Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 1 - I. an introduction to the week

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 1 - II. an introduction to python

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

python

- Guido van Rossum

From wikipedia:

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

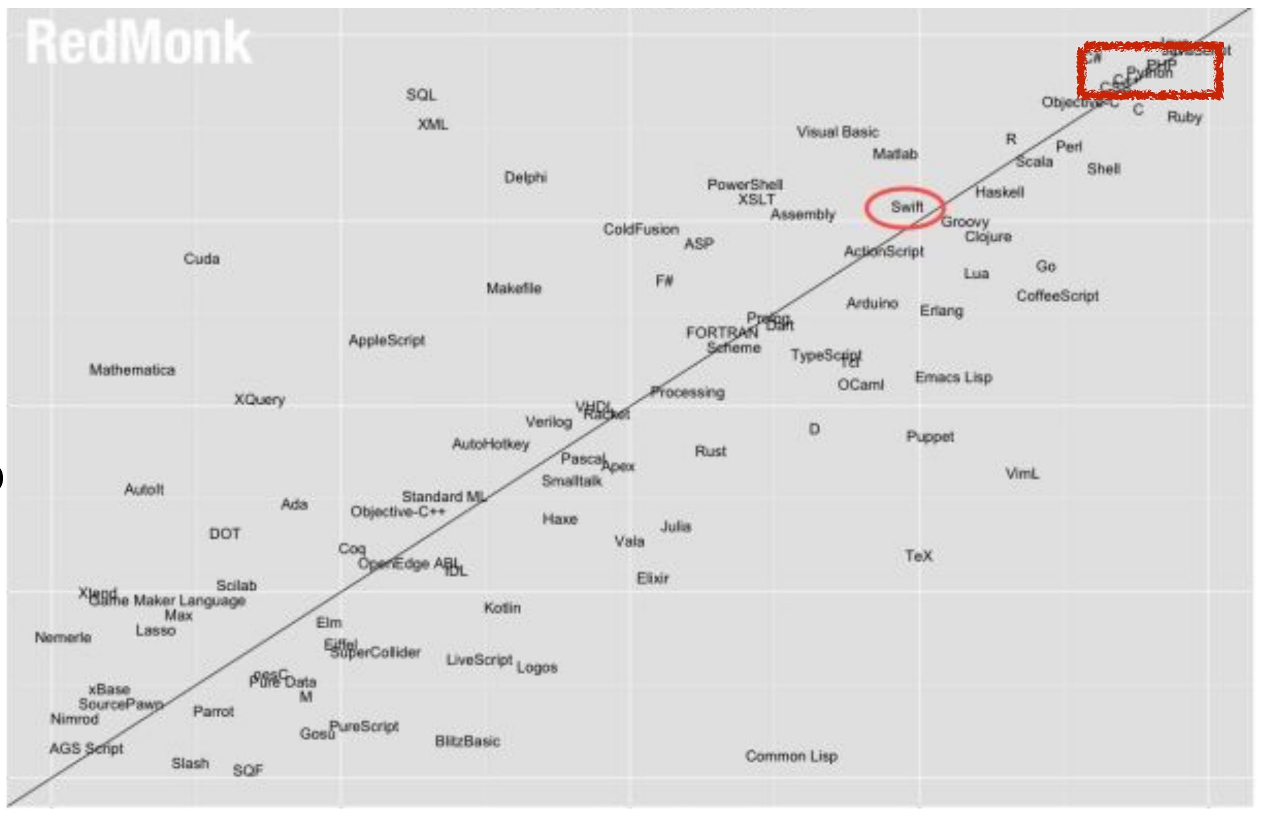
-Guido van Rossum in 1996

python

- appears in every programming top-ten list
    - hacker news
    - dice job list
    - book sales

you
should know it!

number of tags on stack overflow



number of github projects

python disclaimers

- batteries included
- weakly typed variables (dynamic)
- its an interpreter (kinda)
  - loops are slow
  - until they are not (compile it)
- can't use parallel instructions natively
  - unless you use IPython
- can be the glue for your different codebases

python releases

- 1.0 (up to 1.6)
    - basic python, complex numbers, lambdas
- 2.0 (still used, but it’s the beginning of the end)
    - unified types, made completely object-oriented
- 3.0 (still actively developed)
    - eliminate multiple paradigms (kinda)
    - 2.x not necessarily compatible with 3.x

installation

- on **mac** or **linux**:
  - open a terminal
  - do nothing
    - OS X and linux ship with python
  - …but you probably want to install python 3
- on **windows**, **mac**, or **linux**
  - go to https://www.python.org and get python 3 on your system
- but you will want access to the **packages**
  - something like anaconda or pip
  - allows you to install most any python package that is registered

hello world

- from interpreter
- from script
- from jupyter

exercise

- install python 3 on your machine
    - try using anaconda first!
- run “hello world” examples

Scripting for Data Science in Python and R

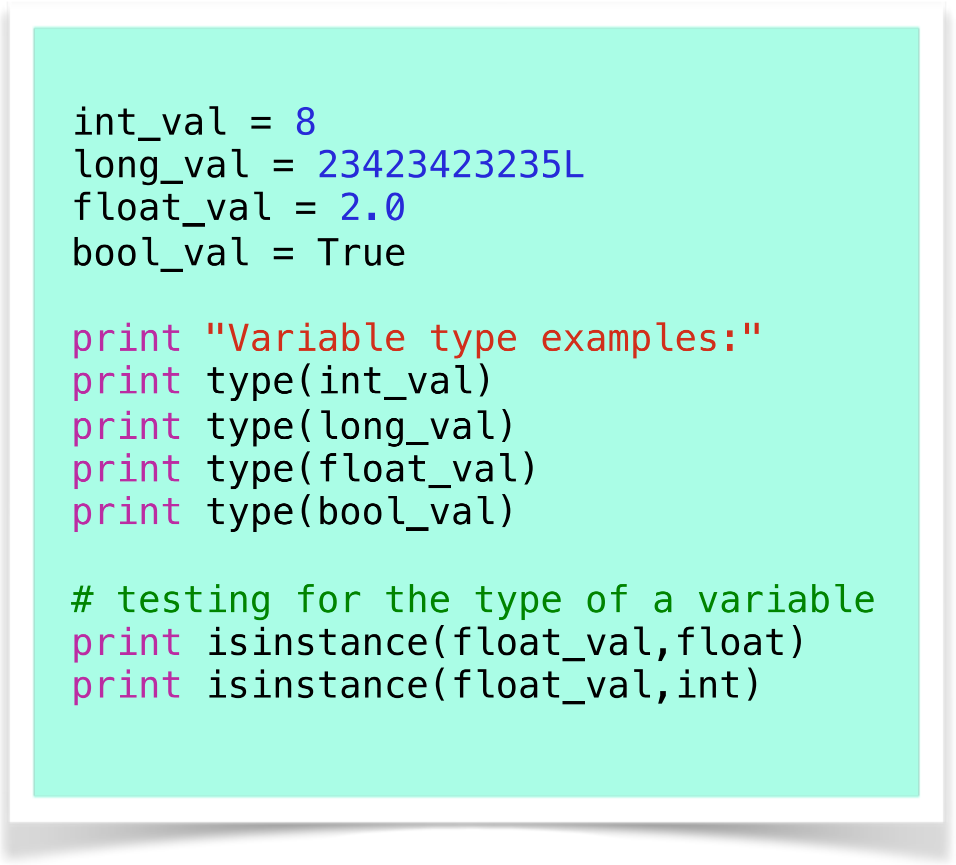
SMU Interdisciplinary Master's Degree in Data Science

Unit 1 - III. python basics

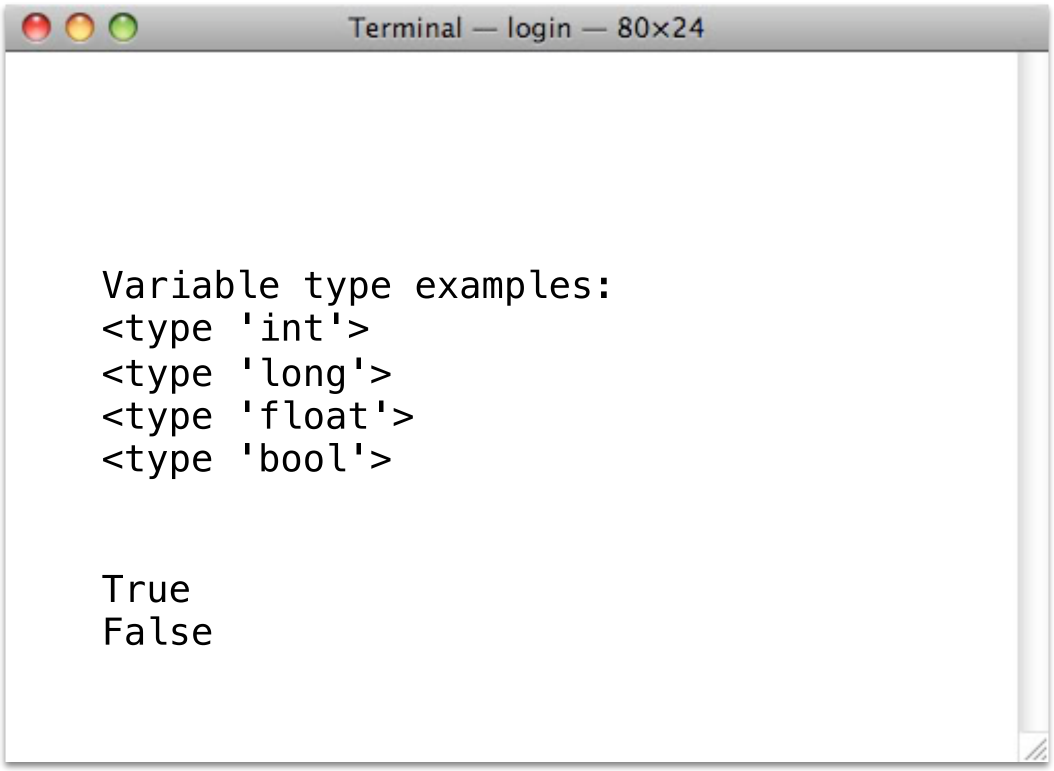
Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

comments, variables, types

```
# this is a comment
# place this at the top of python file to enable running as >>./filename.py
! /usr/bin/python
# otherwise you can run with >>python filename.py
# this command can be run from a terminal/cmd window
```



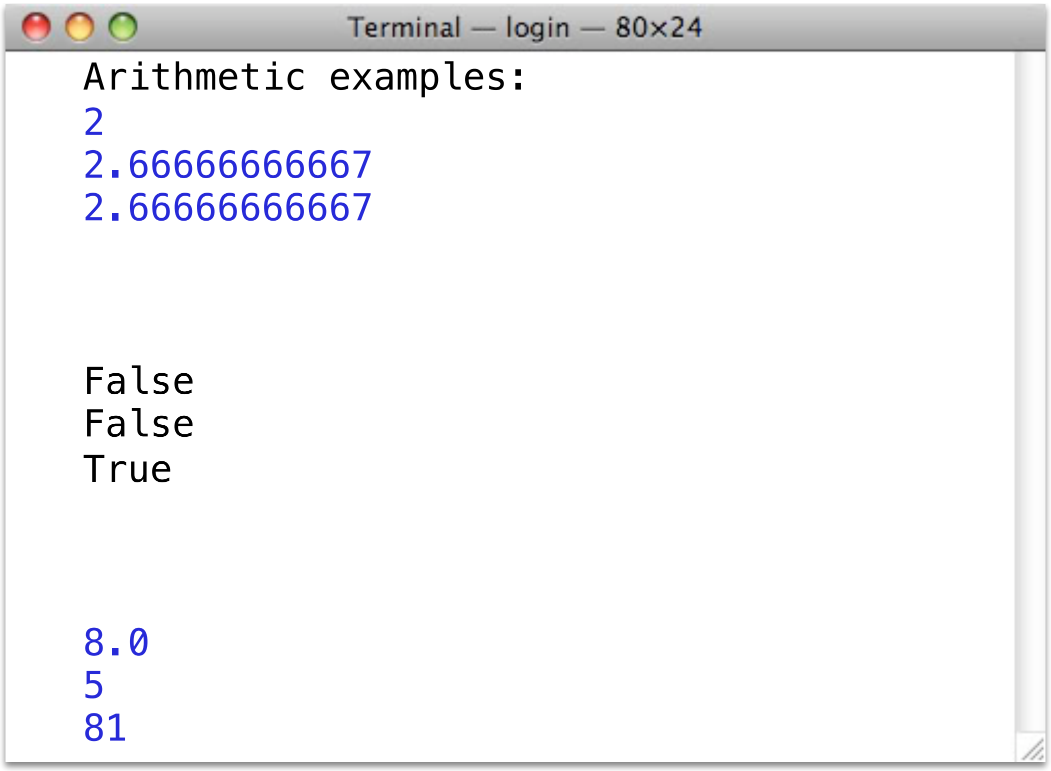
```
int_val = 8
long_val = 23423423235L
float_val = 2.0
bool_val = True

print "Variable type examples:"
print type(int_val)
print type(long_val)
print type(float_val)
print type(bool_val)

# testing for the type of a variable
print isinstance(float_val,float)
print isinstance(float_val,int)
```



```
Variable type examples:
<type 'int'>
<type 'long'>
<type 'float'>
<type 'bool'>


True
False
```

arithmetic and casting

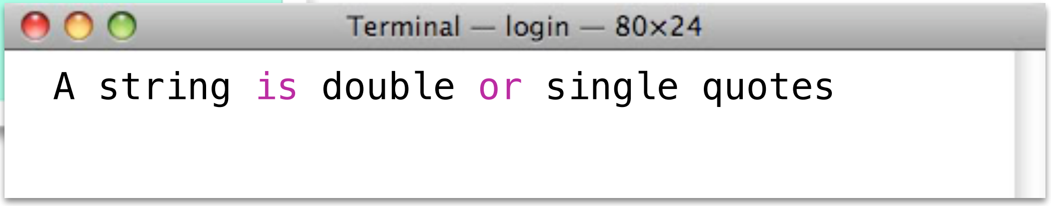
```
print "\nArithmetic examples:"
print 8 / 3
print float(8) / 3
print float(8) / float(3)



print True and False  # logicals
print 8 == 3   # logical equallty
print 5 <= 6   # logical comparison



print 2.0*4.0  # multiplication
print 65%6    # remainder, modulus
print 3**4    # 3 to the fourth power
```



```
Arithmetic examples:
2
2.66666666667
2.66666666667



False
False
True



8.0
5
81
```

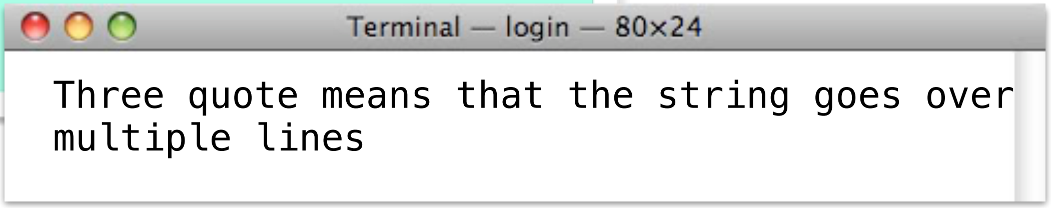
strings and string operations

```
str_val = "A string is double or single quotes”
print str_val
```



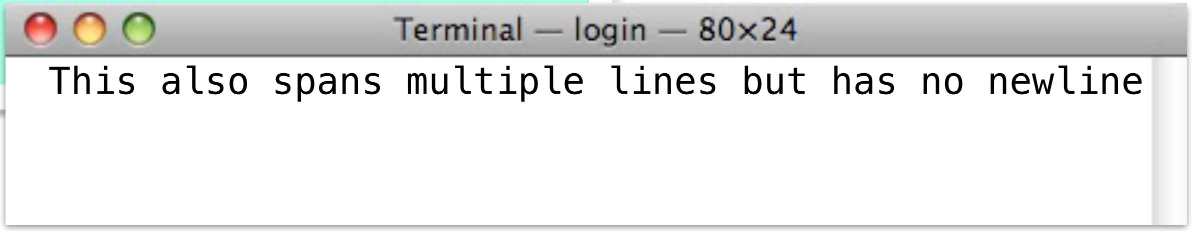
```
A string is double or single quotes
```

```
str_val_long = '''Three quote means that the string goes over
    multiple lines’''
print str_val_long
```



```
Three quote means that the string goes over
multiple lines
```

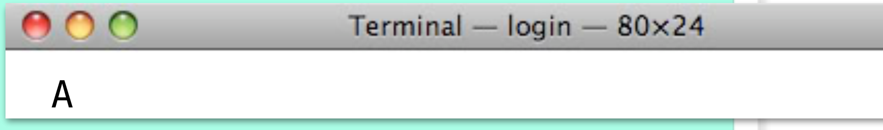
```
str_val_no_newline = '''This also spans multiple lines \
    but has no newline'''
print str_val_no_newline
```



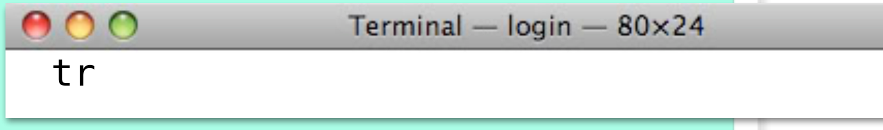
```
This also spans multiple lines but has no newline
```

strings and string operations

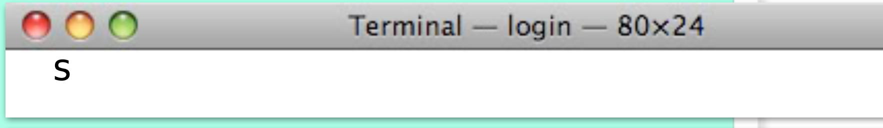
```
# string can be accessed in a variety of
different ways
print str_val[0] # initial element "0th" element


print str_val[3:5] # elements 3 and 4, but not 5


print str_val[-1] # the last element in the
string


print str_val[-5:] # the last five elements


print str_val[0:5] + str_val[5:] # print the
first five elements, then from the fifth and on


str_val[5] = 'G' # this is an error, strings are
immutable once they are set
```



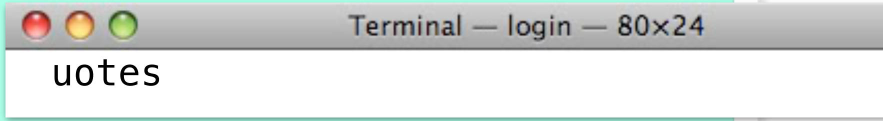
```
A
```



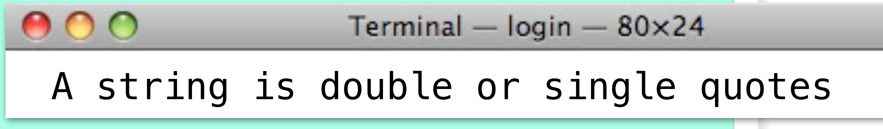
```
tr
```



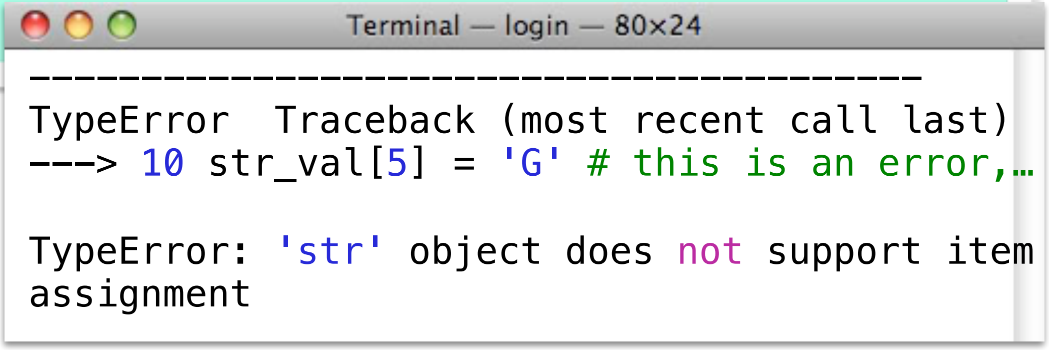
```
s
```



```
uotes
```



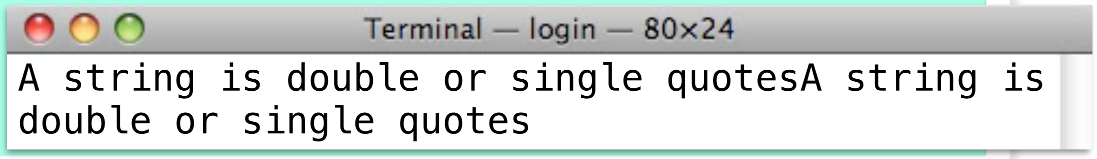
```
A string is double or single quotes
```



```
----------------------------------------
TypeError  Traceback (most recent call last)
---> 10 str_val[5] = 'G' # this is an error,…

TypeError: 'str' object does not support item
assignment
```

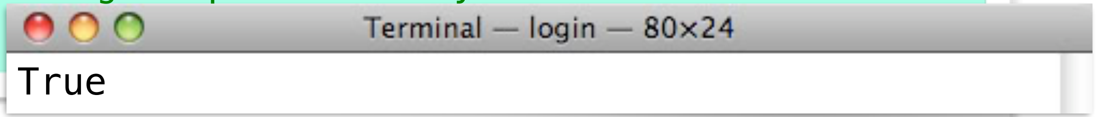
strings and string operations

```
# some common operations for strings
print str_val*2 # mutliply is like adding many times, here it repeats the string
```



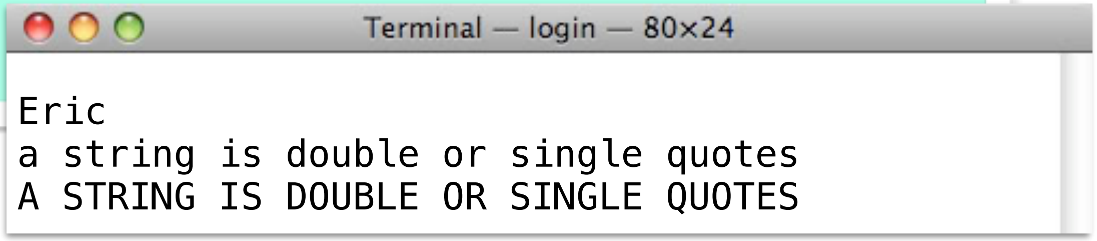
```
A string is double or single quotesA string is
double or single quotes
```

```
print 'Python' > 'Java' # compare the strings alphabetically
```



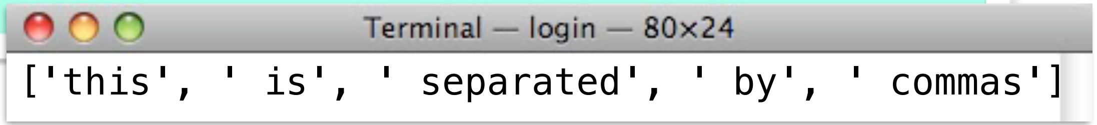
```
True
```

```
print "eric".capitalize() # the dot operator works like most other OOP languages
print str_val.lower()
print str_val.upper()
```



```

Eric
a string is double or single quotes
A STRING IS DOUBLE OR SINGLE QUOTES
```

```
print "this, is, separated, by, commas".split(',') # this results is returned as a
list, which we need to talk about!
```



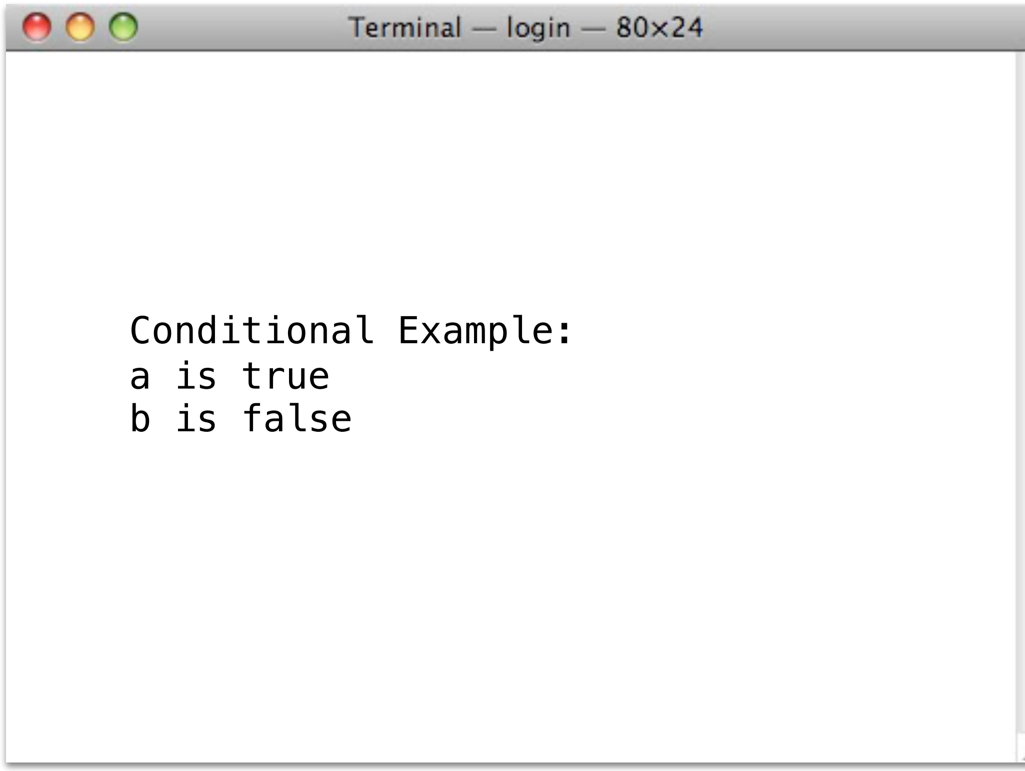
```
['this', ' is', ' separated', ' by', ' commas']
```

calculator example 1

- build the interpreter

conditionals

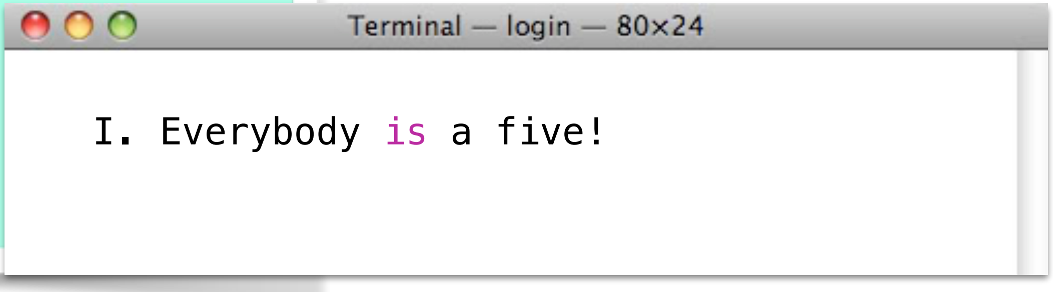
```python
# conditional example
print "\nConditional Example:"

a, b = True, False

if a:
    print "a is true"
elif a or b:
    print "b is true"
else:
    print "neither a or b are true"

# conditional assignment
val = "b is true" if b else "b is false"
print val
```



```

Conditional Example:
a is true
b is false
```

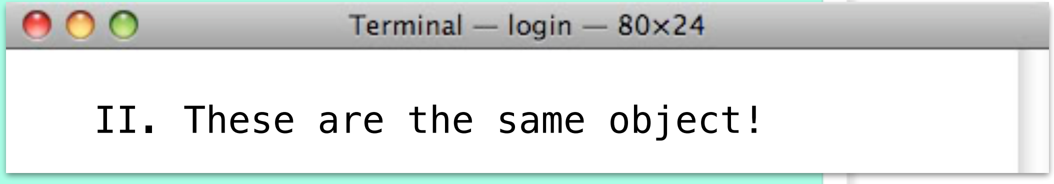
conditionals

```python
# I. the traditional == works as expected
a=5
b=5
if a==b:
    print "I. Everybody is a five!"
else:
    print "I. Wish we had fives…”
```



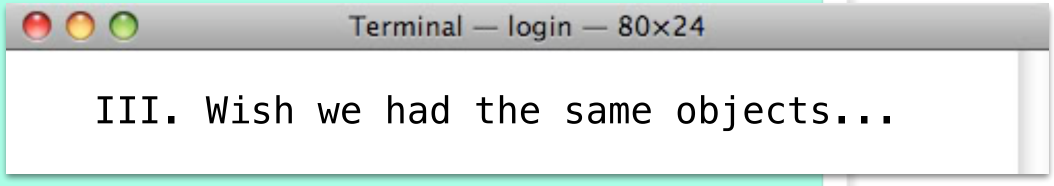
```
I. Everybody is a five!
```

```python
# II. the "is" function is for object comparison, much like comparing pointers
a=327676
b=a
if a is b:
    print "II. These are the same object!"
else:
    print "II. Wish we had the same objects..."
```



```
II. These are the same object!
```

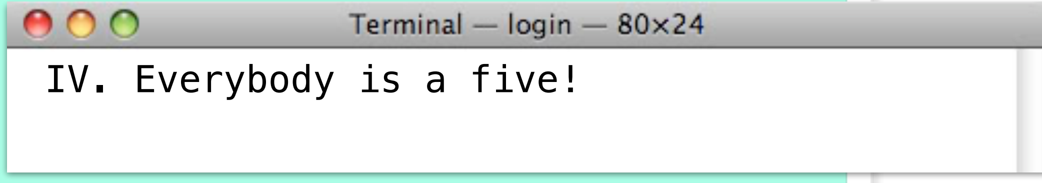
```python
# III. while these have the same value, they are not the same memory
a=327676
b=327675+1
if a is b:
    print "III. These are the same object!"
else:
    print "III. Wish we had the same objects..."
```



```
III. Wish we had the same objects...
```

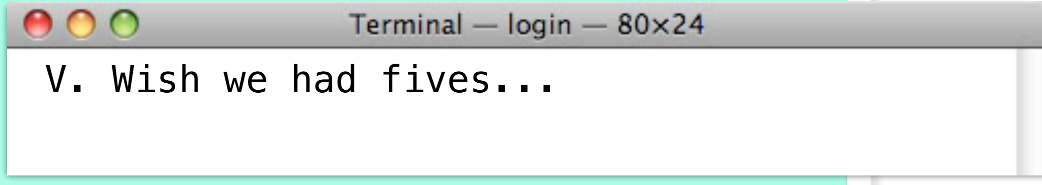
conditionals

```python
# IV. you would expect this to say wish we had fives,
# but small integers like this are cached so right now they do point to the same memory
a=5
b=4+1
if a is b:
    print "IV. Everybody is a five!"
else:
    print "IV. Wish we had fives..."
```



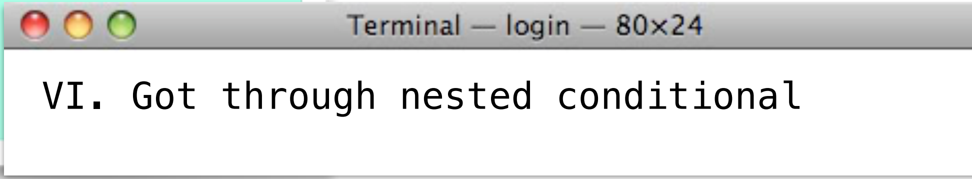
```
IV. Everybody is a five!
```

```python
# V. but if we change the memory, that caching gets released
b = b*2.0
b = b/2.0
if a is b:
    print "V. Everybody is a five!"
else:
    print "V. Wish we had fives..."
```



```
V. Wish we had fives...
```

```python
# you can also perform nested conditionals, like bounding
if 5 < 8 < 6: # not true because 8 is not less than 6
    print 'VI. How did we get here'
elif 4 < 18 < 22:
    print "VI. Got through nested conditional"
```

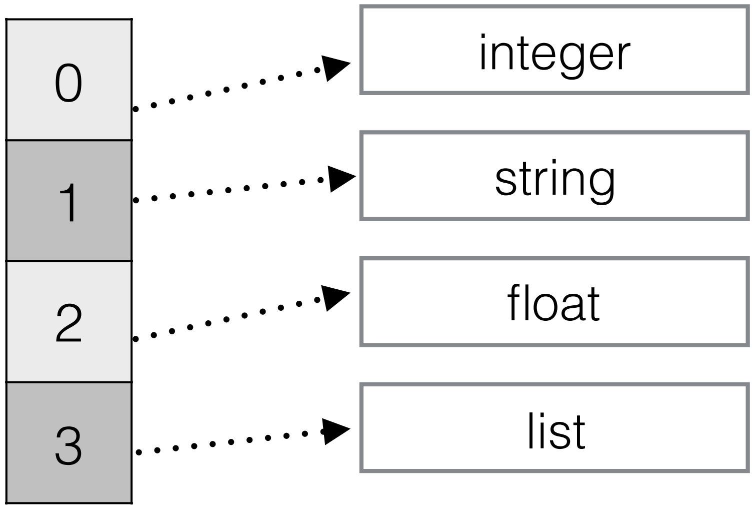


```
VI. Got through nested conditional
```

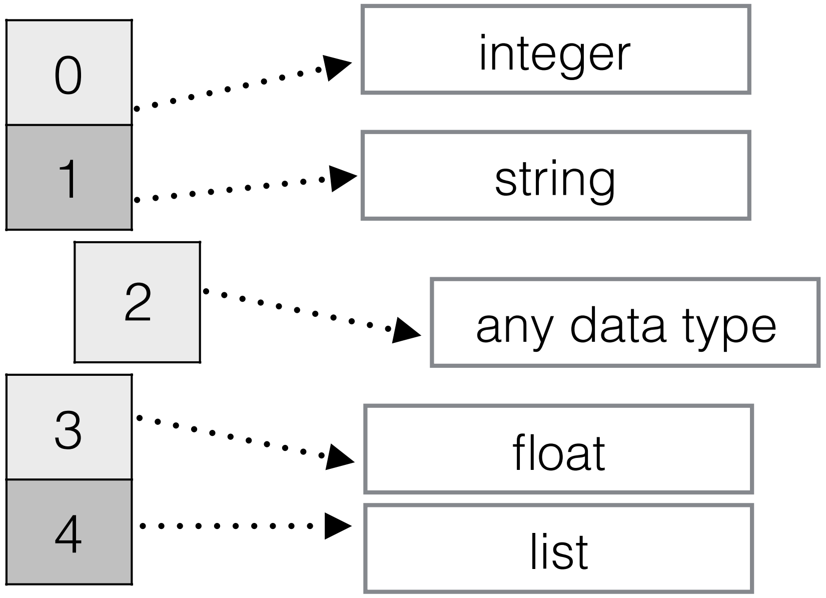
calculator example 2

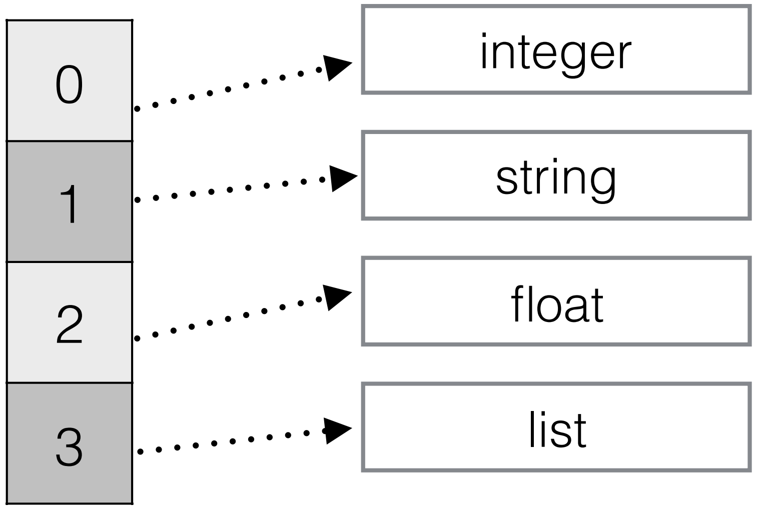
- build conditionals statements

python lists and tuples



a list can be changed

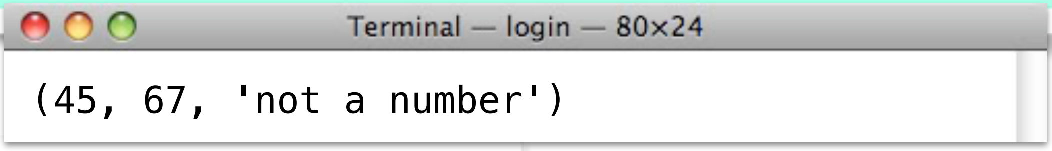




a tuple is immutable

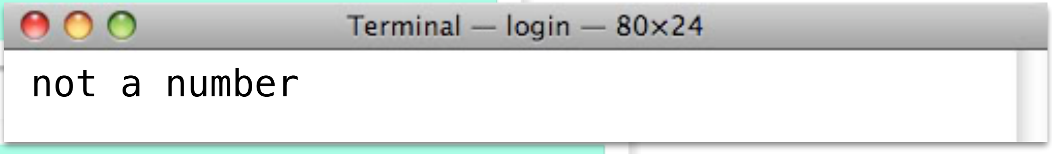
python tuples

```python
# tuples are immutable lists and are designated by commas
# you can store ANYTHING inside a tuple, its basically a complex object container
a_tuple = 45, 67, "not a number"
print a_tuple
```



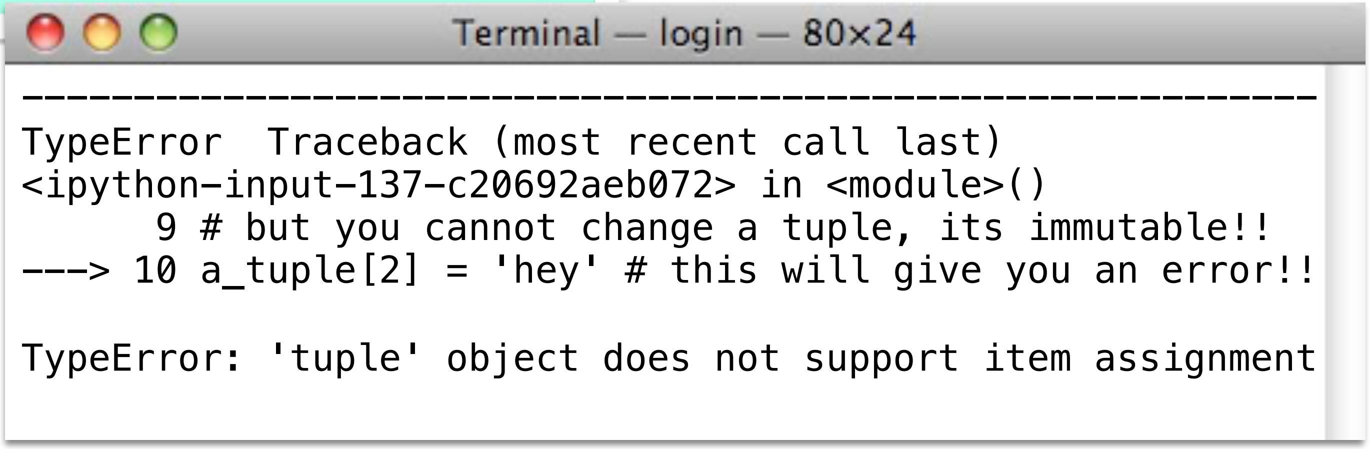
```
(45, 67, 'not a number')
```

```python
# you can access a tuple with square brackets
print a_tuple[2]
```



```
not a number
```

```python
# but you cannot change a tuple, it’s immutable!!
a_tuple[2] = 'hey' # this will give you an error!!
```



```
----------------------------------------------------------
TypeError  Traceback (most recent call last)
<ipython-input-137-c20692aeb072> in <module>()
      9 # but you cannot change a tuple, its immutable!!
---> 10 a_tuple[2] = 'hey' # this will give you an error!!

TypeError: 'tuple' object does not support item assignment
```

python lists

```
# A List is one of the most powerful tools in python from which
# most abstract types get created and implemented
# a list is very much like the mutable version of a tuple
# it can hold any type of information
a_list = [45, 67, "not a number”]

# we can add to a list through the append function
a_list.append("A string, appended as a new element in the list")
print a_list
```

```
[45, 67, 'not a number', 'A string, appended as a new element in the list']
```

```
# Lists can have other lists in them
tmp_list = ["a list", "within another list", 442]
a_list.append(tmp_list)
print a_list
```

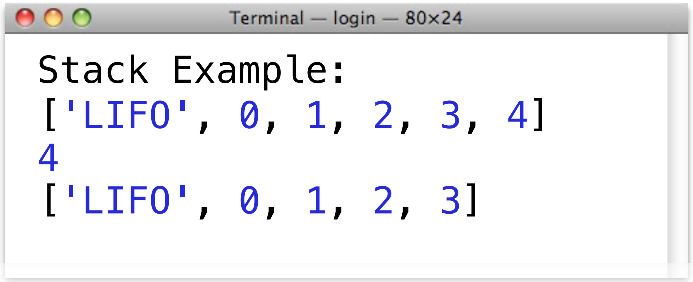
```
[45, 67, 'not a number', 'A string, appended as a new element in the list',
['a list', 'within another list', 442]]
```

```
# all of the indexing we learned from before still works with lists
print a_list[-1]
print a_list[-2:]
```

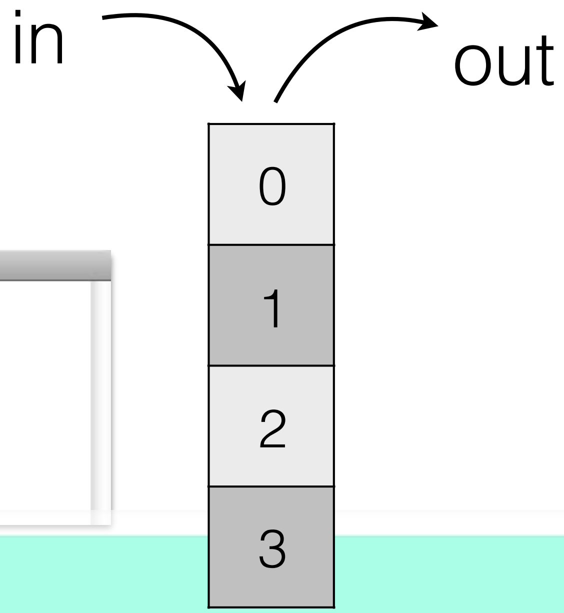
```
['a list', 'within another list', 442]
['A string, appended as a new element in the list', ['a list', 'within another
list', 442]]
```

stacks and queues

```
# list as a stack
print "\nStack Example:"
list_example = []
list_example.append('LIFO')

for i in range(0, 5):
    list_example.append(i)

print list_example
val = list_example.pop()
print val
print list_example
```

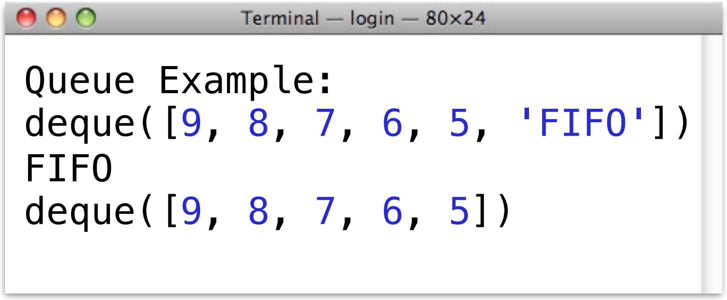


```
Stack Example:
['LIFO', 0, 1, 2, 3, 4]
4
['LIFO', 0, 1, 2, 3]
```

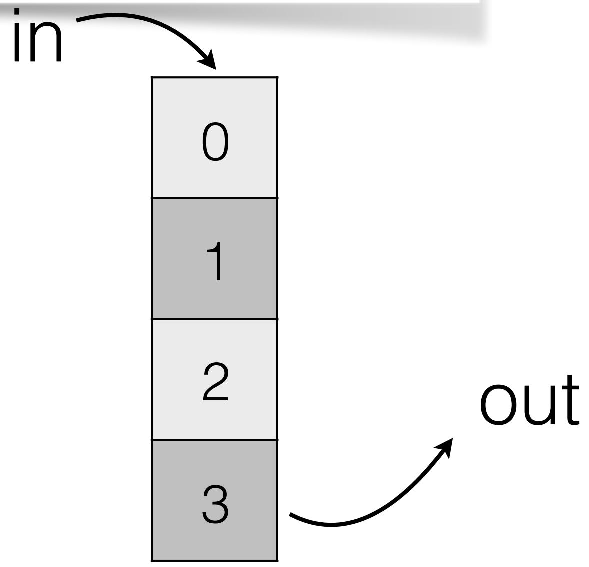


```
# list as a queue
print "\nQueue Example:"
from collections import deque # this is an import, we will get back to that later
```

```
q_example = deque()
q_example.appendleft("FIFO")
for i in range(5, 10):
    q_example.appendleft(i)

print q_example
val = q_example.pop()
print val
print q_example
```



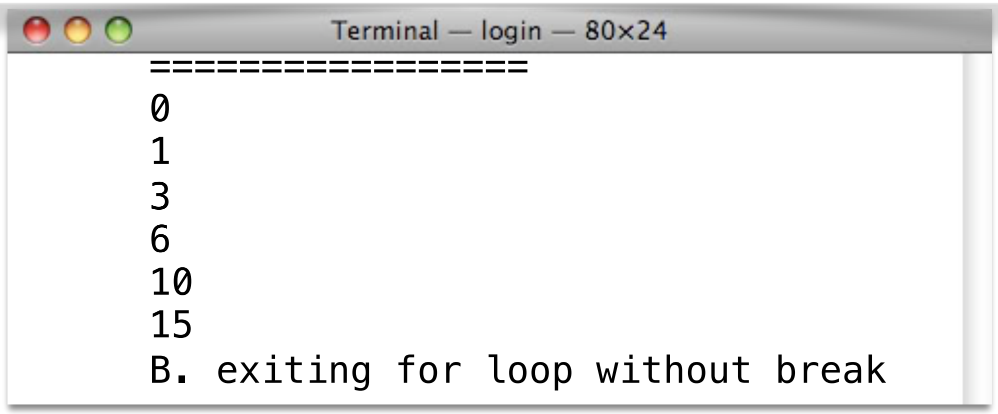
```
Queue Example:
deque([9, 8, 7, 6, 5, 'FIFO'])
FIFO
deque([9, 8, 7, 6, 5])
```



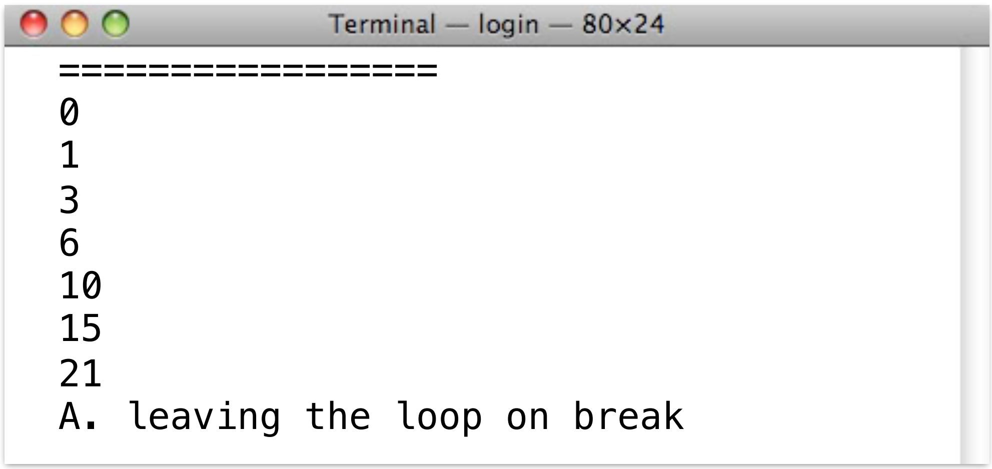
python loops

```
import random
print '================='
val = 0
for i in range(0, random.randint(1, 10) ):
    val += i
    print val
    if val>20:
        print ' A. leaving the loop on break'
        break # break out of loop
else: # this else belongs to the for loop
    print 'B. exiting for loop without break'
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9



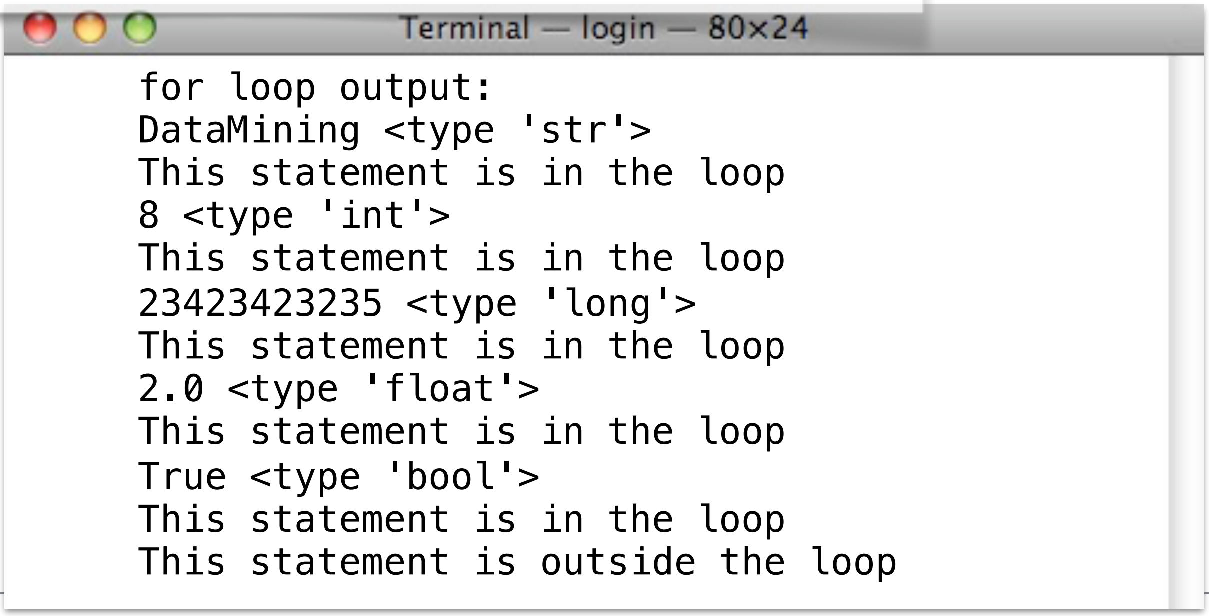
```
=================
0
1
3
6
10
15
B. exiting for loop without break
```



```
=================
0
1
3
6
10
15
21
A. leaving the loop on break
```

python loops

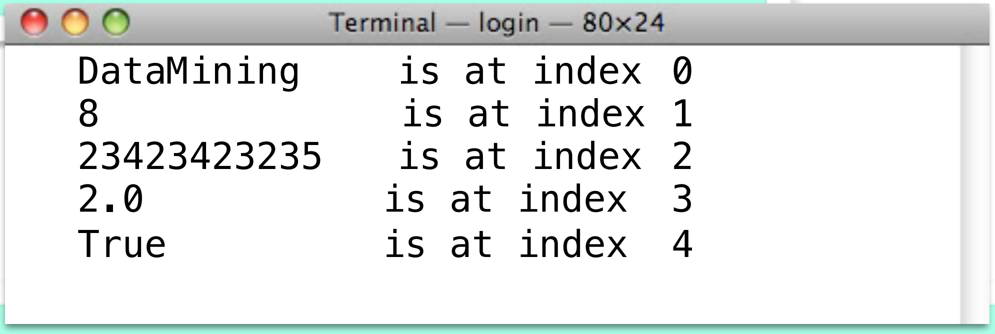
```python
#============================================
# for loop example with list
print "\nfor loop output:"

list_example = [int_val, long_val, float_val, bool_val]
list_example.insert(0, "DataMining")

# notice that the loop ends with a colon and
# is designated by the tab alignment
for val in list_example:
    print str(val) + ' ' + str(type(val))
    print "This statement is in the loop"

print "This statement is outside the loop"
```



```
for loop output:
DataMining <type 'str'>
This statement is in the loop
8 <type 'int'>
This statement is in the loop
23423423235 <type 'long'>
This statement is in the loop
2.0 <type 'float'>
This statement is in the loop
True <type 'bool'>
This statement is in the loop
This statement is outside the loop
```

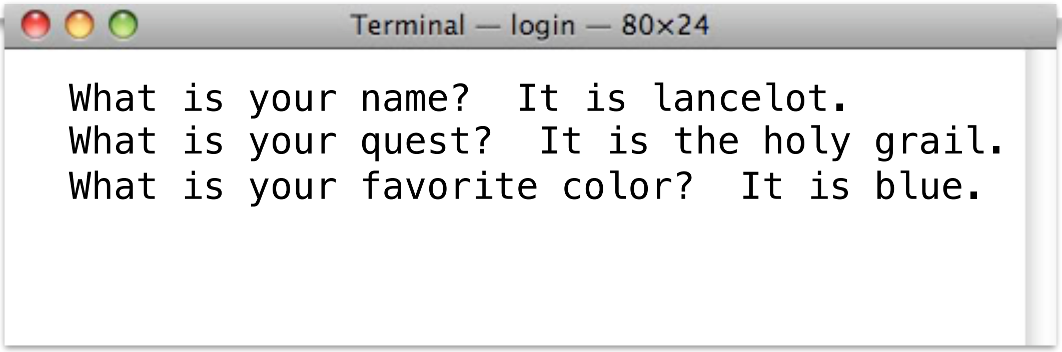
python loops with lists

```python
# you can also get the index using the enumerate example
for index, val in enumerate(list_example):
    print str(val), '\t is at index \t', index
```



```
DataMining     is at index  0
8               is at index  1
23423423235    is at index  2
2.0            is at index  3
True           is at index  4
```

```python
# this is a classic example for zipping, provided by the official python tutorial
# notice the references to Monty Python

# say you have two lists of equal size that you would like to
# loop through without indexing, you can use the "zip" function
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print 'What is your %s?  It is %s.' % (q, a)
```



```
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

building a functional calculator

- using reverse polish notation
- stacks
- user input