Scripting for Data Science in Python and R

SMU Interdisciplinary Master's Degree in Data Science

Unit 2 - 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 2 - II. more python basics

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

pythonic

- many different coding “styles”
- “best” styles get the distinction of “pythonic”
    - ill-formed definition
    - changes as the language matures
- pythonic code is:
    - simple and readable
    - uses dynamic typing when possible
- …or to quote Tim Peters…

python zen

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality be
Errors should never pass
Unless explicitly silenc
In the face of ambiguity                                 ss.
There should be one-- an                                 s way to do it.
Although that way may no                                 ou're Dutch.
Now is better than never.
Although never is often better tha  right now.
If the implementation is hard to e plain, it's a bad idea.
If the implementation is easy to e plain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

type this

get this

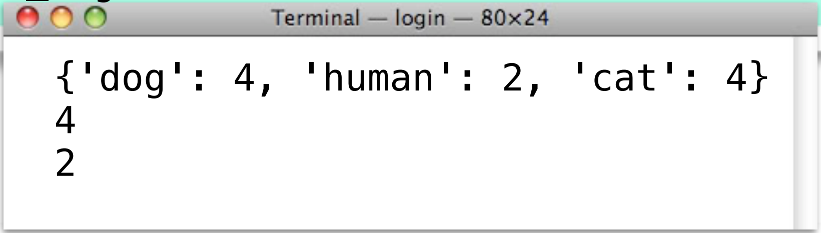
python is quirky
but, don't assume that means
it is not a **serious** tool

pythonic conventions

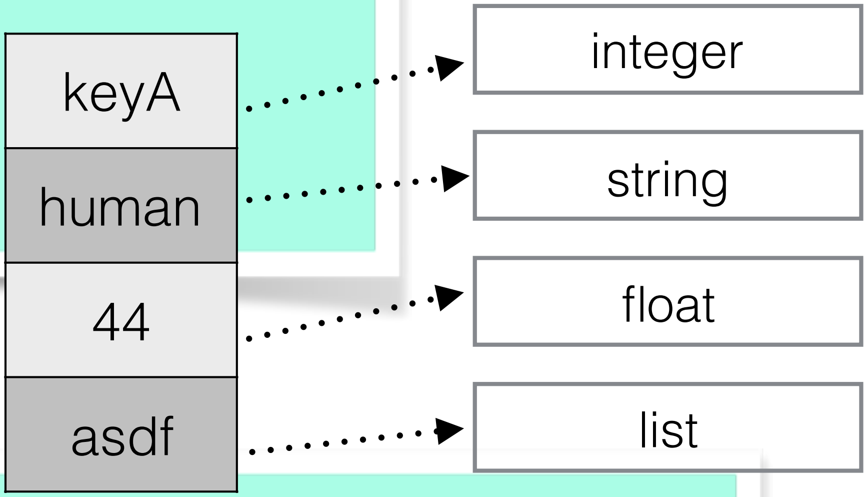
- too many to go over
    - check out: http://legacy.python.org/dev/peps/pep-0008/
- name variables with underscore: `this_is_my_variable`
- use indentation to increase clarity
- one space before and after = i.e., `x = 5`
- space mathematics well `hypot2 = x*x + y*y`

more pythonic: dictionaries

```python
# Dictionaries map keys to values.
# Here we set up a key as a string and the value as a number
num_legs = { 'dog': 4, 'cat': 4, 'human': 2 }

# You access Subscripts via the "key"
print num_legs
print num_legs['dog']
print num_legs['human']
```



```
{'dog': 4, 'human': 2, 'cat': 4}
4
2
```

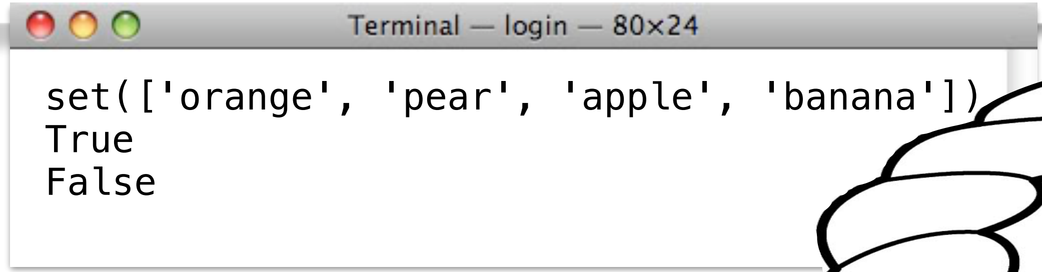


```python
# Entries can be added, updated, or deleted.
# Again, these are just containers for any memory type
num_legs['human'] = 'two'
num_legs['bird'] = 'two and some wings'
num_legs[45] = 'a key that is not a string' # notice that key is not a string
# the key just needs to be some immutable memory

del num_legs['cat']
print num_legs
```

```
{'bird': 'two and some wings', 45: 'a key that is not a string', 'dog': 4,
'human': 'two'}
```

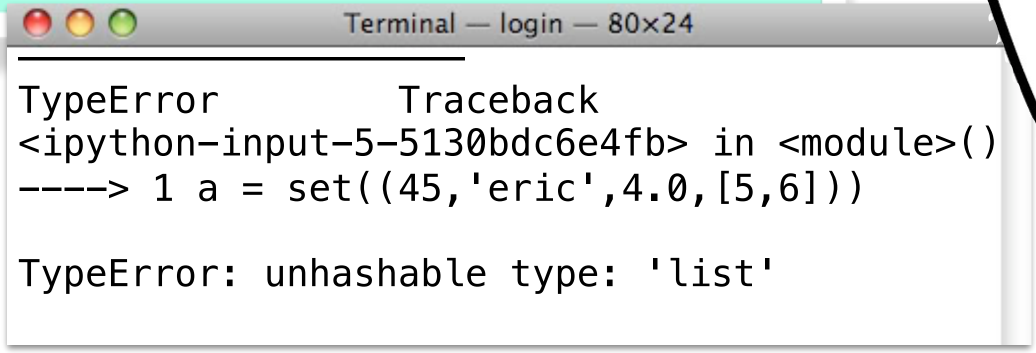
more pythonic: sets

```python
# Sets, taken from the Python sets tutorial
# https://docs.python.org/2/tutorial/datastructures.html#sets
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
fruit = set(basket) # create a set without duplicates
print fruit
print 'orange' in fruit # fast membership testing
print 'crabgrass' in fruit
```

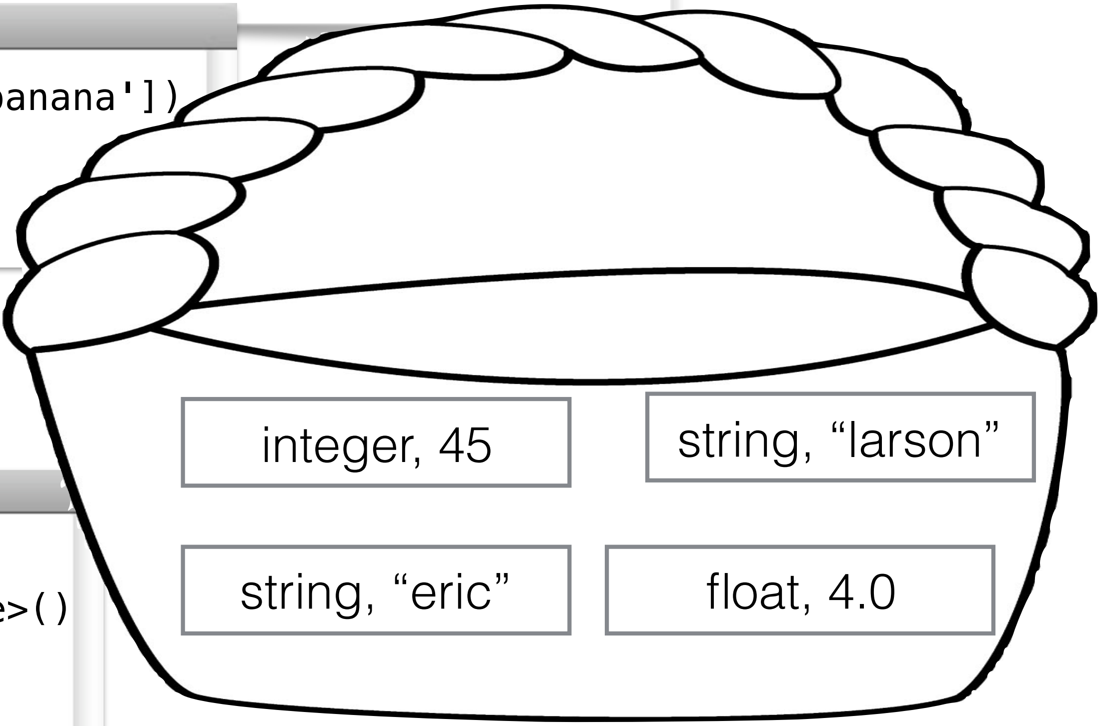


```
set(['orange', 'pear', 'apple', 'banana'])
True
False
```

```python
a = set((45,'eric',4.0,[5,6]))
```



```
TypeError                 Traceback
<ipython-input-5-5130bdc6e4fb> in <module>()
----> 1 a = set((45,'eric',4.0,[5,6]))

TypeError: unhashable type: 'list'
```

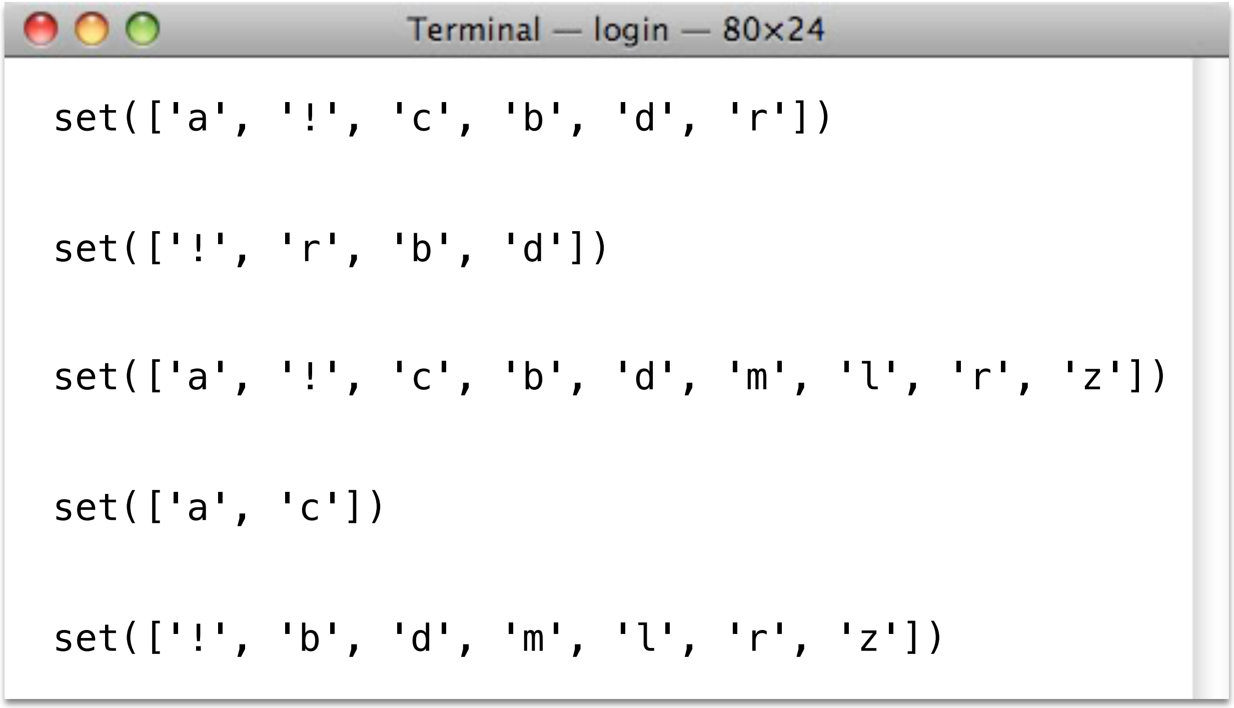


a basket of **hashable** data

more pythonic: sets

```python
# Demonstrate set operations on unique letters from two words
a = set('abracadabra')
b = set('alacazam')
a.add('!') # also add the some punctuation
```

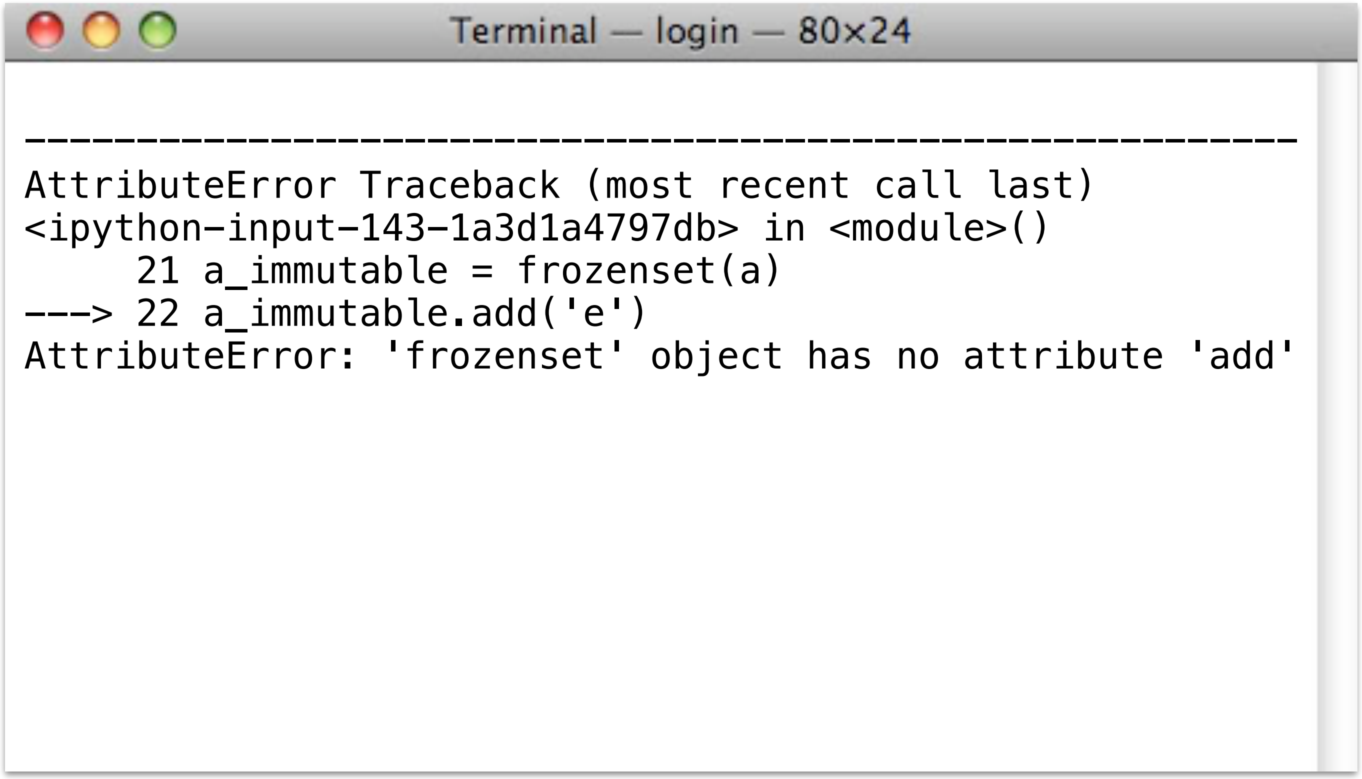
```python
# set operations
print a   # unique letters


print a - b # in a but not in b


print a | b # either a or b


print a & b # both a and b


print a ^ b # a or b, not both
```



```
set(['a', '!', 'c', 'b', 'd', 'r'])


set(['!', 'r', 'b', 'd'])


set(['a', '!', 'c', 'b', 'd', 'm', 'l', 'r', 'z'])


set(['a', 'c'])


set(['!', 'b', 'd', 'm', 'l', 'r', 'z'])
```

more pythonic: immutable sets

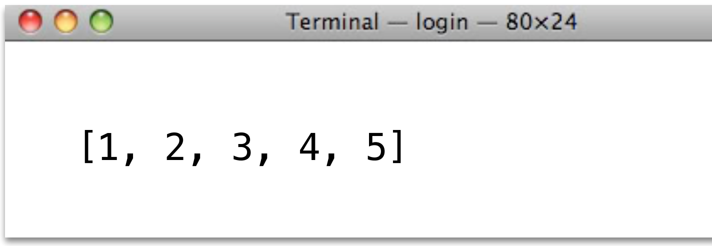
```
a_immutable = frozenset(a)
a_immutable.add('e')   # the set is immutable, so we cannot add to it, this will
give an error!
```



```
---------------------------------------------------------------------------
AttributeError Traceback (most recent call last)
<ipython-input-143-1a3d1a4797db> in <module>()
     21 a_immutable = frozenset(a)
---> 22 a_immutable.add('e')
AttributeError: 'frozenset' object has no attribute 'add'
```

more pythonic: functions

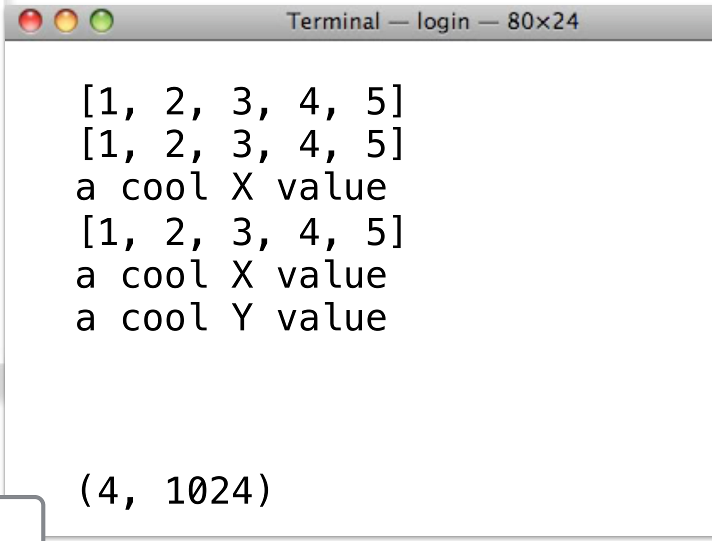
```python
# more functions examples
def show_data(data):
   print data

some_data = [1,2,3,4,5]
show_data(some_data)
```



```
[1, 2, 3, 4, 5]
```

```python
# you can also define default values for the functions
def show_data(data,x=None,y=None):
    # print the data
    print data
    if x is not None:
        print x
    if y is not None:
        print y

some_data = [1,2,3,4,5]
show_data(some_data);
show_data(some_data,x='a cool X value')
show_data(some_data,y='a cool Y value',x='a cool X value')
```

default value



```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
a cool X value
[1, 2, 3, 4, 5]
a cool X value
a cool Y value



(4, 1024)
```

```python
# as well as have multiple return types in the function
def get_square_and_tenth_power(x):
    return x**2,x**10

print get_square_and_tenth_power(2)
```

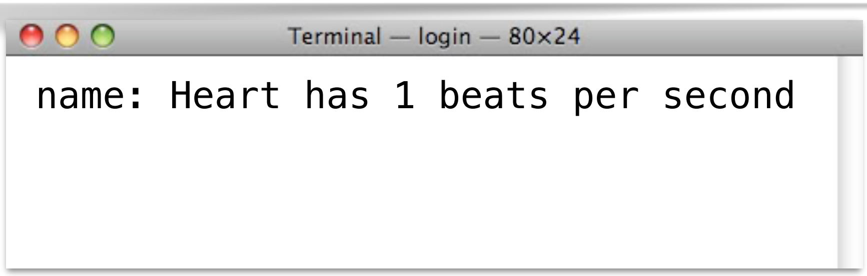
return a tuple

calculator example 3

- making the code more pythonic

classes

```python
# This is a class that inherits from a generic object
class BodyPart(object):
    # this is a class variable, shared across all instances
    def __init__(self,name):
        self.name = name # the name attribute is unique to each instance of the class

# now define a class that sub classes from the defined BodyPart CLass
class Heart(BodyPart):
    def __init__(self,rate=60,units="minute"):
        self.rate = rate
        self.units= units
        super(Heart,self).__init__("Heart")

    def print_rate(self):
        print "name: " + str(self.name) + " has " + str(self.rate) + " beats per " +
                 self.units


my_heart = Heart(1,"second")
my_heart.print_rate()
```

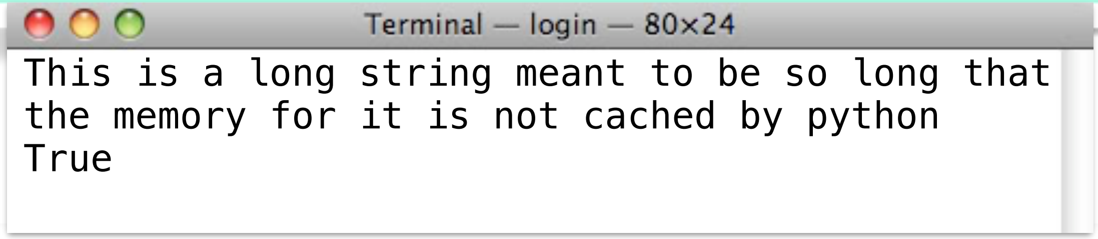


```
name: Heart has 1 beats per second
```

classes

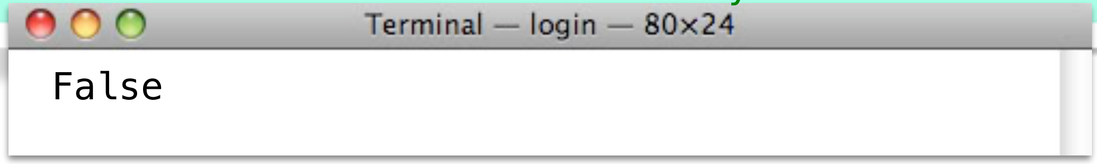
```
# This is a class that inherits from a generic object
class BodyPart(object):
    kind = "This is a long string meant to be so long that the memory for it is not
            cached by python"

   # this is a class variable, shared across all instances
   def __init__(self,name):
       self.name = name # the name attribute is unique to each instance of the class

my_heart = Heart(1,"second")
generic_part = BodyPart("Foot")
print my_heart.kind
print my_heart.kind is generic_part.kind # true, these are the same memory location
```

class shared variable



```
This is a long string meant to be so long that
the memory for it is not cached by python
True
```

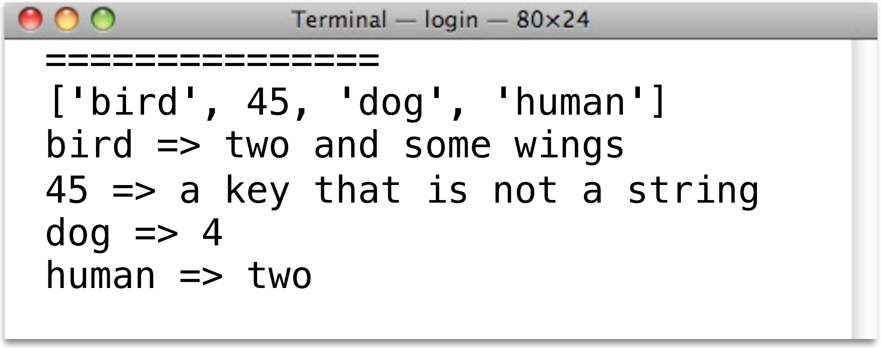
```
# take the following for example, these are not the same object
a = "This is a long string meant to be so long that the memory for it is not cached by
python"
b = "This is a long string meant to be so long that the memory for it is not cached by
python"
print a is b # not the same memory location
```



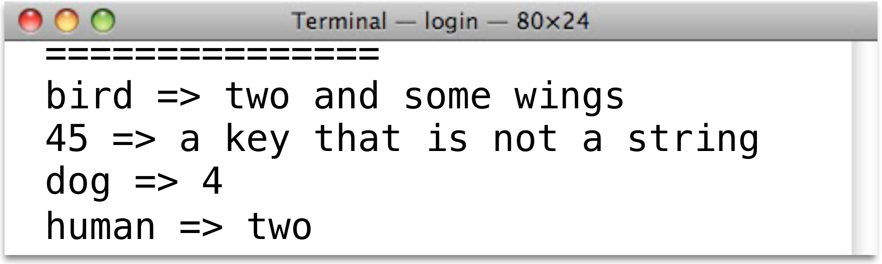
```
False
```

python loops with dictionaries

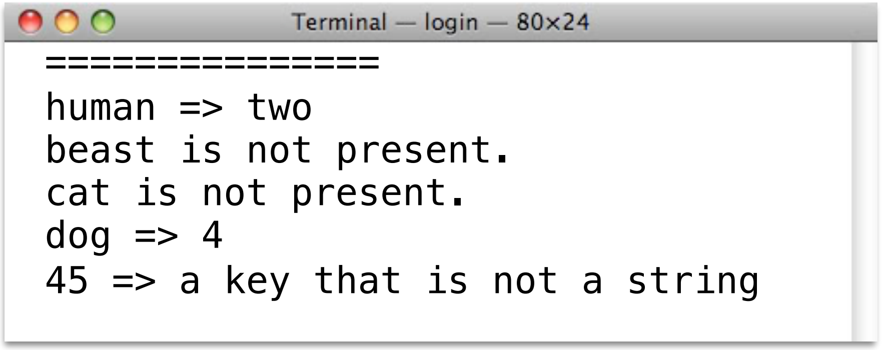
```
# Looping through dictionaries
print '================'
# Get all the keys.
print num_legs.keys()
for k in num_legs.keys():
    print k, "=>", num_legs[k]

print '================'
# you can also use the iter_items function
for k, v in num_legs.items():
    print k, "=>", v

print '================'
# Test for presence of a key.
for t in [ 'human', 'beast', 'cat', 'dog', 45 ]:
    print t,
    if t in num_legs:
        print '=>', num_legs[t]
    else:
        print 'is not present.'
```



```
================
['bird', 45, 'dog', 'human']
bird => two and some wings
45 => a key that is not a string
dog => 4
human => two
```



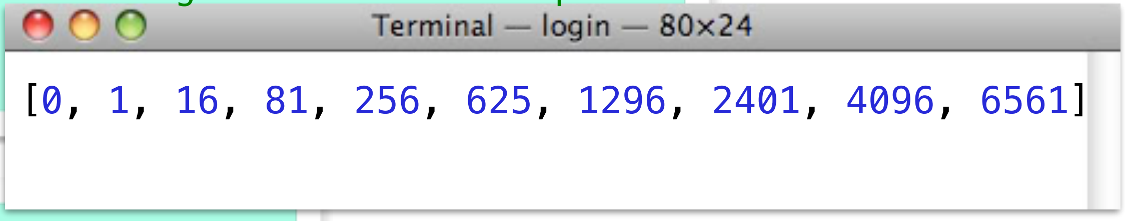
```
================
bird => two and some wings
45 => a key that is not a string
dog => 4
human => two
```



```
================
human => two
beast is not present.
cat is not present.
dog => 4
45 => a key that is not a string
```

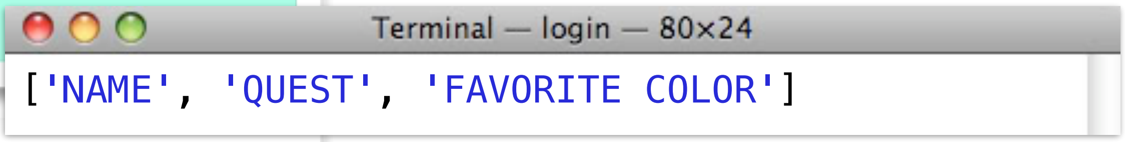
comprehensions

```
# imagine we want to take every element in a range to the fourth power
times_four = [x**4 for x in range(10)]
print times_four
```



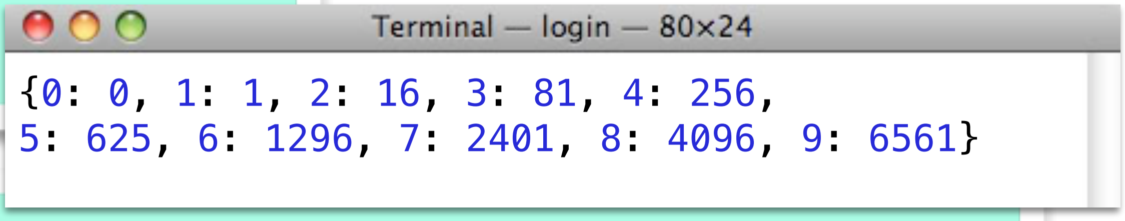
```
[0, 1, 16, 81, 256, 625, 1296, 2401, 4096, 6561]
```

```
# you can also call functions inside a comprehension
questions = ['name', 'quest', 'favorite color']
quest_upper = [x.upper() for x in questions]
print quest_upper
```



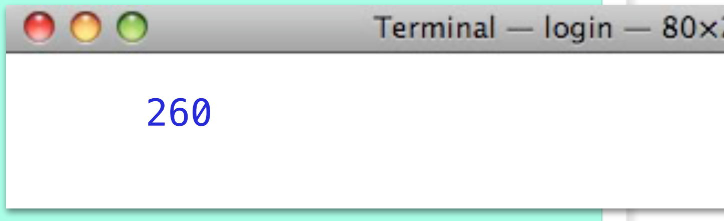
```
['NAME', 'QUEST', 'FAVORITE COLOR']
```

```
# you can also do comprehensions with dictionaries
times_four = {x:x**4 for x in range(10)}
# notice curly braces and key placement
print times_four
```



```
{0: 0, 1: 1, 2: 16, 3: 81, 4: 256,
5: 625, 6: 1296, 7: 2401, 8: 4096, 9: 6561}
```

```
# Finally, all of the enumerate, zipping, and slicing we performed also applies to
# list comprehensions
x_array = [10, 20, 30]
y_array = [7, 5, 3]
# this prints the sum of the multiplication of the arrays
print sum(x*y for x,y in zip(x_array, y_array))
```



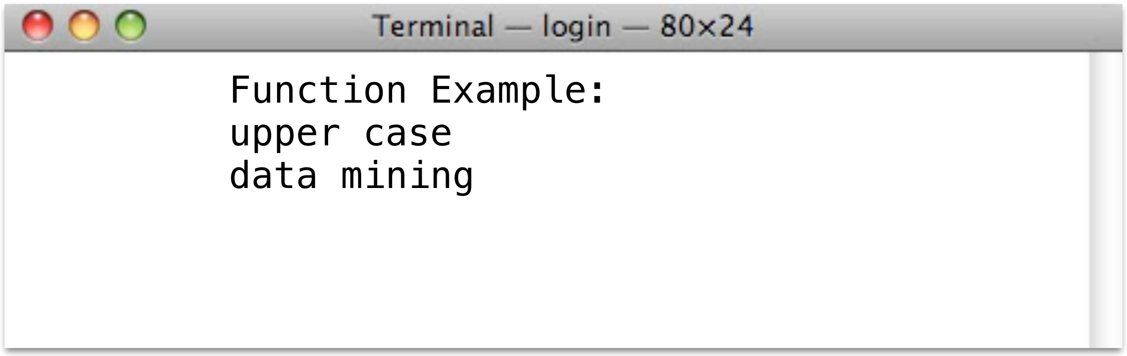
```
260
```

calculator example 4

- making the code more object-oriented
- separating model and view controller

exceptions

```python
# create and call a function
# the function can be defined almost anywhere in file, as long as it is defined
before it gets used
def make_strings_lowercase(str_input):
    assert isinstance(str_input, str)  # test the type of input
    return str_input.lower()

# now we are back on the main execution
print make_strings_lowercase("UPPER CASE")
print make_strings_lowercase("Data Mining")
```

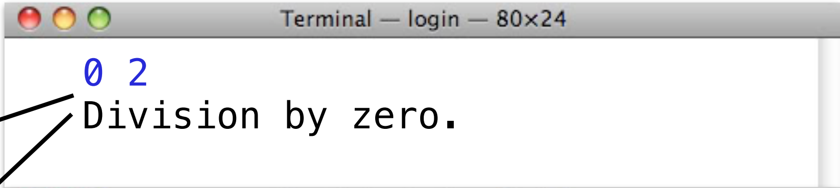


```
Function Example:
upper case
data mining
```

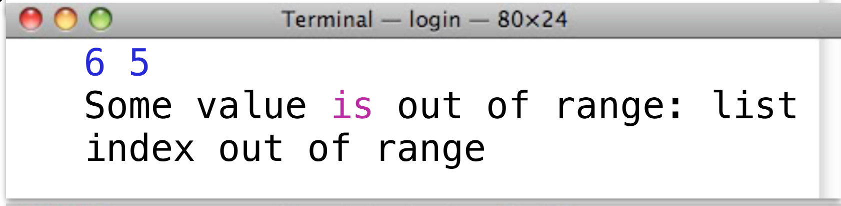
exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

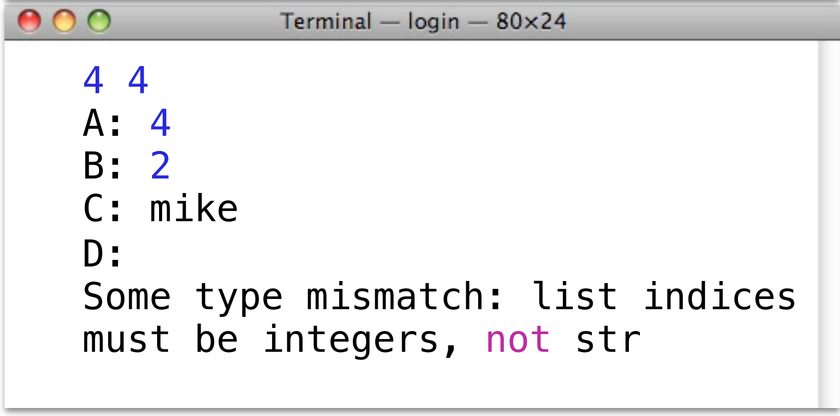
```python
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```



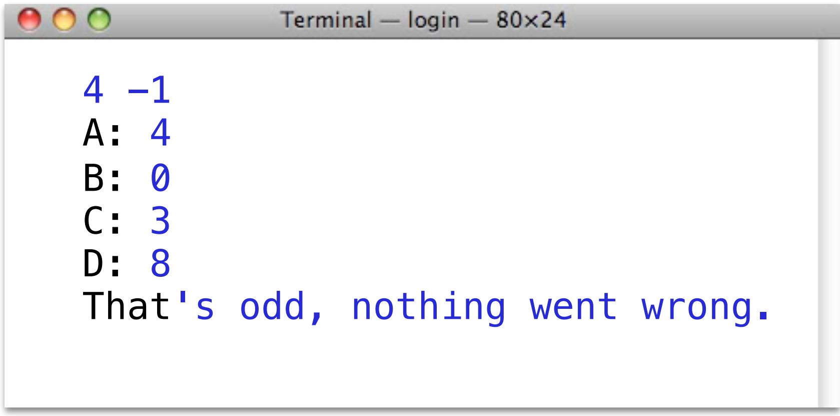
```
0 2
Division by zero.
```



```
6 5
Some value is out of range: list
index out of range
```



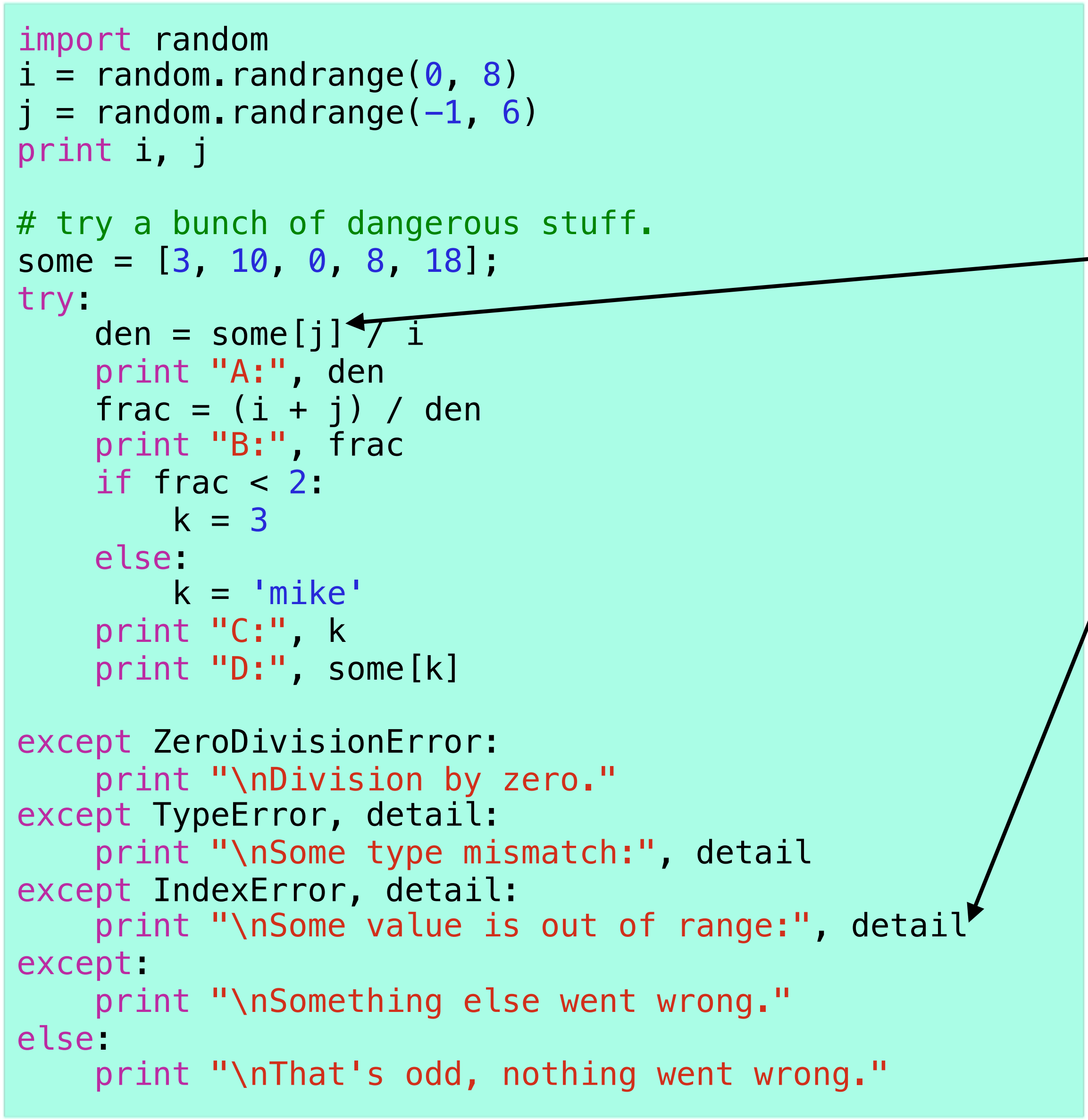
```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```



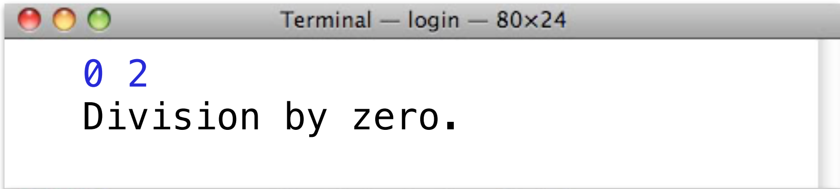
```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

exception handling

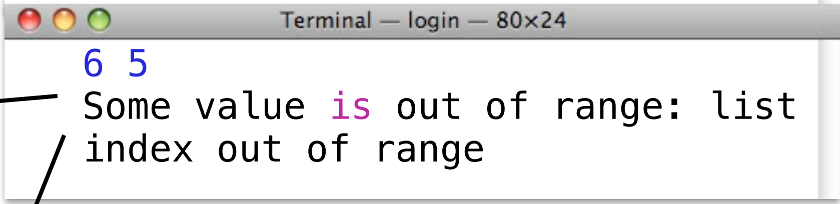
```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```



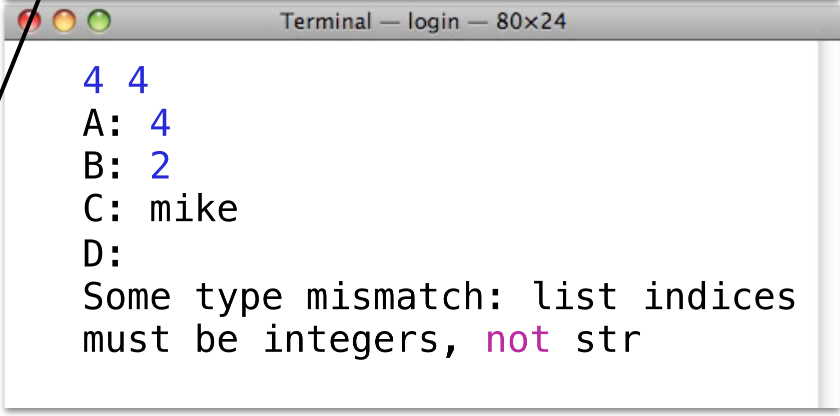
```python
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```



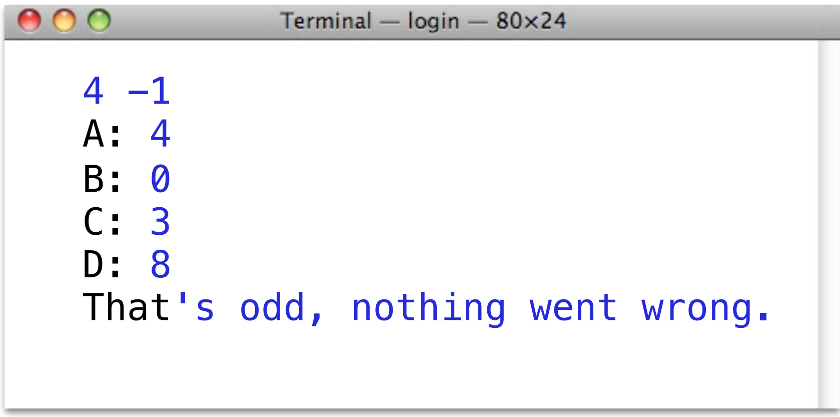
```
0 2
Division by zero.
```



```
6 5
Some value is out of range: list
index out of range
```



```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

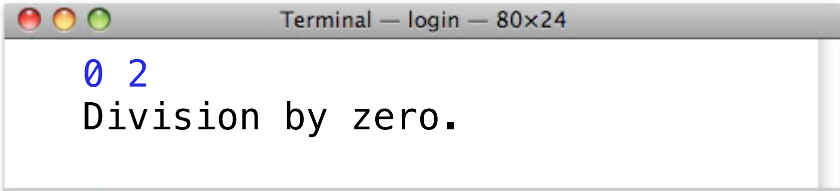


```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

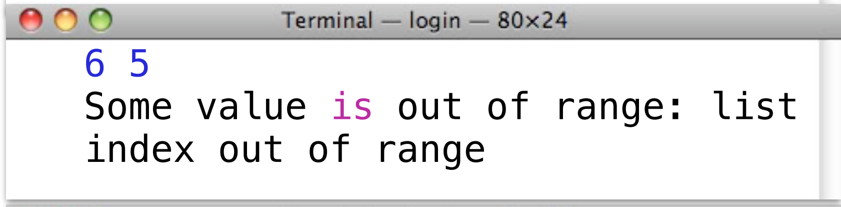
exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

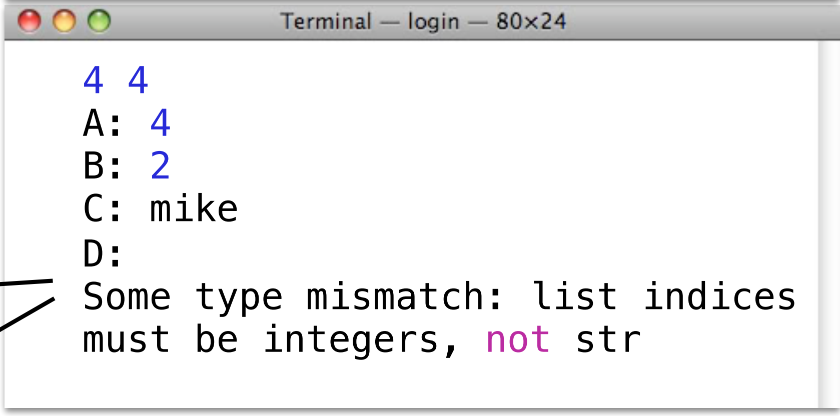
```
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```



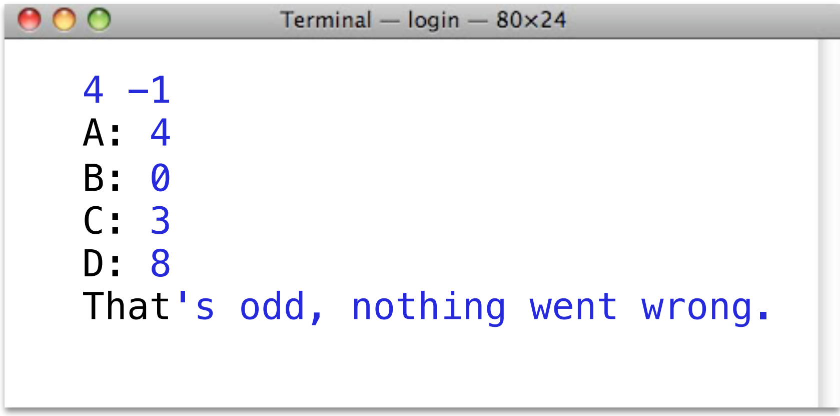
```
0 2
Division by zero.
```



```
6 5
Some value is out of range: list
index out of range
```



```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```

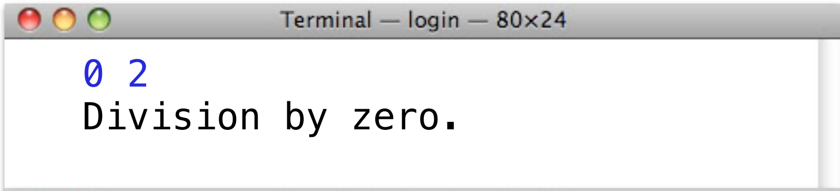


```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

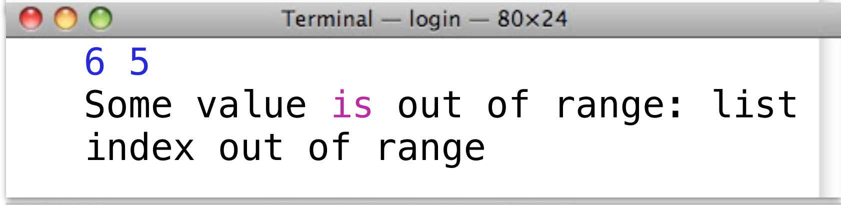
exception handling

```
# example reissued from: http://sandbox.mc.edu/~bennet/python/code/exc_py.html
```

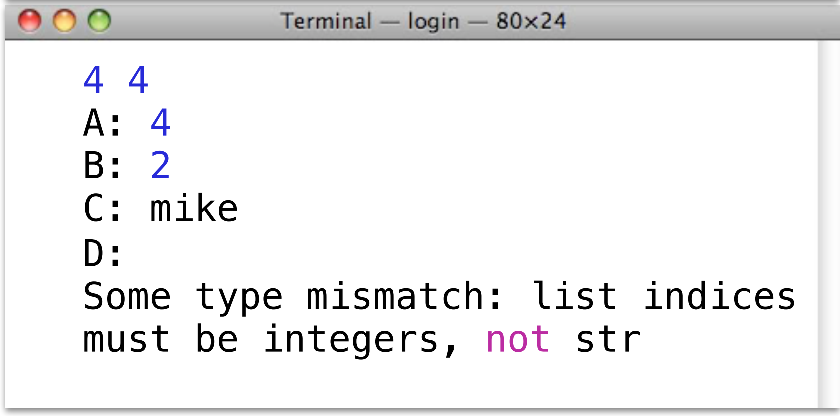
```python
import random
i = random.randrange(0, 8)
j = random.randrange(-1, 6)
print i, j

# try a bunch of dangerous stuff.
some = [3, 10, 0, 8, 18];
try:
    den = some[j] / i
    print "A:", den
    frac = (i + j) / den
    print "B:", frac
    if frac < 2:
        k = 3
    else:
        k = 'mike'
    print "C:", k
    print "D:", some[k]

except ZeroDivisionError:
    print "\nDivision by zero."
except TypeError, detail:
    print "\nSome type mismatch:", detail
except IndexError, detail:
    print "\nSome value is out of range:", detail
except:
    print "\nSomething else went wrong."
else:
    print "\nThat's odd, nothing went wrong."
```



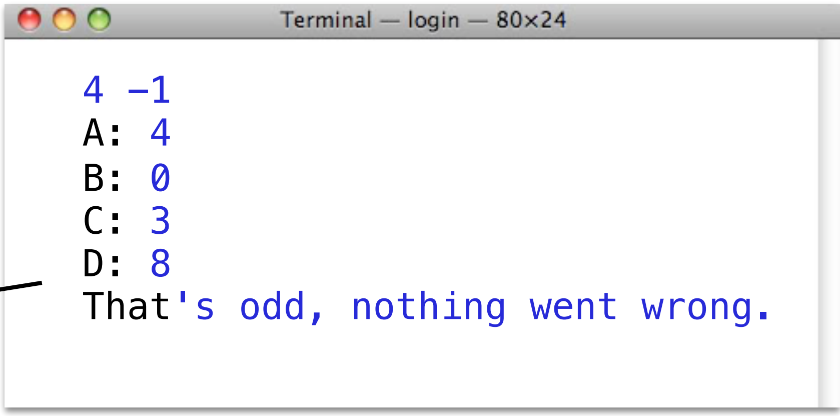
```
0 2
Division by zero.
```



```
6 5
Some value is out of range: list
index out of range
```



```
4 4
A: 4
B: 2
C: mike
D:
Some type mismatch: list indices
must be integers, not str
```



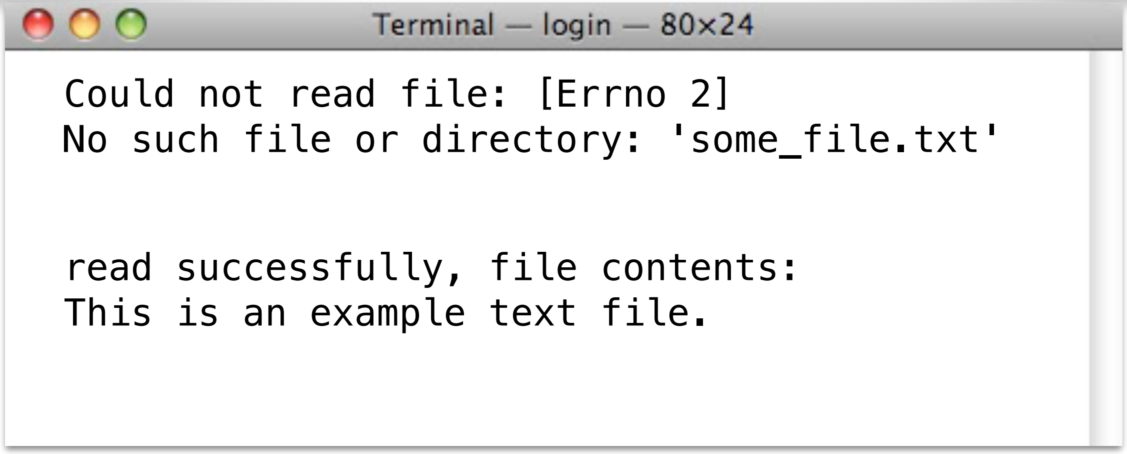
```
4 -1
A: 4
B: 0
C: 3
D: 8
That's odd, nothing went wrong.
```

calculator example 5

- add exception handling to class
- adding inheritance for better error handling

opening a file

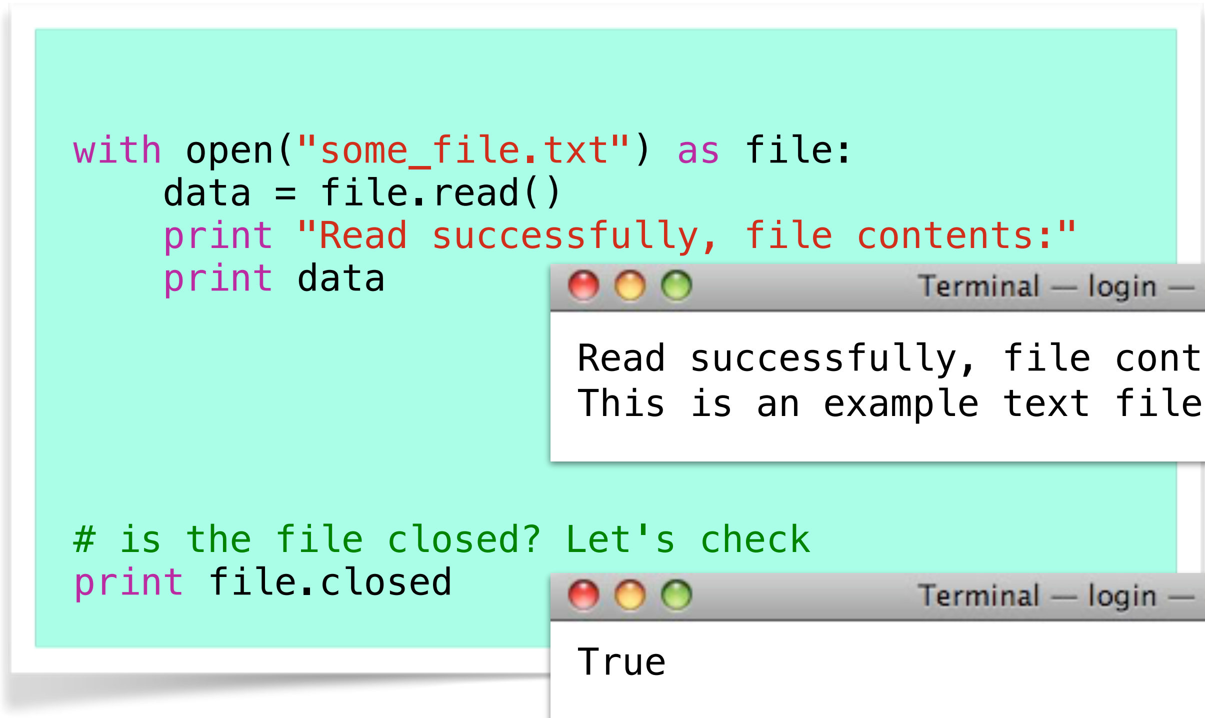
```python
# the regular way of opening a file, lots of error checking
try:
    file = open("some_file.txt")
    data = file.read()
except IOError, detail:
    print "\nCould not read file:", detail
else:
    print "Read successfully, file contents:"
    print data
finally:
    # this always gets called, close the file if it’s open
    if not file.closed:
        file.close()
```



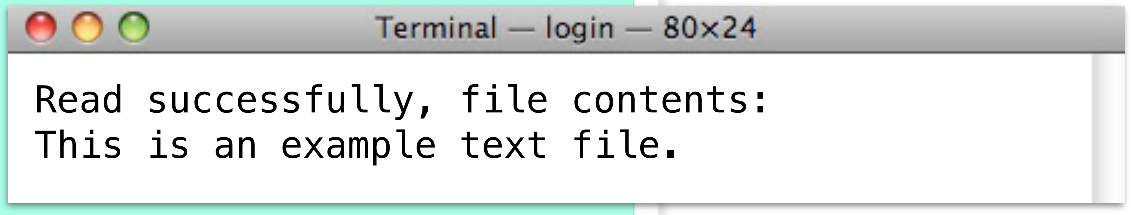
```
Could not read file: [Errno 2]
No such file or directory: 'some_file.txt'


read successfully, file contents:
This is an example text file.
```

we can get rid of this through OOP!

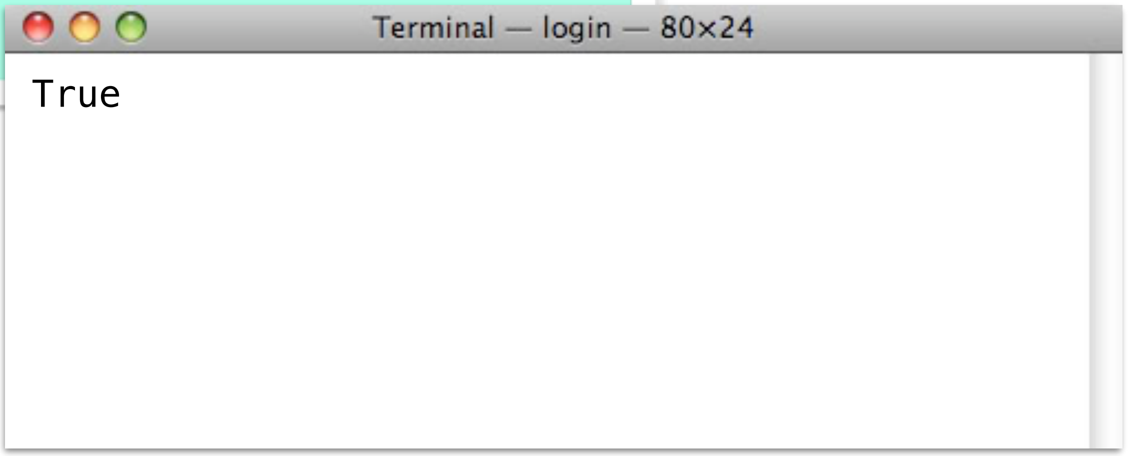
opening a file using “with”



```
with open("some_file.txt") as file:
    data = file.read()
    print "Read successfully, file contents:"
    print data


# is the file closed? Let's check
print file.closed
```

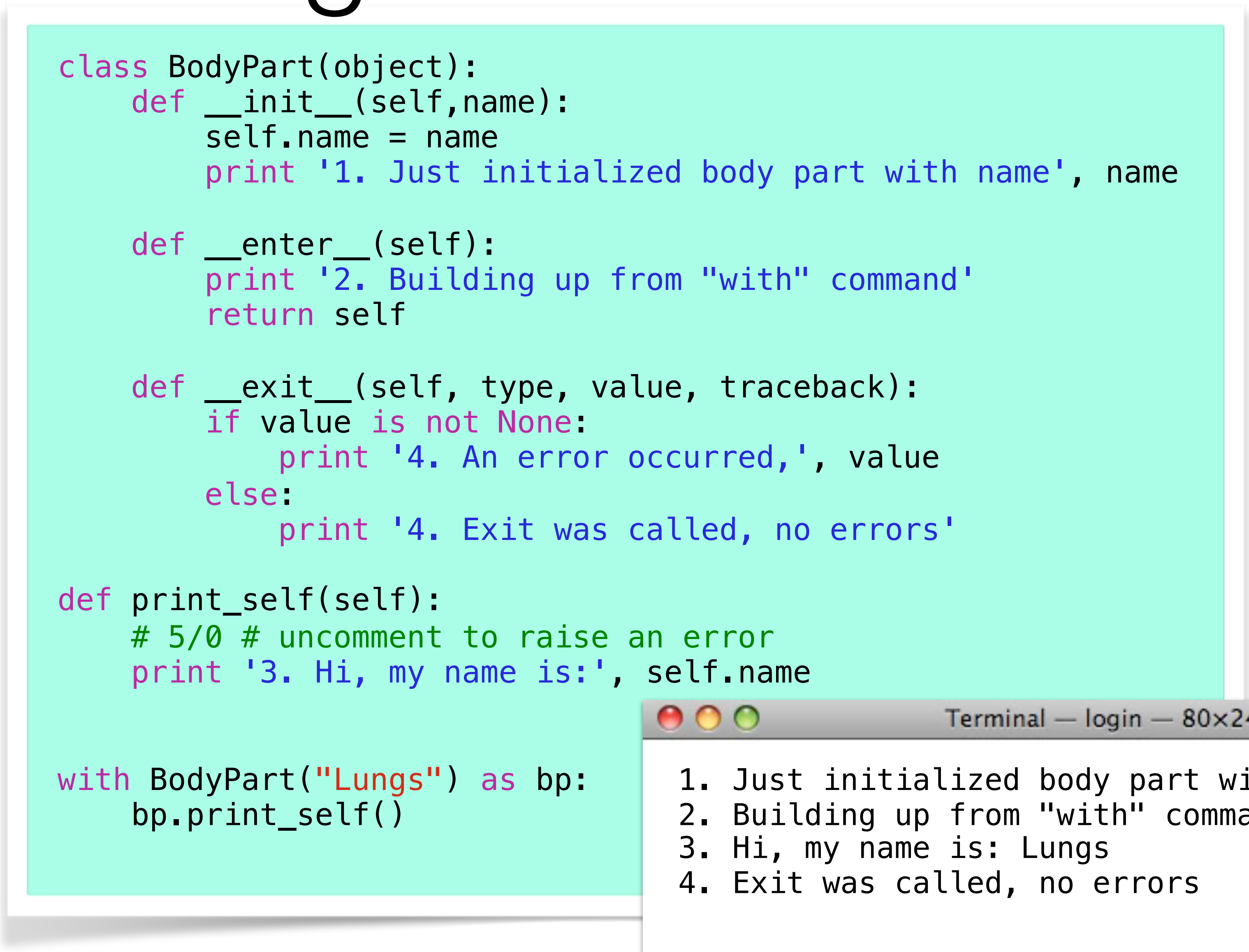


```
Read successfully, file contents:
This is an example text file.
```

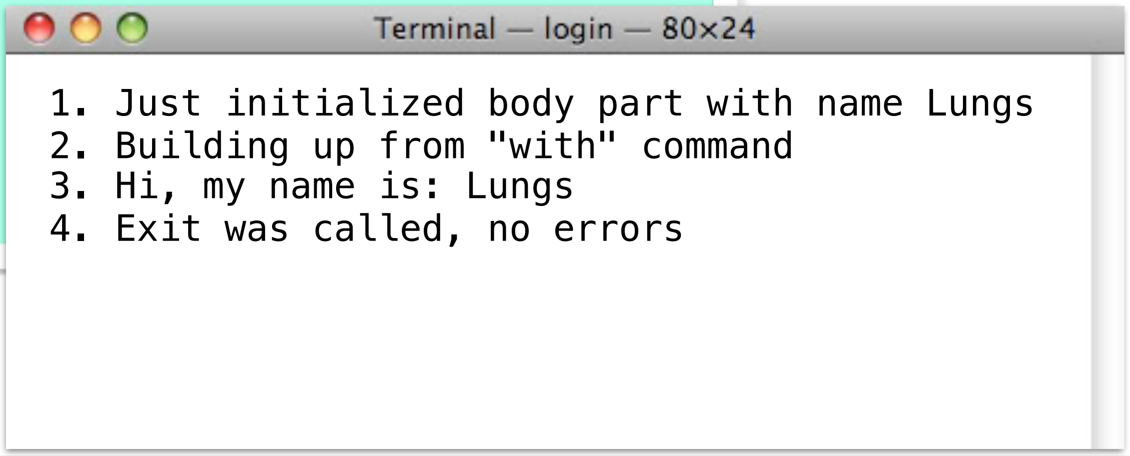


```
True
```

writing a class to use “with”



```python
class BodyPart(object):
    def __init__(self,name):
        self.name = name
        print '1. Just initialized body part with name', name

    def __enter__(self):
        print '2. Building up from "with" command'
        return self

    def __exit__(self, type, value, traceback):
        if value is not None:
            print '4. An error occurred,', value
        else:
            print '4. Exit was called, no errors'

def print_self(self):
    # 5/0 # uncomment to raise an error
    print '3. Hi, my name is:', self.name


with BodyPart("Lungs") as bp:
    bp.print_self()
```

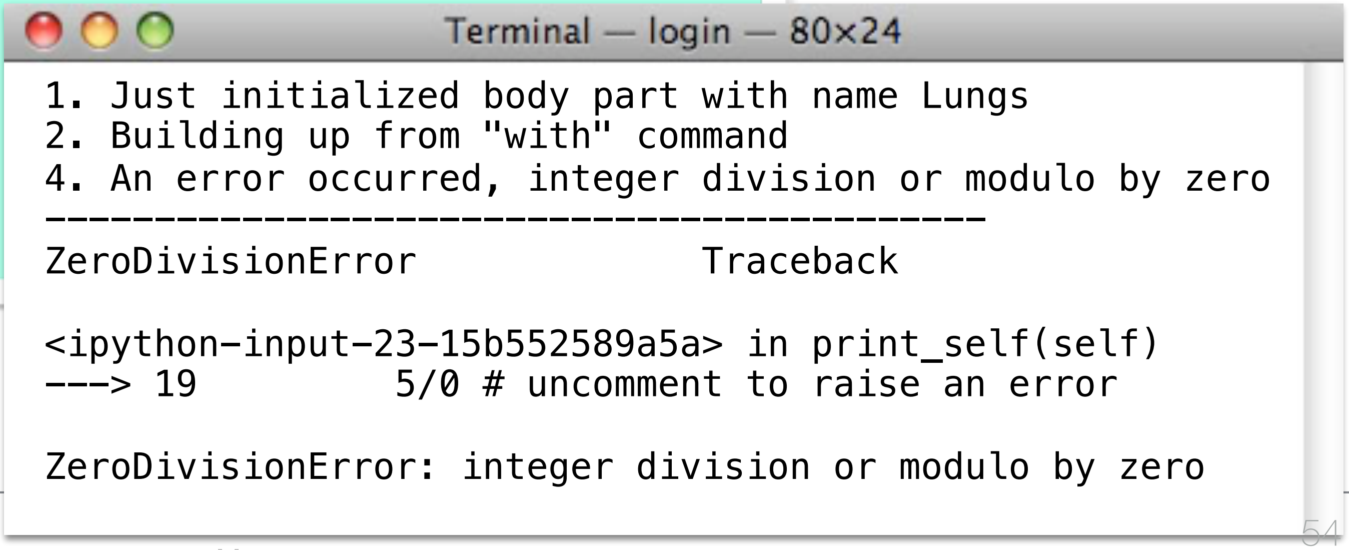


```
1. Just initialized body part with name Lungs
2. Building up from "with" command
3. Hi, my name is: Lungs
4. Exit was called, no errors
```

writing a class to use “with”

```python
class BodyPart(object):
    def __init__(self,name):
        self.name = name
        print '1. Just initialized body part with name', name

    def __enter__(self):
        print '2. Building up from "with" command'
        return self

    def __exit__(self, type, value, traceback):
        if value is not None:
            print '4. An error occurred,', value
        else:
            print '4. Exit was called, no errors'

def print_self(self):
    5/0 # uncomment to raise an error
    print '3. Hi, my name is:', self.name


with BodyPart("Lungs") as bp:
    bp.print_self()
```

divide by zero!



```
1. Just initialized body part with name Lungs
2. Building up from "with" command
4. An error occurred, integer division or modulo by zero
-------------------------------------------
ZeroDivisionError                         Traceback

<ipython-input-23-15b552589a5a> in print_self(self)
---> 19         5/0 # uncomment to raise an error

ZeroDivisionError: integer division or modulo by zero
```

calculator example 6

- adding custom functions from a user file
- possible exercises to extend your python programming knowledge