1 Another Highlight: Complex Numbers

```python
In [1]: import math
    y = (-7)**0.5
    x = 2+3j
    print(y)
    print(y.real)
    print(y.imag)
    print(abs(y))

# discrete Fourier transform of a given signal x[n]
def dft(x):
    N = len(x)
    return [1/N * sum(x[n] * math.e**(-1j*2*math.pi*n/N*k) for n in range(N)) for k in range(N)]
dft([0, 1, 0, 0])
```

2 Python Standard Library Highlights

Another reason to like Python (which we’ve not really utilized so far) is that it has a huge standard library of useful modules/functions/classes. We certainly can’t talk about it all here (see https://docs.python.org/3/library/index.html, the list is huge), but we can talk about some highlights.

2.1 Collections (import collections)

```python
In [ ]: items = ['cat', 'dog', 'ferret', 'tomato', 'chicken', 'toad']
```
```python
o = {}
for item in items:
    if item[0] not in o:
        o[item[0]] = 0
    o[item[0]] += 1

o2 = {}
for item in items:
    if item[0] not in o2:
        o2[item[0]] = []
o2[item[0]].append(item)

print(o)
print(o2)

def histogram(x):
    o = {}
    for i in x:
        o[i] = o.get(i, 0) + 1
    return o

histogram('brontosaurus')

In [ ]: entry = 'Adam', 'Hartz', 29, None, 'Hazel'

firstname = entry[0]
lastname = entry[1]
age = entry[2]
hair = entry[3]
eyes = entry[4]

firstname, lastname, age, hair, eyes = entry

In [ ]: class Env:
    def __init__(self, elts=None, parent=None):
        self.elts = elts or {}
        self.parent = parent

    def __getitem__(self, key):
        if key in self.elts:
            return self.elts[key]
        elif self.parent is not None:
            return self.parent[key]
        else:
            raise KeyError(key)

    def __setitem__(self, key, val):
```

2
self.elts[key] = val

x1 = {'cat': 'dog'}
x2 = {'coca': 'cola', 'cat': 7}
x3 = {'hello': 'goodbye'}
e = Env(x1, parent=Env(x2, parent=Env(x3)))
e['coca']

2.2 Working with iterators (import itertools)

In [ ]: def count(start, step=1):
   while True:
       yield start
       start += step

   c = count(17, 0.1)
   for i in range(5):
       print(next(c))

In [ ]: def repeat(inp, n=None):
   # yield elements from inp forever
   # for example, cycle('ABCD') => 'A' 'B' 'C' 'D' 'A' 'B' 'C' 'D' ...
   pass

   c = repeat('cat.', 20)
   for i in c:
       print(i)

   c = repeat('dog.')
   for i in range(101):
       print(next(c))

In [ ]: def cycle(inp):
   # yield elements from inp forever
   # for example, cycle('ABCD') => 'A' 'B' 'C' 'D' 'A' 'B' 'C' 'D' ...
   pass

   c = cycle('hello')
   for i in range(21):
       print(next(c))

In [ ]: def a():
   yield '6.009'

   def b():
       yield 'cat'
       yield 'dog'
       yield 'tomato'
def chain(*args):
    # yield from each iterator in order
    pass

c = chain(a(), b(), ['hello', 'there'])
for i in c:
    print(i)

2.3 Other Highlights

- mathy things: math, cmath, random, statistics
- rational numbers: fractions
- tools for working with functions: functools
- implementations of built-in operations as functions: operator
- tools for interacting with operating system: os, sys
- tools for dealing with errors/reporting: traceback, logging
- tools for creating/interacting with Internet protocols/etc
  - email, smtplib, etc
  - http.server, urllib.request, etc

These modules can be super useful, but aren't really worth talking about here (their contents are kind of boring).

2.4 External Packages

Outside of the standard library, there are a wealth of other useful packages!
Examples:

- sympy for symbolic algebra
- numpy for numeric computation (fast operations on large multi-dim arrays+matrices)
- matplotlib for generating plots
- nltk for natural language processing
- etc, etc, etc