:
9          print "Execution at elif"
10         print "n is negative!"
11     else:
12         print "Execution at else"
13         for i in range(2, n+1):
14             f *= i
15         print "The value of {}! = {}".format(n, f)
16 else:
17     print "Factorial does not exist"
```

Defining Functions

Data independent functions. But make use of this feature wisely.

Exercises

Exercises

Problem 1 - Easy!

Find the first 12 Fibonacci numbers.
In a Fibonacci series/sequence, if F_n is the n^{th} term of the series, (provided $n \geq 2$) then it can be given in terms of the previous 2 terms, F_{n-1} and F_{n-2} by the recurrence relation,

$$F_n = F_{n-1} + F_{n-2}$$

Find all the values till 12th term (F_{12}), save them and print them using loops. Take $F_0 = 0$ and $F_1 = 1$. **Use numpy arrays. Set the data type to be int8. Do you notice anything strange? Set the data type to be int16 and repeat. Do you notice anything strange?** If yes/no, can you guess what could be the reason?

Problem 2 - Hard and Long!

The table shows the plot of relative humidity and altitude. Assume surface temperature $T_S = 30^\circ\ C$ and $P_S = 10^5\ Pa$. Use SI units in calculation.

Table: Altitude Vs RH Measurement

Z (km)	0	0.1	0.3	0.5	1	2	3	4	5	6	7	8	9	10
RH (%)	60	70	80	75	60	50	80	90	60	40	20	5	2	1

- Calculate e(z) which is given by $e = \frac{RH \times\ e_s}{100}$. (Use % values of RH directly)
- Calculate T(z) which is given by $T = T_S - \frac{gZ}{C_p}$, where $g = 9.806\ m/s^2$ and $C_p = 1005\ J/kgK$
- Calculate ρ(z) which is given by $\rho = \frac{P}{RT}$
- Calculate P(z) which is given by $P = P_s \exp \frac{-gZ}{RT}$, where $R = 287\ J/kgK$
- Calculate q(z) which is given by $q = 0.622 \left[\frac{e}{P} \right]$
- Calculate $e_s(Z)$ which is given by $e_s = A \exp \left[-\frac{B}{T} \right]$. Where, $A = 2.53 \times 10^{11}\ Pa$ and $B = 5420\ K$

Problem 2 Continued ...

- Calculate W(z), given by $W = q \times \rho$
- Calculate h(z), given by $h = Lq + C_p T$. Where $L_q = 2.5006 \times 10^6\ J/kg$. (Use T in K here)
- Calculate MSE(z), given by $MSE = h + gZ$
- Calculate Velocity v(z), which is given by the relation, $v = v_0 \left(\exp \left[\frac{Z}{Z_s} \right] - 1 \right)$, where, $Z_s = \frac{RT_s}{g}$ (Where T_s is in Kelvin) and $v_0 = 100\ m/s$

Next Session ...

- Extracting Data from text/excel files
- Saving in matrix files (.mat)
- 1-D Plotting
- Advanced Mathematical Operations
- Interactive 2-D Plotting
- Some algorithms
- Plot Enhancements