1 Python

We did a brief interactive tour of basic Python language features, including print, variables, functions, lists, list comprehensions, loops, tuples, dictionaries, import, dir, and help. The code listing can be found at the end of this document, which you can play around with in the Python interpreter.

Some free resources for Python include:


We won’t be using any complex or exotic features of Python, so it is probably not necessary to buy a Python book just for this course. If you would like one for the future, however, Learning Python and Programming Python by Mark Lutz are both excellent.

2 Runtime analysis

In this class we do not go into mathematical proofs of runtimes and other algorithmic properties, which is the main thrust of 6.046. We do, however, need a basic feel for analyzing and expressing the runtime of an algorithm.

We went through a simple example of searching a genome $G$ of length $n$ for all occurrences of a short motif $M$ of length $m$. For example, perhaps $M = \text{AAGCTA}$. A naive algorithm would test for a match in every position, as follows. (Here we just count the occurrences, but we could obviously store them in a data structure.)

```python
occ = 0
for i in range(n-m):
    match = True
    for j in range(m):
        if M[j] != G[i+j]:
            match = False
    if match:
        occ = occ +1
```

This algorithm performs $nm - m^2$ character comparisons (our "unit" of runtime for this purpose). Since $m << n$, we’ll wave our hands a bit and say that the runtime is $O(nm)$.

Suppose we now wanted to find the occurrences of all motifs $M$ of length $m$. There are $4^m$ such motifs. The simplest way would be to run the above algorithm once for each motif, giving us a runtime of $O(4^m nm)$.

Here’s a better way to solve the same problem. We can scan the genome once, and use a bookkeeping table to store the occurrences of all the motifs we see along the way.

```python
T = [0 for i in range(4 ** m)]
for i in range(n-m):
    idx = f(G[i:(i+m)])
    T[idx] = T[idx] + 1
```

Here we have assumed the existence of a function $f$ which takes a motif and returns an index into our table corresponding to that motif, in $O(m)$ time. For example, perhaps $f(\text{AAAAAA}) = 0$ and $f(\text{TTTTTT}) = 4095$. The loop of the above algorithm runs in $O(nm)$ time. But, in line 1, we had to create and initialize the big table. The total runtime is therefore $O(4^m + nm)$, which is much better than our first try.

This problem can be solved much more efficiently with hashing techniques, which we’ll learn about next week.
3 Honeybee Colony Collapse Disorder

This is not part of the course. There was some extra time, so we talked about a story that was in the news yesterday and how it relates to many of the topics we’ll be covering in 6.047.

4 Code listing for Python tour

```python
# hello world
print 'hello, world!'
print "hello, world!"

# functions & variables
def fact(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n*fact(n-1)

print fact(8)
x = fact(8)

print 'x = %d' % x

# lists & loops
lst = [1, 2, 3, 4]
print lst
print lst[2]
lst[2] = 0
lst
lst.append(5)
lst
del lst[0]
lst

lst = range(1,5)
lst

print len(lst)

for i in range(1,5):
    print i

# list comprehensions
print [x*x for x in [1,2,3,4]]
print [x*x for x in range(1,11) if x % 2 == 0]

# tuples
from math import sqrt
def csqrt(n):
    if n >= 0:
        return (sqrt(n), 0)
    else:
        return (0, sqrt(-n))
```
real, imag = csqrt(-16)
print '%d+%di' % (real.imag)

# dictionaries (hash tables)
profs = {}
profs['6.047'] = 'kellis'
profs['7.012'] = 'lander'
profs['6.047']
profs['bogus']

# import, dir, help
from math import cos
import math
dir(math)
help(math.hypot)

# matrices (lists of lists)
m = [[0 for j in range(10)] for i in range(10)]
m[3][4]
m[4][5] = 6
m