Lecture 3: Boolean and Conditionals

Kaleel Mahmood
Department of Computer Science and Engineering
University of Connecticut

Previously on the hit TV show CSE: 1729…

Three Exciting Developments

1. The spin off series "*The Office Hours*" is happening 5 days a week!!!

	Monday	Tuesday	Wednesday	Thursday	Friday
9	Michael 9-11		Michael 9-11		
10				Jacob 10-12	Andy 10-12
11				Akul 11-2	
12				Samantha 12-2	Akul 12-2
1	Marquis, Kaustubh 1-4		Andy 1-3	Adrienne 1-3	Michael 2-3
2					
3			Adrienne 3-5	Marquis 3-5	Kaustubh 3-4
4	Samantha, Rachel 4-6		Rachel 4-6		
5					
6	Matthew, Jacob 6-8	Matthew 6-8			
7					
8					

Previously on the hit TV show CSE: 1729…

Three Exciting Developments

2. We learned how to define variable and functions in Scheme….

```
(define <variable> <value>)
```

```
(define (<function-name> <argument-var>)
        <body>)
```

And we met the evil version of Dr. Racket…the Scheme Questioner!

Previously on the hit TV show CSE: 1729…

Three Exciting Developments

3. We talked about Global/Method environments and why there can be three spidermen in the movies

```
1  (define spiderman 3)
2
3
4  (define (kaleelFunction x)
5    (define spiderman 6)
6    (+ spiderman x)
7    )
8
9  (kaleelFunction 2)
10
11 spiderman
12
```

GLOBAL Environment

spiderman = 3

Method Environment

x =2

spiderman = 6

return 8

Lecture Overview

Booleans and If Statements

Conditionals

Previous Questions

BOOLEAN VALUES IN SCHEME

- Along with numbers, which we've already explored, SCHEME can maintain Boolean values (true/false). These are denoted `#t` and `#f`.
- As with numbers, they evaluate to themselves.
- Scheme has a full set of logical functions:
  - `(and x y)` , true exactly when both x and y are true,
  - `(or x y)`, true when either x or y is true (or both),
  - `(not x)`, returns of negation of its argument.

BOOLEAN VALUES IN SCHEME

```
> (define a #t)
```

[redacted]

CONDITIONALS VIA IF

- Simple conditionals are implemented by the if special form. The syntax is

```
(if <pred>
    <then-clause>
    <else-clause>)
```

- The expression `<pred>` is evaluated.
- If it evaluates to #t, `<then-clause>` is evaluated, and its value is returned. Otherwise, `<else-clause>` is evaluated, and its value is returned.
- Note that *only one of the two* expressions `<then-clause>` and `<else-clause>` is evaluated. (This requires if to be a special form.)

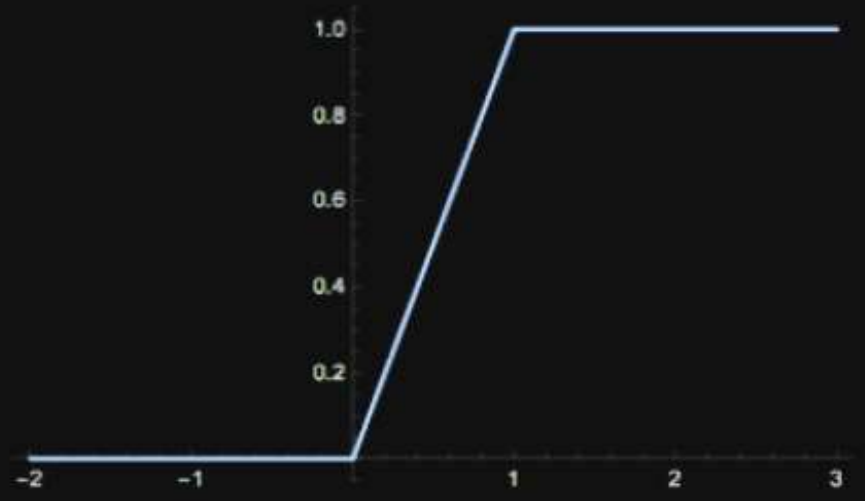
Some if examples

```
> (if #t 3 4)
3
```

NESTED IFS: LEGAL... BUT... HARD TO READ

- Consider defining the function:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$



- Leads to the Scheme code:

```
(define (f x)
        (if (< x 0)
                0
                (if (> x 1)
                        1
                        x)))
```

A "nested" if

AN ALTERNATIVE: CONDITIONALS VIA THE COND SPECIAL FORM

- The SCHEME *conditional* special form:

```
(cond (<guard1> <expr1>)
      (<guard2> <expr2>)
       ...
      (<guardn> <exprn>))
```

- Evaluates the expressions <guard1>, <guard2>, ... in this order until it finds one, say <guardk>, that evaluates to #t. Then returns the value obtained by evaluating <exprk>.
- NOTE: *Only one* of the <exprk> is evaluated. This is important, as we shall see in future lectures.
- We do not define the result if none of the guards evaluate to #t.

A Conditional Example

Goal: Write the following code, given x

If x=1, then z = 1

if x>1 then z = 100

Lastly add z and x together.

```
(define x 1)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

Question: What happens if x=0?

Let’s check some test cases…

```
(define x 1)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
2
>
```

```
(define x 2)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
102
>
```

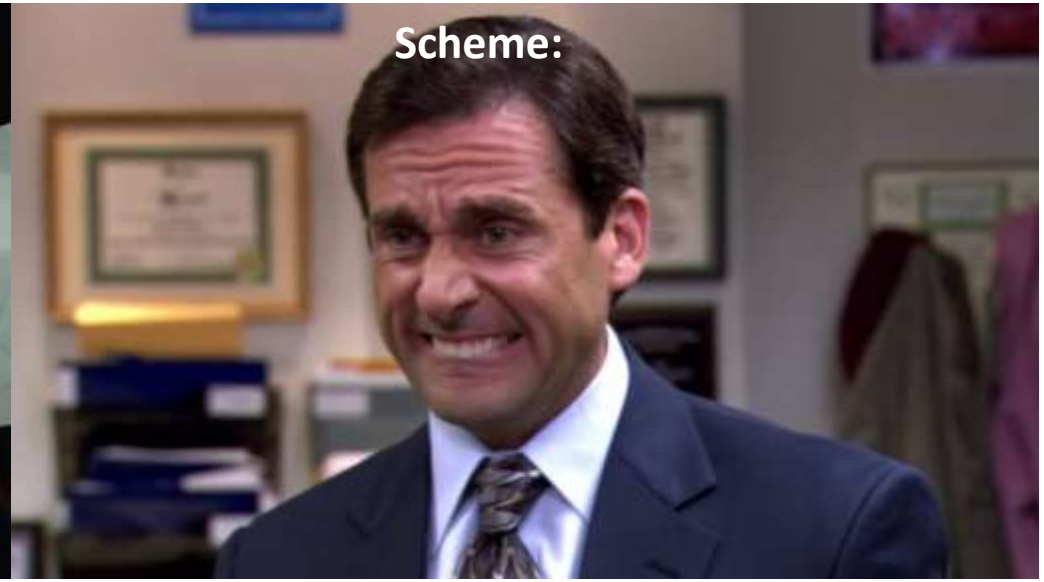
Let’s check some test cases…

```
(define x 0)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
+: contract violation
  expected: number?
  given: #<void>
```



Fixing our broken code….

```
(define x 0)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
(define x 0)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)
                (#t 0)))

(+ z x)
```

Food for thought: Do the same design flaws exist in Python?

```
x = 0

def GetZValue(x):
    if x == 1:
        return 1
    elif x>1:
        return 100

solution = x + GetZValue(x)
print(solution)
```

Is this fix also OK?

```
(define x 0)

(define z (cond ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
(define x 1)

(define z (cond (#t 0)
                ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
1
>
```

Goal: Write the following code, given x

If x=1, then z = 1

if x>1 then z = 100

Lastly add z and x together.

Scheme vs Python…

Scheme

```scheme
(define x 1)

(define z (cond (#t 0)
                ((= x 1) 1)
                ((> x 1) 100)))

(+ z x)
```

-Lets you compile wrong logic.
-Lets you run the wrong logic and won't throw error.

Python

```python
x = 0

def GetZValue(x):
    else:
        return 0
    if x == 1:
        return 1
    elif x>1:
        return 100

solution = x + GetZValue(x)
print(solution)
```

+Doesn't let you write wrong logic statements. Will immediately complain about syntax.

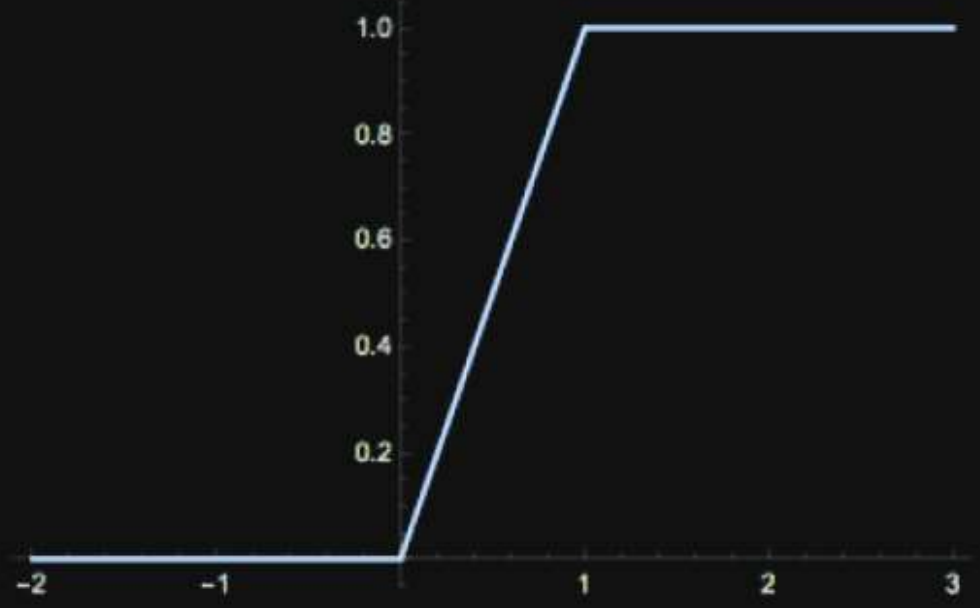
Why do I keep bringing up Scheme vs Python?

- Answer: Because the professor hates Scheme….?
- NO!
- When learning a language it is important to analyze the strengths and weaknesses.
- We don’t want to just *LEARN* Scheme, we want to understand when it is a good tool to use!

THE PIECEWISE LINEAR EXAMPLE AGAIN...WITH COND

- Consider defining the function:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$



- Using cond, this leads to the Scheme code:

```
(define (piecewise x)
  (cond ((< x 0) 0)
        ((< x 1) x)
        (#t  1)))
```

Why do these "guards" differ from those above?

THE ELSE GUARD IN COND

- You can use else as a guard that always evaluates to true. (You might call this "syntactic sugar" for #t.)

```
> (cond (#f 1)
        (else 2))
2
```

- Equivalently, you could use #t.

```
> (cond (#f 1)
        (#t 2))
2
```

Answering Last Lectures Questions…

- How can I create other types of variables in Scheme?

Scheme uses implicit variable declaration (the type of variable is assumed by what value you are storing in it).

```
(define x 1.55)
x

(define y "hello there")

y
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
1.55
"hello there"
>
```

Side note: Scheme is case insensitive for variable names…e.g. X is the same as x

What/who are keywords?

- Keywords – Reserved words in the language that have a special meaning or function and cannot be used as new function or variable names.
- For example “True” is a keyword in Python and hence you cannot use “True” to name a variable:

```
1
2   True = 5
3
```

What are the keywords in Scheme?

- The short answer: They are hard to find.
- The shorter answer: This will happen if you accidently use a keyword as a variable:

```
(define truncate 1.55)
truncate
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: R5RS; memory limit: 128 MB.
define-values: assignment disallowed;
 cannot re-define a constant
  constant: truncate
  in module:top-level
>
```

The long answer: You can find it here:

https://groups.csail.mit.edu/mac/ftpdir/scheme-7.4/doc-html/scheme_2.html#SEC27

Other Questions You May Have…

1. Start experimenting yourself.
2. For some questions I may say we'll try and answer it later in the course.

Figure Sources

- https://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Eo_circle_green_white_checkmark.svg/1200px-Eo_circle_green_white_checkmark.svg.png
- https://cdn.kapwing.com/collections/Panik-Kalm-Panik-bhebh.jpg
- https://media.istockphoto.com/photos/bear-raising-paw-picture-id149070679?k=20&m=149070679&s=170667a&w=0&h=UQx69eH2LHfO13qMw3acZDz4j_hEbQeagNlIgj5934c=
- https://www.looper.com/img/gallery/the-offices-michael-scott-was-almost-a-murderer/intro-1591207215.jpg
- https://upload.wikimedia.org/wikipedia/commons/5/5c/Mark_Zuckerberg_-_Move_Fast_and_Break_Things.jpg
- https://media.istockphoto.com/photos/angry-female-brunette-punching-computer-screen-with-large-hole-on-picture-id504882250?k=20&m=504882250&s=170667a&w=0&h=X74o-smWHNMa785h6VLfm-Lcj7pgR_E0w5De10U3SZU=
- https://roost.nbcuni.com/bin/viewasset.html/content/dam/Peacock/Campaign/landingpages/library/theoffice/mainpage/office-social-min.png/_jcr_content/renditions/original
- https://i.ytimg.com/vi/wnWjxG1fytM/maxresdefault.jpg
- Professor Greg Johnson’s lecture slides.