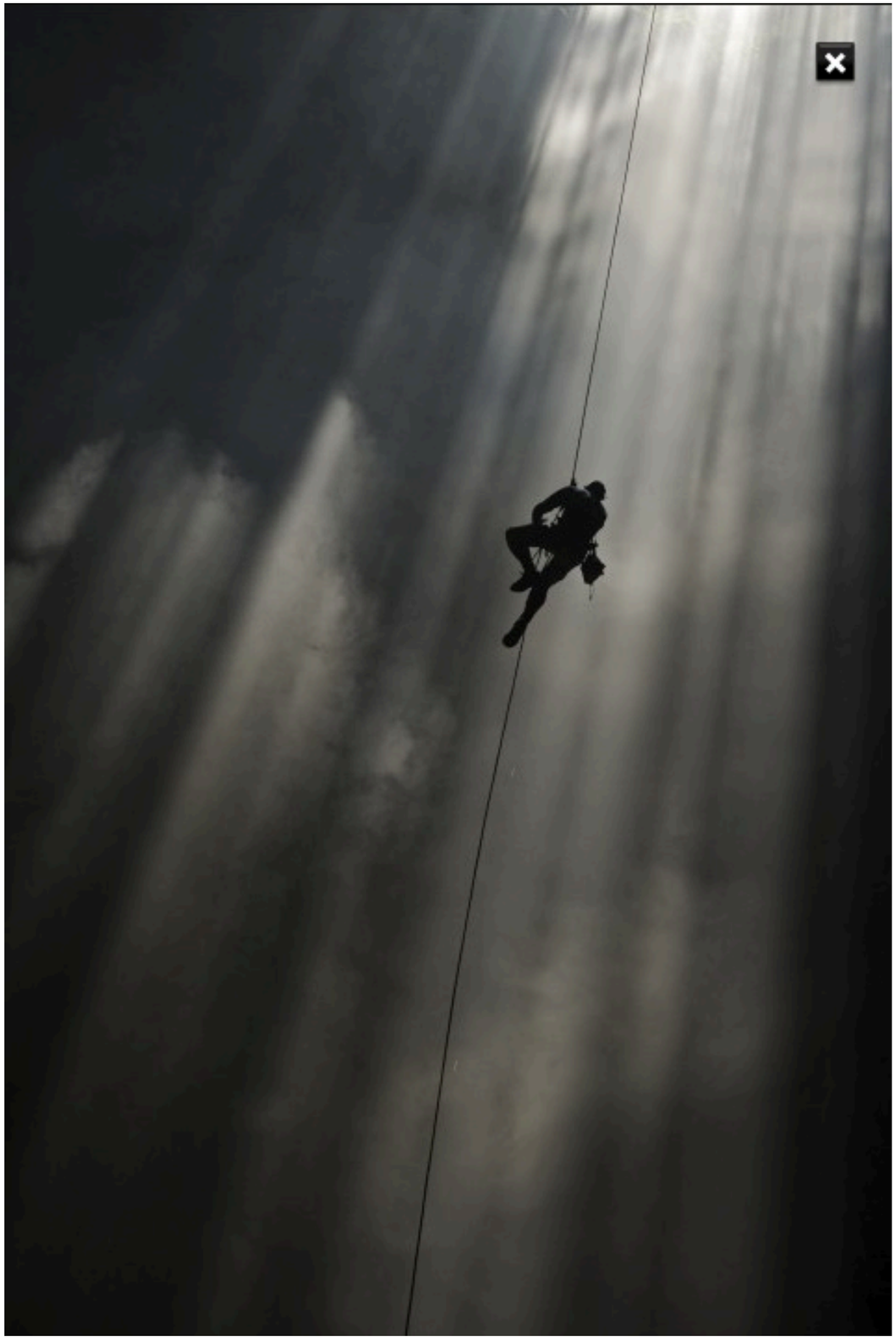

Working with Strings

Lecture 14

February 10th, 2012

Mathieu Perreault

Regular expressions

- *import re* - re is the regular expression module (http://docs.python.org/library/re.html)

  - *re.findall(‘<regexp>’,x)* - find all places where the regular expression matches something in x

```
>>> re.findall("[a-z]+", “test33index.py”)
['test', 'index', 'py']
```

- *re.search(‘<regexp>’,x)* - finds all the places where the regular expression matches something in x, where x is a string.

  - Returns a *MatchObject*

The MatchObject

- m.group() - return the matched string
- m.groups() - return the sequence of group matches
  - m.groups()[i] - return the ith match group.

- Examples:
  - *m = re.search('([0-9]+).+\.([a-z]+)','test33index.py')*
  - *m.group() -> '33index.py'*
  - *m.groups() -> ('33', 'py')*
  - *m.groups()[1] -> '33' m.groups()[2] -> 'py'*

More on string substitutions

- Main substitution types: %s (string), %d (digits/integers), %f (float)
- Not only used with print! You can use this anywhere.

```
mylist = []
first = John
last = Smith
mylist.append(“McGill,%s,%s” % (first, last))
mylist.append(“McGill,” + first + “,” + last) #Equivalent but more
complicated
```

Converting a file to another format

- Use the string substitution trick to simply transform a file.

- For example, from tab-separated (‘\t’) file to a comma-separated file.

```
ifile = open(‘species.v9.0.txt’)
ofile = open(‘species.csv’, ‘w’)
for line in ifile:
  # 1. Get the list of values for this line
  # 2. Write first 3 values to the output file, separated by ‘,’
  #    (String substitution!)
```

Converting a file to another format

- Use the string substitution trick to simply transform a file.

- For example, from tab-separated (‘\t’) file to a comma-separated file.

```
ifile = open(‘species.v9.0.txt’)
ofile = open(‘species.csv’, ‘w’)
for line in ifile:
  line = line.strip(‘\n’) # Remove the newline character
  values = line.split(“\t”) # Splitting on tab character
  ofile.write(“%s,%s,%s\n” % (values[0], values[1], values[2]))
```

Converting from one type to another

- If you have a variable that you want to change its type, you can **cast** it.

- Often the source of errors in programs, where Python expects one type but is given another type.

```
a = 3
b = str(a) # Now ‘3’
c = float(a) # Now 3.0
d = int(b) # Now 3
t = tuple([‘my’, ‘list’]) # Now (‘my’, ‘list’)
l = list((1,2,3)) # Now [1,2,3]
```

Tuples

- Tuples are like lists, except you can’t change what is in them. A tuple is a frozen list.

- NO append, insert, pop, etc.

- It turns out string substitution uses tuples!

```
mytuple = (“John”, “Smith”, 54)
print “Name: %s %s, Age: %d” % mytuple
```

- Why would I use tuples in my program if I could use lists (which are more flexible)?

Math module

- The **math** module is essential in Python if you want to use math functions.
    - http://docs.python.org/library/math.html

```
import math
print math.sin(math.pi/4.) # uses radians
print math.sqrt(2)/2
print math.exp(x) # can also be written e**x
print math.factorial(5)
```

Nice to have: List comprehension

- Create a list in one line based on another iteration
  - http://docs.python.org/tutorial/datastructures.html

```
values = [1,2,3,4,5,6]
roots = []
for el in values:
  roots.append(math.sqrt(el))

# With list comprehensions everything is easy!

roots = [math.sqrt(x) for x in values]
# roots is now [1.0, 1.414, 1.732, 2.0, 2.236,
2.449]
```

Nice to have: List comprehension

- Useful when you want to create a new list that looks similar to a previous list.

```
values = [1,2,3,4,5,6]
avg = float(sum(values))/len(values)
transformed = [x-avg for x in values]
```

- Now you have two lists: **values**, and **transformed** which is going to be the same length as **values**.

Command line: how to zip files

- zip command
- zip <name of output zip file> <input files...>
- unzip command will do the opposite!