

Python Workshop

Arun Prasaad Gunasekaran

Indian Institute of Science

11 August 2015 - 15 August 2015



Outline

- 1 Preliminary Settings
- 2 Sample Program
- 3 Features
- 4 Advanced Variables
 - Mutation
 - Type Casting
- 5 Loops and Decision Statements
- 6 Functions
- 7 Exercises



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)
14 ____

```



Breaking it down - Comments

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

```
1 ____# This is a single line comment
2 ____# Comment begins with a hash # symbol
3 ____
```

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

```
1 ____"""
2 ____    This is a Docstring. It starts and ends with
3 ____    triple " or triple ' quotes. Mix and match does not work!
4 ____"""
5 ____
```



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
4 ____
```

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.

```

1 _____i = 1 # Integer. Stores integers
2 _____r = 5.78 # Floats. Stores numbers with decimal parts.
3 _____c = 'h' # Characters. Stores single characters
4 _____s = 'Strings' # Strings. Stores a series of characters
5 _____l = True/False # Logical. Stores binary values
6 _____

```

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

```

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

```



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:

Output:

```

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)
15 _____

```



Breaking it down - operators

Operators:

```

1 _____a + b
2 _____a - b
3 _____a * b
4 _____a / b
5 _____a += b # a = a+b
6 _____a *= b # -=, /= exists
7 _____a ** b
8 _____()
9 _____a % b
10 _____or, and, not
11 _____is, is not
12 _____in, in, not in
13 _____

```

Operations:

```

1 _____Addition
2 _____Subtraction
3 _____Multiplication
4 _____Division
5 _____Increment addition
6 _____Increment multiplication
7 _____Exponent
8 _____Parenthesis
9 _____Modulo Operation
10 _____Logical Operators
11 _____Identity Operators
12 _____Membership Operators
13 _____

```

There are many more! Remember UPEMDAS REL LOG



List Variables

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

```

1 ____x = [1, 3, 5, 7, 8]
2 ____y = [1.5, 5, 8.94, -5.78]
3 ____z = [1, 'f', True, [6.45, "six"], False ]
4 ____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:

Output:

```

1 _____print "x" = '_____x = [1, 3, 5, 7, 8]
2 _____print "y" = '_____y = [1.5, 5, 8.94, -5.78]
3 _____print "z" = '_____z = [1, 'f', True, [6.45, 'six'], False]
4 _____print "l" = '_____l = [1, 3.5, 'a', 'hello', ['34', 3.14, ['t
5 _____print x[0]_____1
6 _____print x[4]_____8
7 _____print y[4]_____IndexError: list index out of range
8 _____print z[3]_____six
9 _____print l[4]_____three
10 _____10 _____

```



Tuple Variables

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

```

1 ____x = (1, 3, 5, 7, 8)
2 ____y = (1.5, 5, 8.94, -5.78)
3 ____z = (1, 'f', True, (6.45, "six"), False )
4 ____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:

Output:

```

1 _____print "x" = '_____x = (1, 3, 5, 7, 8)
2 _____print "y" = '_____y = (1.5, 5, 8.94, -5.78)
3 _____print "z" = '_____z = (1, 'f', True, (6.45, 'six'), False)
4 _____print "l" = '_____l = (1, 3.5, 'a', 'hello', ('34', 3.14, ('t
5 _____print x[0]_____1
6 _____print x[4]_____8
7 _____print y[4]_____IndexError: tuple index out of range
8 _____print z[3]_____six
9 _____print l[4]_____three
10 _____10 _____

```



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:

```
1 ____a = { 'v1':6, 'v2':10, 'lst': \
2 ____      [5.8, "hello"]}
3 ____print a
4 ____print a['v1']
5 ____print a['lst']
6 ____print a['lst'][1]
```

Output:

```
1 ____{'v1': 6, 'v2': 10, 'lst':
2 ____      [5.8, 'hello']}
3 ____6
4 ____[5.8, 'hello']
5 ____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 val
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 b
```

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

```

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

```

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 $T_S = 30^\circ \text{ C}$ and $P_S = 10^5 \text{ Pa}$. 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 $g = 9.806 \text{ m/s}^2$ and $C_p = 1005 \text{ J/kgK}$
- Calculate $\rho(z)$ which is given by $\rho = \frac{P}{RT}$
- Calculate $P(z)$ which is given by $P = P_s \exp \frac{-gZ}{RT}$, where $R = 287 \text{ J/kgK}$
- 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, $A = 2.53 \times 10^{11} \text{ Pa}$ and $B = 5420 \text{ K}$



Problem 2 Continued ...

- Calculate $W(z)$, given by $W = q \times \rho$
- Calculate $h(z)$, given by $h = L_q + C_p T$. Where $L_q = 2.5006 \times 10^6 \text{ J/kg}$.
(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 $v_0 = 100 \text{ m/s}$



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

