Python Review Session

Fall 2014

With Janna Golden and Max Dougherty

Administrative

- Final Exam Review: Sunday 12/14 2-5pm
  - 2050 VLSB
  - Cover readings/lecture and programming concepts
- Paper Final: Tuesday 12/16 7-10pm
  - at RSF Fieldhouse

Goal for Python Review Session

- Give Review of Python **basics** in preparation for the Final Exam
- Focus will be on paper testable concepts and syntax

Python Version

This review session and the Final Exam will use Python 3.

Python Variables

- Python variables have to be declared before they can be used.

```
>>> count = count + 1
NameError: name 'count' is not defined
```

- Correction:

```
>>> count = 0
>>> count = count + 1
>>> count
1
```

- Alternate syntax.

```
>>> count += 1
```

Python Variables 2

- Python variables are case sensitive.

```
>>> word = “waffle”
>>> print(Word)
SyntaxError: invalid syntax
```

- Correction:

```
>>> word = “waffle”
>>> print(word)
‘waffle’
```

Basic Operators

Addition

```
>>> 5 + 6
11
```

Subtraction

```
>>> 5 - 2
3
```

Multiplication

```
>>> 3 * 4
12
```

Division

```
>>> 4/3
1.3333333333333333
```

Attention!: Addition can be used on many data types.

Notice: Division is exact.

+ is a Special Operator

- On strings

```
>>> “bat” + “man”
‘batman’
```

Question: How would you add a space?

```
>>> “bat ” + “man”
‘bat man’
```

Exercise 1: Mixed use? What happens?

```
>>> n = 5
>>> print("Count to " + n)
```

Output:

```
TypeError: Can't convert 'int' object to str implicitly
```

Fix:

```
>>> print("Count to " + str(n))
or
>>> print("Count to", n)
```

Aside: Python Print

- Like most python functions, it **requires** parentheses around the input
- Can print several types of python data in one statement

```
>>> print("The", 3, "numbers are",[3,1,4])
The 3 numbers are [3, 1, 4]
```

+ is a Special Operator: Lists

- Lists can be linked using +

```
>>> [“fee”, “fi”] + [“fo”, “fum”]
[“fee”, “fi”, “fo”, “fum”]
```

- Other examples:

```
>>> [ ] + [“a”]
[“a”]
>>> [5, [4, 3]] + [[[“a”], 2], “z”]
[5, [4, 3], [[“a”], 2], “z”]
```

Exercise 2: What will be returned?

```
>>> [[["f" + "o" * 2] + [["b"]]] + [[["a"]]] + ["r"]
```

Output:

```
[['foo', ['b']], [['a']]], 'r']
```

Note: “o” * n, is a tricky but interesting operator. It duplicates the string, n times.

Aside: Order of Operations

>>> [“f” + “o” * 2]

[‘foo’]

Question: Why not [‘fofo’]?

Answer: Python uses a standard PEMDAS ordering, evaluating from the inside out.

Exercise 3: Find the 2 bugs!

```
>>> [[“why”, [“wont”]] + [“this ” + “work]]]
```

Output:

```
File "<stdin>", line 1
    [["why", ["wont"]] + ["this " + "work]]]
SyntaxError: EOL while scanning string literal
```

Error 1: Missing quotation after `“work`

Error 2: Extra “`]`“ at end of line

+ Does NOT work for Dictionaries!

```
>>> {“age”:16, “weight”:145} + {“age”:15, “height”:5}
TypeError: unsupported operand type(s) for +: 'dict' and
'dict'
```

Question: How would you handle the overlap? Which do you choose?

Better to require a more explicit method.

Side note: Can be done with

```
>>> dict(list(a.items()) + list(b.items()))
```

Note: Overrides items in **a** with items in **b**

Some list Functions

`len(lst)`	`Length of the list.`
`lst.append(x)`	`“Add x as the last item of lst”`
`x in lst`	`True if x is an item of lst`
`x not in lst`	`True if x is not an item of lst`

Index into Lists or Strings

- Use [ ] (square bracket) notation
- The following can be used on both Lists and Strings

To retrieve the first item:

```
>>> lst = [“cow”, “and”, “chicken”]
>>> lst[0]
‘cow’
```

Index into Lists or Strings 2

To retrieve the first letter of a string:

```
>>> lst = [“cow”, “and”, “chicken”]
>>> word = lst[0]
>>> word[0]
‘c’
```

Alternatives:

```
>>> lst[0][0]
‘c’
>>> [“cow”, “and”, “chicken”][0][0]
‘c’
```

Index into Lists or Strings 3

Retrieve all but **first** of list:

```
>>> lst = [“cow”, “and”, “chicken”]
>>> lst[1:]
[‘and’, ‘chicken’]
```

Retrieve all but last of list:

```
>>> lst = [“cow”, “and”, “chicken”]
>>> lst[:-1]
[‘cow’, ‘and’]
```

Alternative: `lst[:len(lst)-1]`

Index into Lists or Strings 4

Retrieve a subset of a the list:

```
>>> lst = [“cow”, “and”, “chicken”]
>>> lst[1:-1]
[‘and’]
```

Note: Again! Notice how the index after “:” is not included.

Exercise 4: What will this return?

```
>>> lst=[['never'],[['gonna'],['give']]],['you'],'up']
>>> lst[:-1][0][1:][0][0][0][:2]
```

Answer: ‘go’

Iterating Over Lists and Strings

Basic iteration:

```
>>> num_letters = 0
>>> for x in [“apple”, “banana”, “pear”]:
        num_letters += len(x)
    return num_letters
```

** **For** loop uses **x** to take on the value of each item of a list in succession.

Exercise 5: Find the 6 bugs!

Count the number of “a”s in a list.

```
>>> def count_a(lst):
        for word in list:
            for Letter in word
                if (letter = “a”):
                    count += 1
            return count
```

Error 1: in first for loop should be lst not list
Error 2: missing colon after word
Error 3: Letter does not equal letter! For loop has a capital L and if statement has a lowercase l
Error 4: incorrect “if” statement, use “==” to check if equal, “=” is only used for variable assignment
Error 5: the `count` variable is not declared before it is used.
Error 6: return is in the wrong place (will return in first iteration of outer for loop)

Iterating Over Range

- Range acts like a list of indices
- example:

```
>>> for i in range(1,5):
        print(i)
1
2
3
4
```

*** Notice that range did not return the last value!!
This is consistent with methods of retrieving items from a list .

Range! (again)

- Range does not make a list!
- Instead it makes an **iterable**, useful as input to our **for** loop

```
>>> range(1:5)
range(1:5)
>>> list(range(1:5))
[1,2,3,4]
```

Range (again 2)

- Range with one input:

  >>>list(range(5))

  [0,1,2,3,4]

  Assumes the range should start at 0
- Range of 0:

  >>>list(range(0))

  []
- Range of negative number

  >>>list(range(-3))

  []
- Reverse range

  >>> list(range(5))[::-1]

  [4,3,2,1,0]

Exercise 6: What will these print?

We want to print a name in reverse order.

exercise 1:

```
>>> name = "gerald"
>>> for i in range(len(name))[::-1]:
        print(i)
```

```
5
4
3
2
1
0
```

exercise 2:

```
>>> name = "gerald"
>>> for i in range(len(name))[::-1]:
        print(name[i])
```

```
d
l
a
r
e
g
```