Unit 1 – Language Overview

(Elective course offered at PESIT Bangalore for 6th Semester BE Computer Science and Information Science students)

Anantharaman Narayana Iyer

28 Jan 2013

narayana.anantharaman@gmail.com

References

- Python documentation at: http://docs.python.org/2/tutorial/index.html
- Introduction to Python by Guido van Rossum, slide deck presented at LinuxWorld, Jan 2002

Course Structure

- **Unit 1**: Python language features overview
- **Unit 2**: Programming with Python Standard Library
- **Unit 3**: Web Programming in Python
- **Unit 4**: Programming for the cloud using Python
- **Unit 5**: Web Analytics in Python

Why Python?

- Python enables high programmer productivity
  - Edit/Compile/Execute/Test/… versus interpreter based development. Python doesn't need compile/link process.
  - Python programs are more compact
    - High level data types – example: Lists, Dictionaries etc
    - Indentation based statement grouping
    - Dynamic typing as opposed to strong static typed languages

Where we may not use Python?

- Class Discussion

Python Development

- Interactive development
  - Interactive mode with IDLE
  - Command line interpreter
- Executable Python Script
- Argument Passing
  - argv variable in the sys module can be used to process command line arguments
  - argv stores a list of strings containing command line arguments and can be accessed as:

    import sys

    argv[index]

Example

```
import sys
print 'hello world'
print 'number of arguments = ', len(sys.argv)
print sys.argv[0]
```

Problem

- You are required to do a project over this course that is broken up in to 5 mini projects, one per Unit. Each mini project would be implemented as a set of Python files. Design a source code organization for your project.

Modules and Import

- A Python file is also called module. Any python file is a module.
- A module may contain definitions for variables, class and function definitions
- One module can make use of the definitions done in other module by import statement
- The fully qualified name for a variable or function in a module is of the form: modulename.variablename

```
import sys
sys.exit(0)
from sys import exit
exit(0)
```

- The fully qualified name avoids namespace conflicts. eg, module1.foo() is different from module2.foo()
- A package in Python is a collection of modules organized under a directory structure to give a package hierarchy. It also contains __init__.py
- Packages can be nested at any deep, providing that the corresponding directories contain their own __init__.py file
- Python standard library is a set of packages and modules: for example, sys, re, os and so on

Python __main__

- Blocks of code in Python are marked through indentation
- Python code that is part of a .py file can be executed directly or can be imported to another Python module
- The outermost statements indented in the Python file or module, perform a one-time set up, that can be used to set up variables and functions as a part of import.
- A special variable "__name__" is set to "__main__" for those modules that are run directly.
- One may often use a boilerplate pattern: if __name__ ==__main to execute the code that we would like to run as a part of main program when the module is run directly, but not when the module is imported by some other module.
- This is often handy when testing a module with its own main function, which otherwise is not needed

Help from command line interpreter

- Inside the Python interpreter, the help() function pulls up documentation strings for various modules, functions, and methods. These doc strings are similar to Java's javadoc.
  - help(len) -- docs for the built in len function
  - help(sys) -- overview docs for the sys module (must do an "import sys" first)
  - dir(sys) -- dir() is like help() but just gives a quick list of the defined symbols
  - help(sys.exit) -- docs for the exit() function inside of sys
  - help('xyz'.split) -- it turns out that the module "str" contains the built-in string code, but if you did not know that, you can call help() just using an **example** of the sort of call you mean: here 'xyz'.foo meaning the foo() method that runs on strings
  - help(list) -- docs for the built in "list" module
  - help(list.append) -- docs for the append() function in the list module

Comments in Python

```
a = 10 # comment example
'''
This is a multi line comment with 3 single quote
'''
b = 20

"""
One more
Comment with 3 double quote
"""
c = a + b
print c
```

Variables

- Assignment

  a = 100

  b = 200

  x = (a + b) * (a – b)

- Multiple assignments in a single statement

  a = b = c = d = 100

- Variables need to be assigned a value before use

  a = 100

  c = a + z

Traceback (most recent call last):

  File "<pyshell#9>", line 1, in <module>

    c = a + z

NameError: name 'z' is not defined

Complex Numbers in Python

- Complex numbers are supported in Python. Imaginary numbers are written with a suffix of j or J. Complex numbers with a nonzero real component are written as (real+imagj), or can be created with the complex (real, imag) function.
- Complex numbers are always represented as two floating point numbers, the real and imaginary part. To extract these parts from a complex number *z*, use z.real and z.imag.
- The conversion functions to floating point and integer (float(), int() and long()) don't work for complex numbers — there is no one correct way to convert a complex number to a real number. Use abs(z) to get its magnitude (as a float) or z.real to get its real part.

Examples

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
------------------------------------------------
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Text Processing and Strings

- Text is a sequence of characters, while binary data is a sequence of bytes that can have any value between 0x0 to 0xFF
- Python strings, as opposed to C strings, are immutable.
- Often times, we create and process strings as a sequence of characters, many of these operations are applicable to sequence of bytes

Basic Text Processing

- There are 3 fundamental operations we perform while doing text processing
  - Parsing the data in to a structure internal to our application
    - Example: Extract a table from a HTML document
  - Transforming the input into something similar in some way but with changes of some kind
    - Example: Convert a text in to upper case letters
  - Generating completely new data
    - Example: Generate a HTML document given the text input, Generate executable scripts/source code, Generate data

Strings

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

String - example

Example 1

```
hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
Note that whitespace at the beginning of the line is\
significant."
print hello
```

Example 2

```
print """
Usage: thingy [OPTIONS]
        -h                      Display this usage message
        -H hostname             Hostname to connect to
"""
```

String examples (contd)

If we make the string literal a “raw” string, \n sequences are not converted to newlines, but the backslash at the end of the line, and the newline character in the source, are both included in the string as data. Thus, the example:

hello = r"This is a rather long string containing\n****

several lines of text much as you would do in C."

print hello

would print:

This is a rather long string containing\n\

several lines of text much as you would do in C.

Strings (contd)

- Strings can be concatenated (glued together) with the + operator, and repeated with *:

>>> word = 'Help' + 'A'

>>> word

'HelpA'

>>> '<' + word*5 + '>' '<HelpAHelpAHelpAHelpAHelpA>'

- Two string literals next to each other are automatically concatenated; the first line above could also have been written word = 'Help' 'A'; this only works with two literals, not with arbitrary string expressions:

>>> 'str' 'ing' *# <- This is ok*

'string'

>>> 'str'.strip() + 'ing' *# <- This is ok*

'string'

>>> 'str'.strip() 'ing' *# <- This is invalid*

File "<stdin>", line 1, in ?

'str'.strip() 'ing' ^

SyntaxError: invalid syntax

Strings (contd)

- Strings can be subscripted (indexed); like in C, the first character of a string has subscript (index) 0. There is no separate character type; a character is simply a string of size one.
- Substrings can be specified with the *slice notation*: two indices separated by a colon.

```
word = ‘HelpA’
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'
```

- Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

```
>>> word[:2] # The first two characters
'He'
>>> word[2:] # Everything except the first two characters
'lpA'
```

Strings (contd)

- Unlike a C string, Python strings cannot be changed. Assigning to an indexed position in the string results in an error:

>>> word[0] = 'x'

Traceback (most recent call last):

File "<stdin>", line 1, in ?

TypeError: object does not support item assignment

>>> word[:1] = 'Splat'

Traceback (most recent call last):

File "<stdin>", line 1, in ?

TypeError: object does not support slice assignment

- However, creating a new string with the combined content is easy and efficient:

>>> 'x' + word[1:]

'xelpA'

>>> 'Splat' + word[4]

'SplatA'

Strings (contd)

Degenerate Indices

```
>>> word[1:100]
'elpA'
>>> word[10:]
''
>>> word[2:1]
''
```

Negative Indices

```
>>> word[-1] # The last character
'A'
>>> word[-2] # The last-but-one character
'p'
>>> word[-2:] # The last two characters
'pA'
>>> word[:-2] # Everything except the last two characters
'Hel'
```

Strings (Contd)

- String Reversal with Slice
  - Example:
  - s = ‘abcd’
  - print s[::-1]
  - What is the output of: s[::2], s[::-2]?
- Syntax:
  - String_variable[start:stop:step]

Strings - % operator

```
# % operator
  text = "%d little pigs come out or I'll %s and %s
and %s" % (3, 'huff', 'puff', 'blow down')
# add parens to make the long-line work:
# also applicable for {}, []
  text = ("%d little pigs come out or I'll %s and %s
and %s" %
    (3, 'huff', 'puff', 'blow down'))
```

Problems

- Let a python string variable x assigned to ‘Hello world’. What is the output for:
  1. print x[-3]
  2. print x[3:]
  3. print x[3:7]
  4. print x[3:-5]
  5. print x[6:-5]

Lists

- Comma separated values within square brackets

  [***expr1, expr2, ..., exprN***]

- Elements need not be of same type
- Example:

a = [1, 2, ‘hello’, ‘world’, [‘a’, 123]]

Lists

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

- Like string indices, list indices start at 0, lists can be sliced, concatenated & so on:

```
>>> a[0]
'spam'
>>> a[3]
1234
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> 3*a[:3] + ['Boo!']
['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']
```

Lists

- All slice operations return a new list containing the requested elements. This means that the following slice returns a shallow copy of the list *a*:

```
>>> a[:]
['spam', 'eggs', 100, 1234]
```

- Unlike strings, which are *immutable*, it is possible to change individual elements of a list:

```
>>> a
['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
```

Lists

- Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:

```
>>> # Replace some items:
... a[0:2] = [1, 12]
>>> a
 [1, 12, 123, 1234]
>>> # Remove some:
... a[0:2] = []
>>> a
[123, 1234]
>>> # Insert some:
... a[1:1] = ['bletch', 'xyzzy']
>>> a
 [123, 'bletch', 'xyzzy', 1234]
>>> # Insert (a copy of) itself at the beginning
>>> a[:0] = a
>>> a
[123, 'bletch', 'xyzzy', 1234, 123, 'bletch', 'xyzzy', 1234]
>>> # Clear the list: replace all items with an empty list
>>> a[:] = []
>>> a
[]
```

List operations

```
>>> a = range(0, 10, 2)          # [0,2,4,6,8]
>>> a = range(5)                 # [0,1,2,3,4]
>>> a.append(5)                  # [0,1,2,3,4,5]
>>> a.pop()                      # [0,1,2,3,4]
>>> a.insert(0, 42)              # [42,0,1,2,3,4]
>>> a.pop(0)                     # [0,1,2,3,4]
>>> a.reverse()                  # [4,3,2,1,0]
>>> a.sort()                     # [0,1,2,3,4]
```

Fibonacci Example

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
        ... print b
        ... a, b = b, a+b
```

Tuples

(*expr1, expr2, ..., exprN*)

- **tuple**: contains *1* or more values of *any* type

```
t = (42, 19, 4.6, "hi");
```

- You can access a tuple's elements by index:

```
For example: t[1]
```

- Tuples can be sliced
- Tuples can be nested
- Tuples are immutable

Tuples

- Examples

```
a = 10
b = 20
m = a/b
n  = float(a)/b
k = float(a/b)
list = [100, ‘apples’, ‘Oranges]
t    = (a, b, m, n, k, list)
print t
t1 = (t, (t))
t1 = t * 3
t1 = t + t
```

Tuples

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
>>>
```

Tuples and Lists

```
mylist = [1, 2, 3]
dir(mylist)

t = tuple()
dir(t)
```

Since tuples are immutable, they may be implemented more efficiently

Exercise: Run an experiment to verify this

Tuple Assignment

- Often useful for multiple assignments

  x, y = (100, 200)

  (a, b) = (‘hello’, ‘world’)

Tuples (Contd)

- Tuples are comparable
  - Example (1, 2) > (0, 5)
- What is the output for:
  - (1, 2) == (1, 2)
  - 1, 2 == 1, 2
  - (1, 2) * 3 == (1, 2) * 3

Dictionaries

- Dictionaries are also referred to as Hash tables, "associative arrays“
- Creating a dictionary
  - d = {“a": “01", “b": “02“, “c”: “03”}
  - d = {(0, 0) : 0, (0, 1): 1}
  - Exercise:
    - Implement an array using a dictionary
    - Populate the dictionary automatically such that it represents a 3 x 3 matrix where a cell (i, j) contains the value (i + j)
    - Implement an encryption for English alphabet characters such that a character is represented by its equivalent Fib number. Consider only upper case letters and assign the base index 2 for character A. For example, the letter A has an index 2, B has an index 3, C has an index 4 and so on. Fib number sequence is: 0, 1, 1, 2, 3, 5, 8… The representations of A, B, C will be Fib(2), Fib(3), Fib(4), which are: 1, 2, 3
- Lookup:
  - d[“a"] -> “01"
  - d[“hello"] # raises KeyError exception
- Delete, insert, overwrite:
  - del d[“a"] # {“b": “02“, “c”: “03”}
  - d[“d"] = “04" # {“b": “02“, “c”: “03”, “d”: “04”}
  - d[“b"] = “b" # {“b": “02“, “c”: “03”}

Dictionary Operations

- Keys, values, items:
  - d = {‘a’: 1, ‘b’: 2}
  - d.keys() -> [‘a’, ‘b’]
  - d.values() -> [1, 2]
  - d.items() -> [(‘a’, 1), (‘b’, 2)]
  - Example:

  for k, v in d.items():

  print k, v

  Exercises:

  (a) A python dictionary object stores Name of a person (assume to be unique) and phone number. Print this dictionary as a HTML table.

  (b) Review the W3C Contacts API. Represent a subset of this schema as a Python dictionary. Print this as a HTML table.
- Presence check:
  - d.has_key(“a") -> True; d.has_key(“x") -> False
- Values of any type; keys almost any
  - {“firstname":“Anantharaman", “lastname”: “Narayana Iyer”, ("hello","world"):1, 42:"yes", “address": {‘street’: “200th Cross”, ‘city’: ‘Bangalore’}}

Dictionaries (Contd)

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is *hashing* (fast lookup technique)
  - **not** lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in **arbitrary order**
  - again, because of hashing

Variables

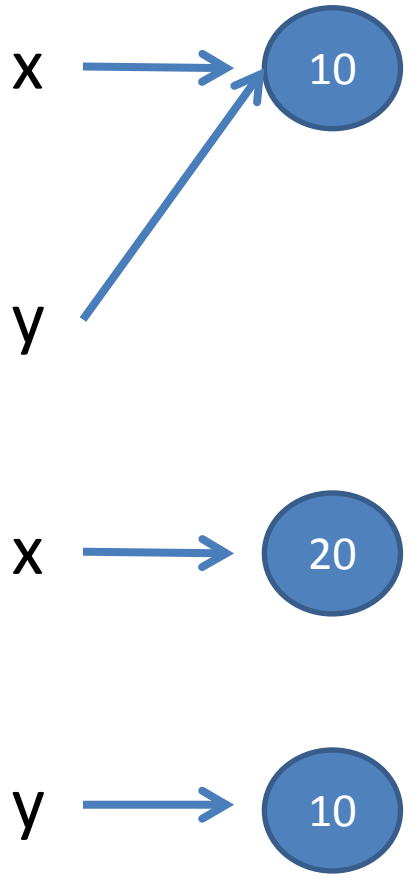
- Assignment manipulates references
  - x = y **does not make a copy** of y but makes x **reference** the object y references
- Example:

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

x = 10

y = x

x = 20



Loops and Conditionals

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

while *condition*:
    statements

for *var* in *sequence*:
    statements

break
continue

Functions and Procedures

def *name*(*arg1*, *arg2*, ...):

    """*documentation*""" # optional doc string

    statements

return # from procedure

return *expression* # from function

Example

```
def gcd(a, b):
    "greatest common divisor"
    while a != 0:
        a, b = b%a, a    # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'
>>> gcd(12, 20)
4
```

Example

```
def count1():
    global d
    d = dict()
    for w in words:
        if w not in d.keys():
            d[w] = 1
        else:
            d[w] += 1
def count2():
    global d
    d = dict()
    for w in words:
        d[w] = d.get(w, 0) + 1
if (__name__ == "__main__"):
    mess = raw_input("Enter a string for word count program\n")
    words = mess.split()
    print words
    count2()
    print d
```

Exercise:

Improve this program by:

1. Making this case insensitive
2. Removing punctuations
3. Minimizing the number of globals

Random Numbers

- You will often need to generate random numbers as a part of assignment – for example, to generate the test database

```
from random import *
```

`randint(`**`min, max`**`)`

- returns a random integer in range [**min**, **max**] inclusive

`choice(`**`sequence`**`)`

- returns a randomly chosen value from the given sequence
  - the sequence can be a range, a string, ...

```
>>> from random import *
>>> randint(1, 5)
2
>>> randint(1, 5)
5
>>> choice(range(4, 20, 2))
16
>>> choice("hello")
'e'
```

- Exercise: Simulate a 2 dies roll – return a tuple of 2 random numbers

File I/O

Reading Files

name = open("**filename**")

– opens the given file for reading, and returns a file object

name.read() - file's entire contents as a string

```
>>> f = open("hours.txt")
>>> f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```

Line-based File Processing

name.`readline()` - next line from file as a string

- Returns an empty string if there are no more lines in the file

name.`readlines()` - file's contents as a list of lines

```
>>> f = open("hours.txt")
>>> f.readline()
'123 Susan 12.5 8.1 7.6 3.2\n'

>>> f = open("hours.txt")
>>> f.readlines()
['123 Susan 12.5 8.1 7.6 3.2\n',
'456 Brad 4.0 11.6 6.5 2.7 12\n',
'789 Jenn 8.0 8.0 8.0 8.0 7.5\n']
```

Line-based Input Template

- A file object can be the target of a `for ... in` loop
- A template for reading files in Python:

```
for line in open("filename"):
    statements
```

```
>>> for line in open("hours.txt"):
...     print(line.strip())   # strip() removes \n

123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Exercise

- Write a function `stats` that accepts a file name as a parameter and that reports the longest line in the file.
  - example input file, `vendetta.txt`:

```
Remember, remember the 5th of November.
The gunpowder, treason, and plot.
I know of no reason why the gunpowder treason
should ever be forgot.
```

```
>>> stats(”vendetta.txt")
longest line = 46 characters
I know of no reason why the gunpowder treason
```

Exercise Solution

```
def stats(filename):
    longest = ""
    for line in open(filename):
        if len(line) > len(longest):
            longest = line

    print("Longest line =", len(longest))
    print(longest)
```

Writing Files

```
name = open("filename", "w")     # write
name = open("filename", "a")     # append
```

- opens file for write (deletes any previous contents) , or
- opens file for append (new data is placed after previous data)

name.write(str) - writes the given string to the file

name.close() - closes file once writing is done

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()

>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

Exercise

- Write a function `remove_lowercase` that accepts two file names and copies the first file's contents into the second file, with any lines that start with lowercase letters removed.
  - example input file, `carroll.txt`:

```
Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,
Beware the JubJub bird and shun
the frumious bandersnatch.
```

  - expected behavior:

```
>>> remove_lowercase("carroll.txt", "out.txt")
>>> print(open("out.txt").read())
Beware the Jabberwock, my son,
Beware the JubJub bird and shun
```

Exercise Solution

```
def remove_lowercase(infile, outfile):
    output = open(outfile, "w")
    for line in open(infile):
        if not line[0] in
  "abcdefghijklmnopqrstuvwxyz":
            output.write(line)
    output.close()
```

Exception Handling

- try/except/finally
- raise

OO Programming in Python

- Credits: Slides in this section are adopted directly from ***Introduction to Python by Guido van Rossum, slide deck presented at LinuxWorld, Jan 2002***

Classes

class *name*:

"*documentation*"

statements

-or-

class *name*(*base1*, *base2*, ...):

...

Most, *statements* are method definitions:

def *name*(self, *arg1*, *arg2*, ...):

...

May also be *class variable* assignments

Example Class

```
class Stack:
    "A well-known data structure…"
    def __init__(self):                     # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)        # the sky is the limit
    def pop(self):
        x = self.items[-1]                  # what happens if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0         # Boolean result
```

Using Classes

- To create an instance, simply call the class object:

```
x = Stack()  # no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.empty()   # -> 1
x.push(1)                          # [1]
x.empty()   # -> 0
x.push("hello")                                  # [1, "hello"]
x.pop()                  # -> "hello"            # [1]
```

- To inspect instance variables, use dot notation:

```
x.items                  # -> [1]
```

Subclassing

```
class FancyStack(Stack):
    "stack with added ability to inspect inferior stack items"

    def peek(self, n):
        "peek(0) returns top; peek(-1) returns item below that; etc."
        size = len(self.items)
        assert 0 <= n < size                                    # test precondition
        return self.items[size-1-n]
```

Subclassing (2)

```
class LimitedStack(FancyStack):
    "fancy stack with limit on stack size"

    def __init__(self, limit):
        self.limit = limit
        FancyStack.__init__(self)                  # base class constructor

    def push(self, x):
        assert len(self.items) < self.limit
        FancyStack.push(self, x)                   # "super" method call
```

Class / Instance Variables

```
class Connection:
    verbose = 0                                  # class variable
    def __init__(self, host):
        self.host = host                         # instance variable
    def debug(self, v):
        self.verbose = v                         # make instance variable!
    def connect(self):
        if self.verbose:                         # class or instance variable?
            print "connecting to", self.host
```

Instance Variable Rules

- On use via instance (self.x), search order:
  - (1) instance, (2) class, (3) base classes
  - this also works for method lookup
- On assignment via instance (self.x = ...):
  - always makes an instance variable
- Class variables "default" for instance variables
- But...!
  - mutable *class* variable: one copy *shared* by all
  - mutable *instance* variable: each instance its own

Problems

- Processing command line arguments
- We need to organize a set of modules as packages in Python and should be able to use them in other modules
- Importing the modules dynamically at runtime (more than 1 way)
- We need to develop a Python module that is usually used by other modules through import statements. We also would like to run this module directly, particularly when testing.
- Getting help in the command line interpreter for our custom modules/packages
- Processing Complex Numbers in Python
- Given some text through a command line interface, generate the equivalent HTML through string processing
  - Example: htmlgen –t “my page” –b “hello world”
- Reverse the elements of the list
  - Example: a = [1, 2, 3, 4], output = [4, 3, 2, 1]
- It is required to generate a dummy text that has a repeating pattern in order to fill up a text area in a HTML file. The size of the text is a variable. Write a program to generate the text with the required size

Problems

- Represent a 3 x 3 array using only:
  - list
  - Dictionary
  - Tuple
- Transpose the matrix created as above. As a variant, create 1 list variable and a dictionary to store the transposed matrix in the same list variable. What is the maximum size of the dictionary in terms of number of keys?

Queries for Assignments

- Which are the best budget hotels (per day < 5000) to stay in any beach resort in India?
- I want to spend my weekend at a hill station or a forest reserve that is closer to Bangalore (distance not more than 400 km on road). List the options.
- I am a manager in a MNC and I have a few guests visiting from San Francisco. Which places can I suggest for them to do site seeing on a 2 day weekend visit? The guests are open to any tourist spot but would like the place to be within 300 Km from Bangalore
- Which is the most expensive hill station to visit from Bangalore during the month of May? Assume that any distance < 400 km would be covered on road by car and > 400 km by flight to the nearest airport and then by car to tourist location.
- I have holidays on Feb 6th and 7th, suggest me places which i can visit during this time and also the weather should be pleasant

Device Database

1. List the smartphones in Nokia.
2. List the smartphone in Nokia with Windows OS.
3. List the Nokia phones with talk time greater than 10h.
4. List the touch phones with a secondary camera and GPS device
5. List the smart phones in Nokia with thickness less than 12mm

LinkedIn

1. List the email-ids of people who are interested in cloud computing
2. List the email-ids of people who have completed their graduation in PESIT, Computer Science
3. List the details(name, email-ids) of people in Bangalore whose skillset includes Python and Java and with utmost 2 years of industry experience
4. List all the python geeks.
5. List of all project leads in a company XX.

Wiki people 1

1. List of all the Nobel prize winners till date.
2. List of all the Nobel prize winners in 21st century.
3. List all the female Nobel prize winners.
4. List the Nobel prize winners and their work for which they received the award
5. List the Nobel laureates of India and their work.

Wiki people 2

1. List 10 freedom fighters of India
2. List the freedom fighters of southern India
3. List the freedom fighters of India and their achievements
4. List the female freedom fighters of India
5. Give the details of atleast 5 freedom fighters(birth, death, known for, other names)

Responsive Design

1. List the web pages(urls) which support responsive design

2. List the various screen sizes that are supported by a web page.

3. List the features that a particular site supports for a mobile device.

4. I need the list of mobile devices with their features that a responsive design web page can support.

5 Suppose a responsive design web page supports a device ‘X’. List the features of the mobile device X which have not been used in the web page design.

Financial Database

- Goal: Represent the P&L Statement as a Python data structure
- Queries:

1. List the names of companies that have a revenue of more than 100M USD with an operating margin of more than 20% for the given quarter/year
2. Find the top 3 companies that have best revenue growth rate over the past 4 quarters
3. List the names of companies with their EPS data for the given quarter/year
4. Determine the name of the company that has the highest R&D cost as a ratio of revenues for the given quarter/year
5. Given the stock ticker symbol of a company and the quarter/year, determine its total costs

Article Database

- List all the articles which contain the key word BIG-DATA.
- Group the articles into their respective category/ domain based on the title of the article.
  - **E.g.** "Malvertising campaigns at multiple ad networks lead to Black Hole Exploit Kit" belongs to network domain. Similarly can group under business, sports etc.
- Name the article which received highest number of likes.
- Name the Android related article that has the maximum blog comments
- List the Cloud computing articles from ZDNet