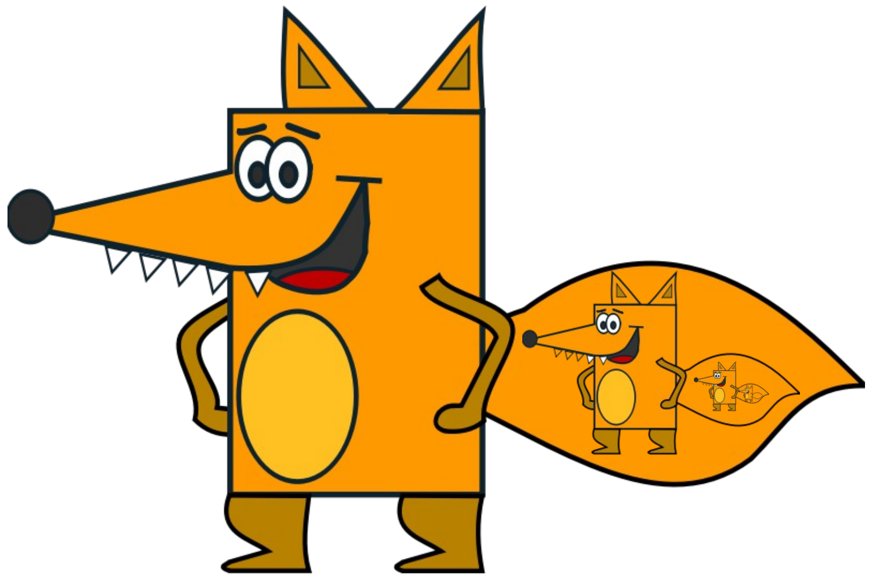
CS240
Fall 2014

Mike Lam, Professor



Tail Recursion

Academic Calendar

- October 23: Last date to drop with a “W”
  - Talk to me if you are considering this
- October 27: Spring 2015 registration begins
  - Open enrollment begins Nov 6
  - Consider registering for **CS 452** (w/ Fox)
    - “Design and Analysis of Algorithms”
    - Basically picks up where we will stop in CS 240
    - More algorithmic analysis & intro to algorithm *design*
    - Counts as a CS elective!
    - Prerequisites: CS 228 and CS 240

Tail Recursion

- A recursive call occurs as the final operation in a function
  - And the result of recursive call is immediately returned
- No need for new stack frame
  - Just re-use the old one
  - Set parameters and jump back to function entry
- Some languages do this optimization automatically
  - Python does not, for philosophical reasons
    - http://neopythonic.blogspot.co.uk/2009/04/tail-recursion-elimination.html
- Regardless, you can **always** manually convert a tail-recursive procedure to an iterative one

Example

- Consider this factorial implementation:

```
def fact_helper(x, acc):
    if x <= 1:
        return acc
    else:
        return fact_helper(x-1, x*acc)

def fact(x):
    return fact_helper(x, 1)
```

```
def fact(x):
    if x <= 1:
        return 1
    else:
        return x * fact(x-1)
```

(old version)

Example

- Consider this factorial implementation:

```
def fact_helper(x, acc):
    if x <= 1:
        return acc
    else:
        return fact_helper(x-1, x*acc)

def fact(x):
    return fact_helper(x, 1)
```

```
      fact(4)
  fact_helper(4, 1)
  fact_helper(3, 4)
 fact_helper(2, 12)
 fact_helper(1, 24)
```

Example

- Consider this factorial implementation:

```
def fact_helper(x, acc):
    if x <= 1:
        return acc
    else:
        return fact_helper(x-1, x*acc)

def fact(x):
    return fact_helper(x, 1)
```

```
   fact(4)
fact_helper(4, 1)
fact_helper(3, 4)
fact_helper(2, 12)
fact_helper(1, 24)
```

```
def fact(x):
    acc = 1
    while x > 1:
        acc = x*acc
        x = x-1
    return acc
```

Eliminating Tail Recursion

- Helper function parameter **=>** local variable
- Recursive case parameters **=>** local variable assignment
- Base case check **=>** loop conditional
- Base case return **=>** variable initialization

```
def fact(x):
    return fact_helper(x, 1)

def fact_helper(x, acc):
    if x <= 1:
        return acc
    else:
        return fact_helper(x-1, x*acc)
```

```
def fact(x):
    acc = 1
    while x > 1:
        acc = x*acc
        x = x-1
    return acc
```

Example

- Consider binary search:

```
def search(array, item):
    return helper(array, item, 0, len(array))

def helper(array, item, left, right):
    mid = (right-left)//2 + left
    if array[mid] > item:
        return helper(array, item, left, mid)
    elif array[mid] < item:
        return helper(array, item, mid+1, right)
    else:
        return left < len(array) and array[left] == item
```

Example

- Iterative version of binary search:

```
def search(array, item):
    left = 0
    right = len(array)
    while right > left+1:
        mid = (right-left)//2 + left
        if array[mid] > item:
            right = mid
        elif array[mid] < item:
            left = mid+1
        else:
            left = mid
            right = mid+1
    return left < len(array) and \
        array[left] == item
```

Exercise

- Eliminate tail recursion:

```
def foo(v):

    return _foo(v, [])

def _foo(v, w):
    if len(v) == 0:
        return w
    else:
        w.append(v[0] * v[0])
        return _foo(v[1:], w)
```

Exercise

- Eliminate tail recursion:

```
def foo(v):

    return _foo(v, [])

def _foo(v, w):
    if len(v) == 0:
        return w
    else:
        w.append(v[0] * v[0])
        return _foo(v[1:], w)
```

Straightforward conversion:

```
def bar(v):
    w = []
    while len(v) > 0:
        w.append(v[0] * v[0])
        v = v[1:]
    return w
```

Exercise

- Eliminate tail recursion:

```
def foo(v):

    return _foo(v, [])

def _foo(v, w):
    if len(v) == 0:
        return w
    else:
        w.append(v[0] * v[0])
        return _foo(v[1:], w)
```

Straightforward conversion:

```
def bar(v):
    w = []
    while len(v) > 0:
        w.append(v[0] * v[0])
        v = v[1:]
    return w
```

More efficient version:

```
def baz(v):
    w = []
    i = 0
    while i < len(v):
        w.append(v[i] * v[i])
        i += 1
    return w
```