

61A Lecture 30

Monday, November 18

Information Hiding

Announcements

- Homework 9 due Tuesday 11/19 @ 11:59pm
- Project 4 due Thursday 11/21 @ 11:59pm
- Extra reader office hours in 405 Soda this week
 - Monday: 5pm-6:30pm
 - Tuesday: 6pm-7:30pm
 - Wednesday: 5:30pm-7pm
 - Thursday: 5:30pm-7pm

Attributes for Internal Use

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

```
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()
>>> [next(fibs) for _ in range(10)]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

"Please don't reference these directly. They may change."

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.

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

    >>> fibs = fib_generator()
    >>> [next(fibs) for _ in range(10)]
    [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
```

There is no way to access values bound to "previous" and "current" externally

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.

```
class empty_iterator:
    """An iterator over no values."""
    def __next__(self):
        raise StopIteration
empty_iterator = empty_iterator()
```

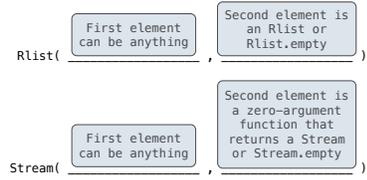
The instance

The class

Streams

Streams are Lazy Recursive Lists

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



Once created, Streams and Rlists can be used interchangeably using `first` and `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)

Stream Processing

(Demo)

Stream Implementation

Stream Implementation

A stream is a recursive list with an explicit first element and a rest-of-the-list that is computed lazily.

```
class Stream:  
    """A lazily computed recursive 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.

```
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)
```

This body is not executed until compute_rest is called

Not called yet

```
>>> 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

(Demo)