CIS 343 - Structure of Programming Languages

Nathan Bowman

Based on slides by Ira Woodring

Lazy Evaluation

When we used Python version of `map`, it produced "strange" results

```python
vals = [1, 2, 3]
print(map(lambda x : 2*x, vals))
# <map at 0x7f35490828b0>
```

The map seems to work correctly, because it can turn into list with correct entries

```
vals = [1, 2, 3]
map(lambda x : 2*x, vals)
# <map at 0x7f35490828b0>

list(map(lambda x : 2*x, vals))
# [2, 4, 6]
```

This is an example of **lazy evaluation**

Python is not computing entries of map until required to do so

`map` does not hold entries of resulting list, but instead keeps track of what it would need to compute them

```
for entry in map(lambda x : 2*x, vals):
    print(entry)

# 2
# 4
# 6
```

Each `entry` computed only when needed at start of loop iteration

Eager evaluation of Fibonnaci numbers

```python
def fib_eager():
    f0 = 0
    f1 = 1
    fibs_list = []
    for i in range(10):
        fibs_list.append(f1)
        temp = f0
        f0 = f1
        f1 = f1 + temp
    return fibs_list
```

Lazy evaluation of Fibonnaci numbers

```python
def fib_lazy():
    f0 = 0
    f1 = 1
    for i in range(10):
        yield f1
        temp = f0
        f0 = f1
        f1 = f1 + temp
```

```
fib_eager()
# [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

fib_lazy()
# <generator object fib_lazy at 0x7f3548791350>
```

```python
for v in fib_eager():
    print(v)

for v in fib_lazy():
    print(v)
```

Both print

```
1
2
3
5
8
13
21
34
55
```

Why bother?

Note that in lazy evaluation example, did not need to save entire list

If loop is stopped early, such as after getting number of certain size, lazy evaluation avoids computing remaining numbers

This could also have been accomplished with `while` loop, but generator can be nice abstraction in many cases, such as with `map`

Just a reminder of our original motivation

```
for entry in map(lambda x : 2*x, vals):
    print(entry)
```

Showed this example in Python because Scheme does not use lazy evaluation

To learn more about lazy evaluation in Python, read up on "generators"

Haskell is one example of a pure functional language with lazy evaluation

Benefits of lazy evaluation include

- ability to define infinite data structures
- avoiding needless computation
- shrinking memory footprint in some cases

However, can be difficult for language designer to implement, especially in the presence of I/O and exception handling