Introduction To Python

Week 2: Variables & More

Dr. Jim Lupo
Asst Dir Computational Enablement
LSU Center for Computation & Technology
jalupo@cct.lsu.edu

Resources

- "Building Skills in Python, Release 2.6.5", Apr 20, 2010 by Steven F. Lott

  `http://www.itmaybeahack.com/book/python-2.6/latex/BuildingSkillsinPython.pdf`

- Python Home Page

  `https://www.python.org/`

  See documentation tab in particular.

- Steve Brandt Tutorial:

  `http://stevenrbrandt.com/cios/index.php/python/index.xml`

Introducing Boolean Variables

- Have already seen these:
  - Int
  - Float
  - Complex
  - String
- Add *bool* to the list: true or false value
  - Formal keywords: **True** or **False**
  - In general: non-zero values are all treated as ***true***, 0 is ***false***.

Setting True or False

- x = True

  or

  x = 1
- y = False

  or

  y = 0

- Best to stick with the keyword values as there are hidden dangers.

Logical Operators

- Available: **and**, **or**, **not**
- For instance:

```
if light == 'green' or light == 'yellow' :
```

is the same as:

```
if not light == 'red' :
```

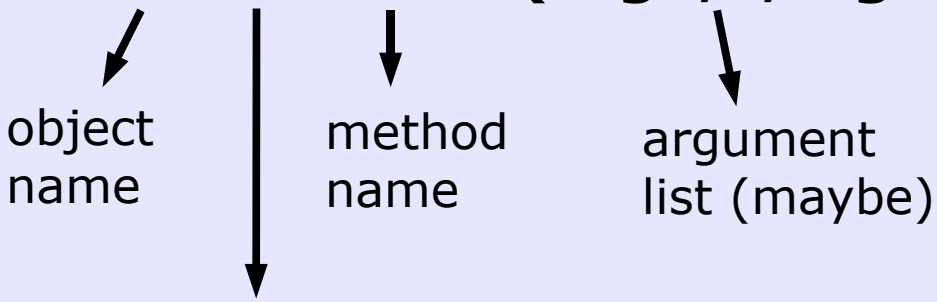
Python Variable Secret

- All python variables are class 1 objects.
- Means all have a ***value***, and associated ***methods*** for manipulating them.

Comparison Methods

- Methods accomplish a common action even if different data is involved.
  - Comparing string values versus float values.
- General syntax:

name.method(arg1,..,argN)



The 'dot operator' disguised as a period.

Documentation Is Your Friend

- Methods
  - Mastering involves:
    - Knowing they exist.
    - Knowing how to find documentation.
  - No way to go through them all.

Comparison Examples*

```
>>> x = 42.
>>> x.__lt__(40)
False
>>> x = 'beta'
>>> x.__gt__('alpha')
True
```

Note: __ is 2 '_'

The object **x** holds different types of values, but the methods associated know what to do: data & processing.

*https://docs.python.org/2/reference/datamodel.html#special-method-names

String Methods*

```
>>> x='gamma'
>>> x.capitalize()
'Gamma'
>>> print x
gamma
>>> x.upper()
'GAMMA'
>>> print x
gamma
```

Did not change contents!

Did not change contents!

*https://docs.python.org/2/library/stdtypes.html#typesseq

Cleaning Methods

```
>>> x = ' gamma '
>>> x.strip()
'gamma'
>>> x.rstrip()
' gamma'
>>> x.lstrip()
'gamma '
```

Note spaces!

If processing data files, often need only 'words', and ignore the *whitespace* (non-printing stuff).

Methods Just Work

```
>>> ' gamma '.strip()
'gamma'
>>> ' gamma '.rstrip()
' gamma'
>>> float('4').__lt__(5.)
True
>>> 4.__gt__(3)
SyntaxError: invalid syntax
>>>
```

???

What does this do?

```
w = raw_input('Y/N? ')
print w.__eq__('Y')
y = '>' + w + '<'
print w.strip().__eq__('Y')
z = '>' + w + '<'
print y, z
```

Returns A List?

- Several string methods say they return a list.
- Let's see an example and then talk about lists in more detail.

```
>>> x = 'Great American Novel'
>>> y = x.split()
>>> print y
['Great','American','Novel']
```

Multiple values in one variable name. Called an *array* in many other languages.

Examples

Start with: x = 'a b c d'

What does **x.split()** produce?

What does **print x** produce?

Structured Data: Lists

```
x = ['a', 15.3, 6+4j, 'b', 'd']
```

- List name is **x** and has 5 elements.
- This one contains 3 different types, which are?
- May be any legal values in any combination.

Access List Element

To reference the ***3rd*** element, do:

```
y = x[2]
```

- '2' is the ***index*** into x.
- Index number are 0-based - start from 0

Create and Add To Lists

```
>>> z = []
>>> x = -42
>>> z.append(42.0)
>>> z.append('a')
>>> z.append(x)
```

Create empty list named **z**.

Append 3 values.

- What does **print z** show?
- What does **print z[1]** show?

Example Continued

```
>>> z.append('1')
>>> print z
[42.0, 'a', -42, '1']
>>> z.sort()
>>> print z
[-42,42.0,'1','a']
```

Contents reordered!

Methods may be mutating or non-mutating.

Some Things To Try

- Add 2 lists together? What is in c?

```
>>> a = ['1', '2', '3']
>>> b = ['z', 'y', 'x']
>>> c = a + b
```

- How many elements in c? len(c)
- List of lists? Sure!

  w = [['1','2','3'],['x','y','z']]

- What is value given by w[1][1] ??

Undefined Operators?

```
>>> print a - b


Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print a - b
TypeError: unsupported operand type(s) for -:
'list' and 'list'
>>>
```

Adding two lists always makes sense, but not subtraction.

Power Of Slices

- Syntax: [start:end:increment]
  - start - first index, default of 0
  - end - last index, default last entry.
  - increment - stride through list, default 1
- [::3] - From first to last, every 3rd.
- [2:] - From 3rd element to last
- [:67] - From 1st to index 67.

The notation is formally called: list comprehension

More List Methods

list.append(x) - add item **x** to **list**.

list.extend(L) - add list **L** to end of **list**.

list.insert(i, x) - insert item **x** at index **i**

list.remove(i) - remove item at index **i**

list.pop([**i**]) - return index **i** from list, default last, then remove the item. Here [**i**] means value is optional, not another list.

list.index(**x**) - **list** index of first item equal to **x**

list.count(**x**) - count **list** items equal to **x**

list.sort(cmp=None, key=None, reverse=False)

list.reverse() - invert the **list** order

Structured Data: Dictionaries

- A dictionary defines a set of ***key:value*** pairs.
- **x = {'a':15.3, 'b':6+4j}**
  - Dictionary name is **x** with 2 elements:
    - First pair, key = 'a', value = 15.3
    - Second pair, key = 'b', value = 6 + 4j
- Use the key to find the value:
  - **z = x['a']** would assign 15.3 to **z**.
  - Can make reading code confusing at times.

Dictionaries: Continued

- z = {}
  - Create a new, empty dictionary.
- z['apple'] = 'red'; z['banana'] = 'yellow'
  - Create entries directly, no append!
- Use member functions to manipulate the dictionary:
  - x.update({'c':42}) - merge dictionaries. If key exists, replace value with new, if not, add to dictionary.

Example

Try:

```
>>> a = {'x':15,'y':16}
>>> b = {'z':17,'x':18}
>>> a.update(b)
```

What does **a** contain?

Working With Dictionaries

- Generate a list of keys?

  z.keys() => ['bananas', 'apples']

- Generate a list of values?

  z.values() => ['yellow','red']

- Why might dictionaries be more advantageous than lists?

Dictionary Examples

```
>>> x = {'a':15.3, 'b':6+4j}
>>> x.update({'c':42})
>>> print x
{'a': 15.3, 'c': 42, 'b': (6+4j)}
>>> x.keys
<built-in method keys of dict object at 0x02E0BA50>
>>> x.keys()
['a', 'c', 'b']
>>> x.values()
[15.3, 42, (6+4j)]
```

Not ordered, like lists!

Dictionary Danger!

- The order is determined by an internal algorithm, not the order of appearance or addition!
- If you need things in order, you have to get the keys or values list, sort it, then do the access you need.
  - Get keys
  - Sort keys
  - Use keys in sorted order to get values.

Other Sequence Types

- sets - list of unique entities

```
>>> x = [0, 1, 2, 3, 1, 5, 1]
>>> set(x)
set([0, 1, 2, 3, 5])
>>> y = set(x)
>>> print y
set([0, 1, 2, 3, 5])
>>> 4 in x
False
```

- tuples - static multiple valued entities.

```
>>> t = 'a', 5, True
>>> print t
('a', 5, True)
>>> s = 42, 'blue', 'Z', 'K'
```

tuples are sort of like N-dimensional points or coordinates, and can be used that way.

Mark Cube Corners

```
>>> corners = [ (0,0,0), (1,0,0), (0,1,0), (0,1,1),
                (1,0,0), (1,1,0), (1,0,1), (1,1,1)]
>>> print corners[2]
(0,1,0)
>>> loc = {}
>>> loc[corner[0]] = [0.0, 0.0, 0.0]
>>> loc[corner[1]] = [13.3, 0.0, 0.0]
```

Problems

- What data types are appropriate for:
  - Recording hourly temperature by day
  - Determine word frequency in a book.
  - Peoples names by birthdate.
  - Garment size and colors by stock number.

- Design the data structure to match anticipated processing.