61A Lecture 30

Monday, April 13
Announcements

• Homework 8 due Wednesday 4/15 @ 11:59pm (small)
• Project 4 due Thursday 4/23 @ 11:59pm (BIG!)
  ▪ Project/Homework party Tuesday 4/14 5pm–6:30pm in 2050 VLSB
  ▪ Early point #1: Questions 1–12 submitted (correctly) by Friday 4/17 @ 11:59pm
  ▪ Early point #2: All questions (including Extra Credit) by Wednesday 4/22 @ 11:59pm
• If you want the first early submission point, you need to:
  ▪ Pass the tests for the designated questions
  ▪ Run `python3 ok --submit`
  ▪ Log on to [http://ok.cs61a.org](http://ok.cs61a.org) and create a group with your partner
Ray Tracing

A technique for displaying a 3D scene on a 2D screen by tracing a path through every pixel.

**The Scene:**
(Demo)

- Light
- Camera
- Sphere at origin

**Dramatization:**

- Light
- Camera
- Sphere
- Distance to Sphere

(Demo)
Information Hiding
Attributes for Internal Use

An attribute name that starts with one underscore is not meant to be referenced externally.

```python
class FibIter:
    """An iterator over Fibonacci numbers."""
    def __init__(self):
        self._next = 0
        self._addend = 1

    def __next__(self):
        result = self._next
        self._addend, self._next = self._next, self._addend + self._next
        return result

>>> fibs = FibIter()
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

This naming convention is not enforced, but is typically respected

A programmer who designs and maintains a public module may change internal-use names

Starting a name with `two underscores` enforces restricted access from outside the class
Names in Local Scope

A name bound in a local frame is not accessible to other environments, except those that extend the frame.

```python
def fib_generator():
    """A generator function for Fibonacci numbers."

>>> fibs = fib_generator()
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

"""
yield 0
previous, current = 0, 1
while True:
    yield current
    previous, current = current, previous + current
```
Singleton Objects

A singleton class is a class that only ever has one instance

NoneType, the class of None, is a singleton class; None is its only instance

For user-defined singletons, some programmers re-bind the class name to the instance

```python
class empty_iterator:
    """An iterator over no values."""
    def __next__(self):
        raise StopIteration
empty_iterator = empty_iterator()
```
Streams
Streams are Lazy Linked Lists

A stream is a linked list, but the rest of the list is computed on demand

\[ \text{Link}( \underline{\text{First element can be anything}}, \underline{\text{Second element is a Link instance or Link.empty}}) \]

\[ \text{Stream}( \underline{\text{First element can be anything}}, \underline{\text{Second element is a zero-argument function that returns a Stream or Stream.empty}}) \]

Once created, Streams and Links can be used interchangeably using \textit{first} and \textit{rest} methods

(Demo)
Integer Stream

An integer stream is a stream of consecutive integers

An integer stream starting at first is constructed from first and a function compute_rest that returns the integer stream starting at first+1

def integer_stream(first=1):
    """Return a stream of consecutive integers, starting with first.

>>> s = integer_stream(3)
>>> s.first
3
>>> s.rest.first
4
"""

def compute_rest():
    return integer_stream(first+1)
return Stream(first, compute_rest)

(Demo)
Cross the Stream

Which definition will produce which row of elements after executing \texttt{s = f()}?

<table>
<thead>
<tr>
<th></th>
<th>s.first</th>
<th>s.rest.first</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{def f(x=1):} \texttt{return Stream([x], lambda: f([x]))}</td>
<td>[1]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>\texttt{def f(x=[1]):} \texttt{return Stream(x, lambda: f(x+[1]))}</td>
<td>[1, 1]</td>
<td>[1, 1]</td>
</tr>
<tr>
<td>\texttt{def f(x=1):} \texttt{s = Stream([x], lambda: s)} \texttt{return s}</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>\texttt{def f(x=[]):} \texttt{x.append(1)} \texttt{return Stream(x, lambda: f(x))}</td>
<td>[1]</td>
<td>[[1]]</td>
</tr>
</tbody>
</table>
Stream Processing

(Demo)
Stream Implementation
A stream is a linked list with an explicit first element and a rest-of-the-list that is computed lazily

```python
class Stream:
    """A lazily computed linked list."""
    class empty:
        def __repr__(self):
            return 'Stream.empty'
    empty = empty()

def __init__(self, first, compute_rest=lambda: Stream.empty):
    assert callable(compute_rest), 'compute_rest must be callable.'
    self.first = first
    self._compute_rest = compute_rest

@property
def rest(self):
    """Return the rest of the stream, computing it if necessary."""
    if self._compute_rest is not None:
        self._rest = self._compute_rest()
    self._compute_rest = None
    return self._rest
```
Higher-Order Functions on Streams
Mapping a Function over a Stream

Mapping a function over a stream applies a function only to the first element right away; the rest is computed lazily.

```python
def map_stream(fn, s):
    """Map a function fn over the elements of a stream s."""
    if s is Stream.empty:
        return s
    def compute_rest():
        return map_stream(fn, s.rest)
    return Stream(fn(s.first), compute_rest)

>>> s = integer_stream(3)
... s
Stream(3, <...>)
>>> m = map_stream(lambda x: x*x, s)
>>> first_k(m, 5)
[9, 16, 25, 36, 49]
```
Filtering a Stream

When filtering a stream, processing continues until an element is kept in the output.

def filter_stream(fn, s):
    """Filter stream s with predicate function fn."""
    if s is Stream.empty:
        return s
    def compute_rest():
        return filter_stream(fn, s.rest)
    if fn(s.first):
        return Stream(s.first, compute_rest)
    else:
        return compute_rest()

Actually compute the rest
A Stream of Primes

The stream of integers not divisible by any \( k \leq n \) is:
- The stream of integers not divisible by any \( k < n \)
- Filtered to remove any element divisible by \( n \)

This recurrence is called the Sieve of Eratosthenes

\[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\]