Discussion 9:

Interpreters & Tail Calls

Caroline Lemieux (clemieux@berkeley.edu)

April 4th, 2019

Calculator and Evaluation

The *Calculator* Language

Scheme-like

4 operators: **/ * + -**

All operands are **numbers**

(Scheme lists)

Our Interpreter

Implemented in Python

'/' '*' '+' '-'

Numbers are numbers!

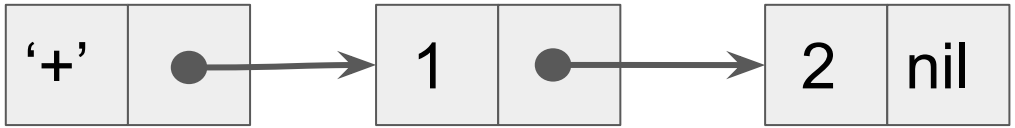
`Pair` class representation →

The `Pair` Class (introduction pg. 1-2)

Calculator expression: **(+ 1 2)**

Scheme list representation: **(cons ‘+’ (cons 1 (cons 2 nil)))**

Representation w/ `Pair`: **Pair(‘+’, Pair(1, Pair(2)))**

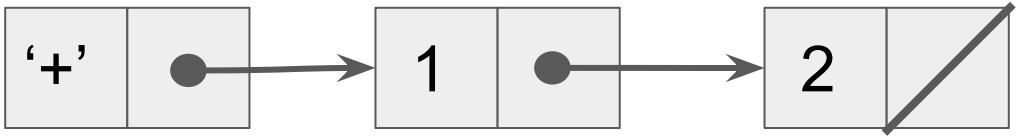


The `Pair` Class Continued (introduction pg. 1-2)

Looks like `Link`!

```
p.first   p.second
l.first   l.rest
```

```
Link('+', Link(1, Link(2)))
```



Let’s practice!

Attendance

links.cs61a.org/caro-disc

magic word: **watermelon**

Tail Recursion

A silly example

```
def silly(positive, negative):
    if positive == 0:
        return negative
    else:
        return silly(positive - 1, negative - 1)

print(silly(4,0))
```

At this link

- Why are we keeping all these frames around???
- Tail optimization: just keep that last frame

When can’t we keep *just* the last frame

```
def silly(positive):
    if positive == 0:
        return 0
    else:
        return silly(positive - 1) - 1

print(silly(4))
```

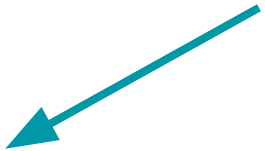
At this link

When the other frames retain information about work we still have to do

Terminology Review

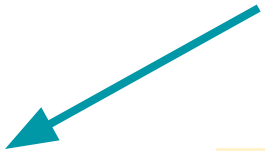
A procedure is **tail recursive** iff

all recursive calls occur are **tail calls**.



A function call is a **tail call**

if it is in a **tail context.**

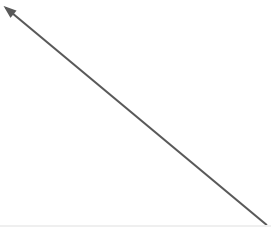


A **tail context** is: in python, the “outer” expression in a ***return*** statement

Tail Recursion or Not?

```
def silly(positive, negative):
    if positive == 0:
        return negative
    else:
        return silly(positive - 1, negative - 1)

print(silly(4,0))
```

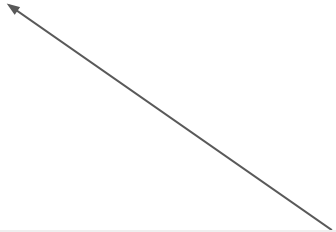


Yes! Call to silly is the outer return expression

Tail Recursion or Not?

```
def silly(positive):
    if positive == 0:
        return 0
    else:
        return -1 + silly(positive - 1)

print(silly(4))
```

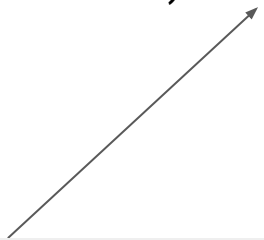


No! We need to add this still

Tail Recursion or Not?

```
def silly(positive):
    if positive == 0:
        return 0
    else:
        return silly(positive - 1) - 1

print(silly(4))
```



No! We need to do all these -1 s

Tail Recursion or Not?

```
def silly(positive):
    if positive > 0:
        return silly(positive - 1)
    else:
        return 0

print(silly(4))
```

Yes! This is still the outermost expression

Tail Recursion or Not?

```
def silly(positive, negative):
    if positive == 0:
        return negative
    elif silly(positive - 1, negative) > 0:
        return silly(positive - 1, negative - 1)
    else:
        return silly(positive - 1, negative - 1)

print(silly(4,0))
```

Not tail position!

Tail position :)

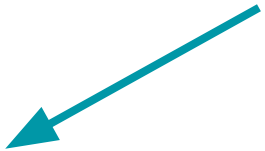
Tail position :)

Conclusion: not tail recursion!

Terminology Review

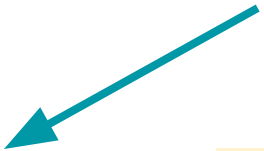
A procedure is **tail recursive** iff

all recursive calls occur are **tail calls**.



A function call is a **tail call**

if it is in a **tail context.**



A **tail context** is, in scheme, roughly **“nothing else left around it to evaluate”**

(after: some memorizeable rules)

Back to our silly example.... in Scheme

```
(define (silly positive negative)
    (if (= positive 0)
        negative
        (silly (- positive 1) (- negative 1))
     )
)
```

Evaluating `(silly 2 0)`

```
(define (silly positive negative)
    (if (= positive 0)
        negative
        (silly (- positive 1) (- negative 1))
     )
)
```

Replacing the function with its body….

```
(silly 2 0)
```

Evaluating (silly 2 0)

```
(define (silly positive negative)
    (if (= positive 0)
        negative
        (silly (- positive 1) (- negative 1))
     )
)
```

If predicate is false, replace with false expression

```
(if (= 2 0)
    0
    (silly (- 2 1) (- 0 1))
```

Evaluating (silly 2 0)

```
(define (silly positive negative)
    (if (= positive 0)
        negative
        (silly (- positive 1) (- negative 1))
     )
)
```

Tail recursion!

All that's left is a call to silly!

```
(silly (- 2 1) (- 0 1))
```

Back to our non-tail silly example.... in Scheme

```
(define (silly positive)
    (if (= positive 0)
        0
        (- (silly (- positive 1)) 1)
     )
)
```

Back to our non-tail silly example.... in Scheme

```
(define (silly positive)
    (if (= positive 0)
        0
        (- (silly (- positive 1)) 1)
     )
)
```

Replacing the function with its body….

```
(silly 2)
```

Back to our non-tail silly example.... in Scheme

```
(define (silly positive)
   (if (= positive 0)
      0
      (- (silly (- positive 1)) 1)
    )
)
```

If predicate is false, replace with false expression

```
(if (= 2 0)
   0
   (- (silly (- 2 1)) 1)
)
```

Back to our non-tail silly example.... in Scheme

```
(define (silly positive)
    (if (= positive 0)
        0
        (- (silly (- positive 1)) 1)
     )
)
```

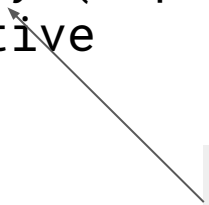
Not tail recursion! :(

Need to call silly **then** evaluate the minus

```
(- (silly (- 2 1)) 1)
```

Pop quiz! Tail recursion or not?

```
(define (silly positive negtaive)
    (if (> positive 0)
        (silly (- positive 1) (- negative 1))
        negative
     )
)
```



Still tail recursion!

Pop quiz! Tail recursion or not?

```
(define (silly positive negtaive)
    (if (> positive 0)
        (silly (- positive 1) (- negative 1))
        negative
     )
)
```

Still tail recursion!

```
def silly(positive, negative):
    if positive > 0:
        return silly(positive - 1, negative -1)
    else:
        return 0
```

Let’s go through 3.1

Let’s go through 4.3 (a)

Try 3.2