



Welcome to Lecture 6: Functions as Data + Lambdas

Class will start at 10:10.

In the meantime, we will go around. Tell me your name, where you're from, & favorite mythical creature.

Today's Topics

- Announcements
- Review
- List Scope
- Mutability vs Immutability
- Functions as input
- Lambdas
- Call, run, and rings in Snap!

Announcements

- Victoria's OH is 6 to 7PM on Mon + Wed (Hybrid: online and in soda-777)
- Victoria's SUPPORT OH 7 to 8PM on Mon (Hybrid: online and in soda-777)
- Computers: You can always use the computers in SDH-200, you will sign into your account here:
<https://acropolis.cs.berkeley.edu/~account/webacct/>
- We are removing "duplicates" for Lab 4: Lists + Loops
 - So don't worry if you didn't get credit!

Review from Last Lecture

- Mutability vs Immutability
 - Mutability: Object can be changed after created
 - Immutability: Object CANNOT be changed after created
 - Lists are one of few data types that are mutable in Snap!
 - Only these functions can mutate a list:

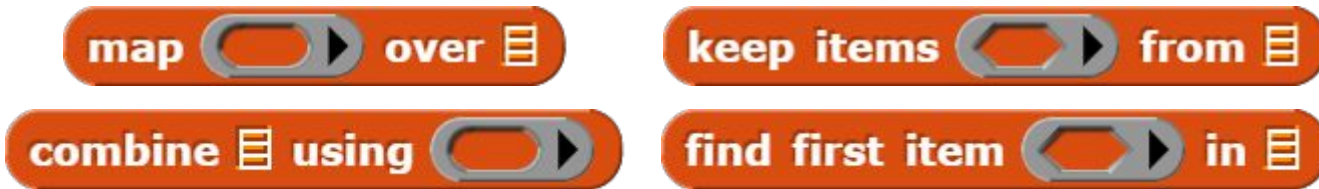


Review from Last Lecture

- Higher Order Functions (HOFs)
 - Definition:

Review from Last Lecture

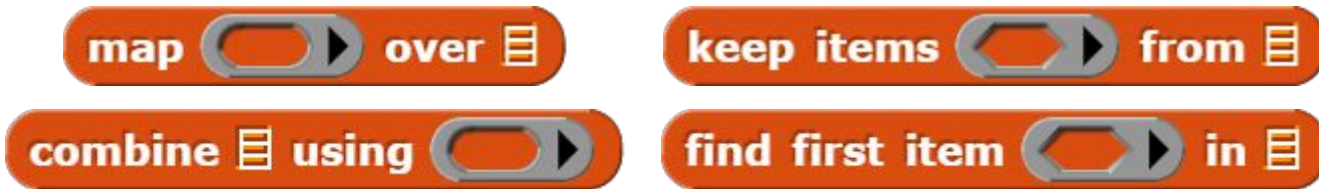
- Higher Order Functions (HOFs)
 - Definition: A function whose input is a function
 - Built in HOFs in Snap!:



- But, we can make our own!
- Do the Built-in HOFs, return new values/lists or modify the input list?

Review from Last Lecture

- Higher Order Functions (HOFs)
 - Definition: A function whose input is a function
 - Built in HOFs in Snap!:



- But, we can make our own!
- Do the Built-in HOFs, return new values/lists or modify the input list?
 - Return new values / list!

Review from Last Lecture

map



over



Performs a function on EACH item in the input list. Will return same size list.

keep items



from



Filters out items if the conditions evaluates to true. Will return less than or equal to input list

combine



using



Reduces input list based on function by applying function on all items. Didactic function! Unintended behavior if less or more than 2 inputs

find first item



in



Finds the first item from the list where the condition evaluates to true.

Review from Last Lecture



Review from Last Lecture

```
map [is a number?] over list a 2 3 4
```

A list view showing the results of the map function. It contains four items, each with a number and a boolean value:

- 1 false
- 2 true
- 3 true
- 4 true

At the bottom, it shows a plus sign icon and the text "length: 4".

Review from Last Lecture

keep items

is a number ▼ ?

from

list ◀▶

Review from Last Lecture



Review from Last Lecture

find first item

is a number ?

in

list

a

2

3

4

Review from Last Lecture

find first item

is a number ?

in

list

a

2

3

4

2

Review from Last Lecture

script variables

a

set a to map is a number ? over list a 2 3 4

report combine a using and

Review from Last Lecture

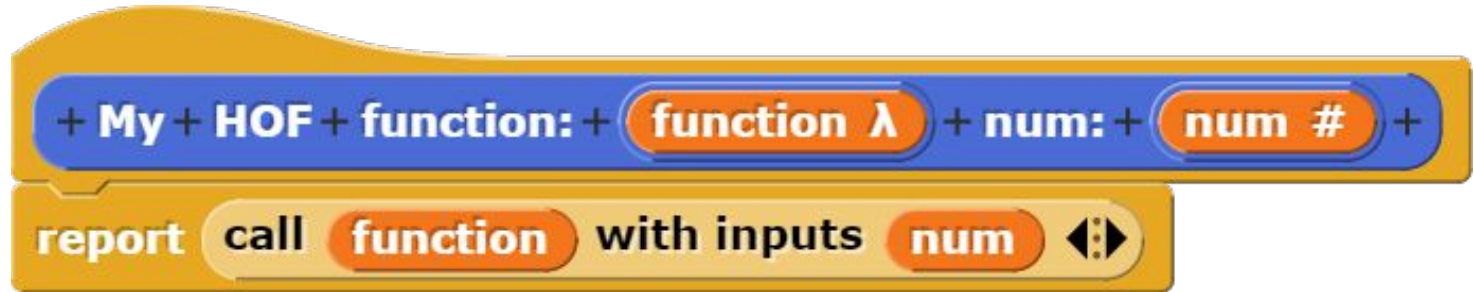
The image shows three Scratch code blocks:

- script variables** block: A grey block with a variable 'a' in a circle and a right-pointing arrow.
- set a to map** block: An orange block with 'set a to map' text, a green 'is a number?' block, and 'over list' text followed by a list containing 'a', '2', '3', and '4'.
- report combine** block: A yellow block with 'report combine a using' text and a green 'and' block.

A speech bubble on the right contains the text **false**.

Functions as Data

- We can make our own HOFs
- The input to our function will be a function!
- Example:



- Call function: Invokes ANY function with inputs dynamically (i.e. we specify function and inputs at runtime)
- Must call function manually with “call” or “run” function

Functions as Data

- Why is “call” necessary?



Functions as Data

- Why is “call” necessary?



- Allows us to invoke the function
- Allows us to pass in inputs to function

Functions as Data

- Call Example

```
+ sum + from + num1: + num1 + to + num2: + num2 +  
script variables a ▶  
set a ▼ to 0  
for i = num1 to num2  
  change a ▼ by i  
report a
```

```
call sum from num1: [ ] to num2: [ ] with inputs 1 4 [ ] 10
```

Build a Drawing HOF!

Draw Normal Square 100



Draw Dashed Square 100



Draw Wiggly Square 100



+Draw + Normal + Square + length +

repeat 4

Draw Line length

turn 90 degrees

+Draw + Dashed + Square + length +

repeat 4

Draw Dashed Line length

turn 90 degrees

+Draw + Wiggly + Square + length +

repeat 4

Draw Wiggly Line length

turn 90 degrees

- Not very efficient
- Tedious if repeated
- Let's generalize [Draw Square](#) into a HoF.

+Draw + Square + length +

pen down

repeat 4

move length steps


turn 90 degrees

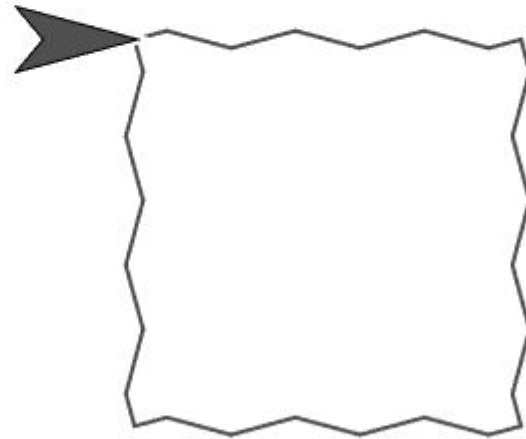
pen up

Build a Drawing HOF!

Objective: Generalize square to draw any line type!

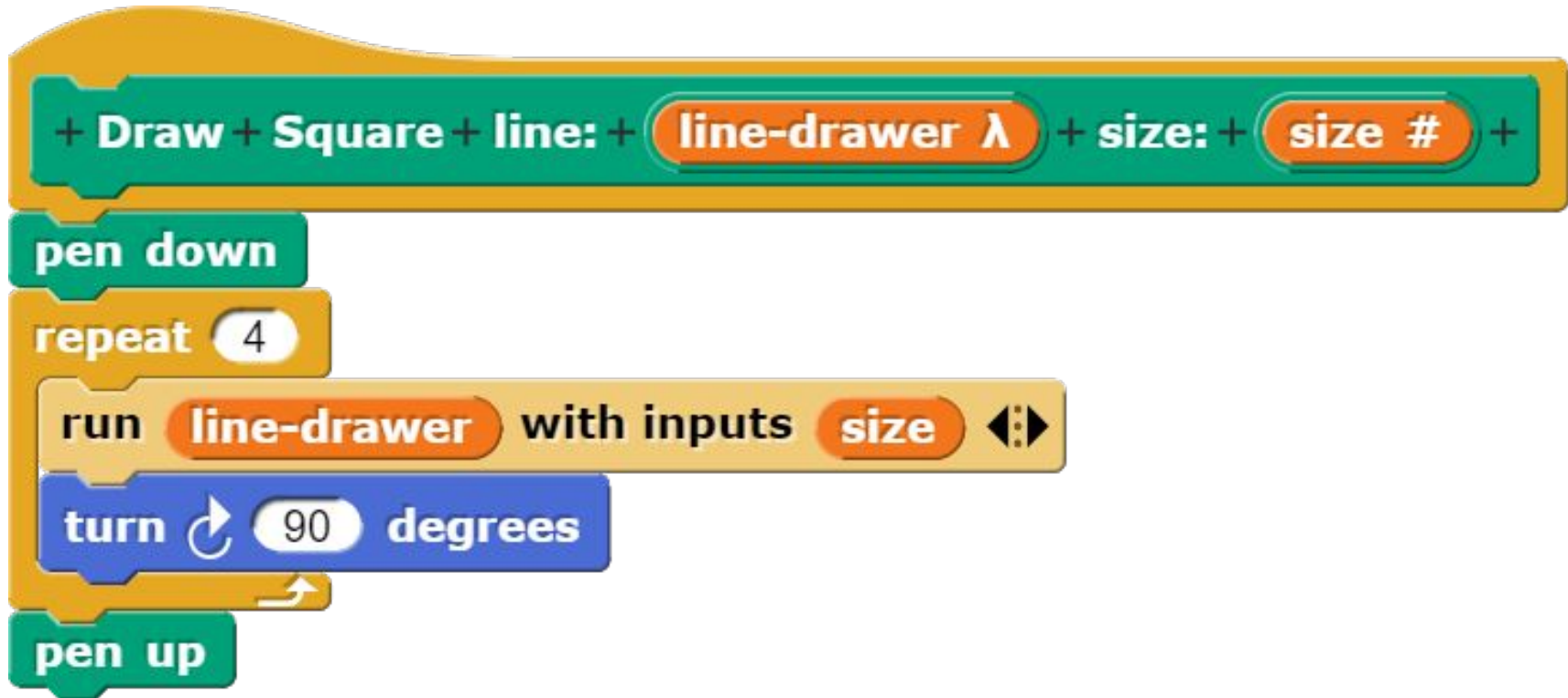
Draw Square line: **Draw Wiggly Line** ▶ size: 100

▼
Fix me! 



Build a Drawing HOF!

Objective: Generalize square to draw any line type!



Lambdas

- Defn: A temporary, anonymous function that disappears after use
- In Snap!, we denote lambdas by:



- We can also create temporary local variables for the lambdas:



- Lambdas create functions - but they're not invoked / called!
- To invoke / call, we need:

OR



Guess that Output!



Guess that Output!

```
script variables x ▶  
set x ▼ to 5 - 5  
report x
```

-5

Guess that Output!



Guess that Output!



Guess that Output!



Guess that Output!

```
script variables x ▶  
set x ▼ to [ ] - 5 ▶  
report call x with inputs [7] ◀▶
```

2

Guess that Output!

The image shows a Scratch 'call' block with the following configuration:

- call** (label on the left)
- input names:** **a** and **b** (two orange circular buttons)
- with inputs:** **2** and **6** (two white rectangular buttons)
- Expression:** $b \times b > a \times a + b$ (constructed from orange circular buttons for variables, mathematical operators, and comparison symbols, all within a green arrow-shaped container)

Guess that Output!

A Scratch 'call' block with a green arrow-shaped body. The body contains the expression $b \times b > a \times a + b$. Below the body, the 'input names' are 'a' and 'b'. The 'with inputs' field contains the values '2' and '6'.

true

Evaluates to:

$$(6 * 6) > ((2 * 2) + 6)$$

$$(36) > ((4) + 6)$$

$$(36) > (10)$$

true

Guess that Output!

The image shows a Scratch 'call' block with the following structure:

- call** (yellow block)
- input names:** `a` (orange circle)
- with inputs** `10` (white box)
- if** `a > 5` (green block with orange circle)
- then** `a - 2` (green block with orange circle)
- else** `a + 1` (green block with orange circle)

The block is designed to calculate the value of $2 \times a$ based on the condition $a > 5$. If the condition is true, it returns $a - 2$; otherwise, it returns $a + 1$.

Guess that Output!



A Scratch 'call' block with the following structure:

- Operator: `2 ×`
- Condition: `if a > 5`
- Then clause: `then a - 2`
- Else clause: `else a + 1`
- Input names: `a`
- With inputs: `10`

16

Evaluates to:

$2 * (\text{if } (10 > 5) \text{ then } (10 - 2) \text{ else } (10 + 1))$

$2 * (\text{if } (\text{true}) \text{ then } (8) \text{ else } (11))$

$2 * (8)$

16

Build that Function!

- Objective: Create the map block from scratch. You should be invoking a function on every item from a list. Do this without using map. Instead, use iteration!



Build that Function!

+ map + function: + function λ + to + list: + input-list ⋮ +

script variables output ▶

set output ▼ to list ▶

for each item in input-list

add call function with inputs item ◀▶ to output

report output

Build that Function!

- Objective: Create the keep block from scratch. You should be invoking a function on every item from a list. Do this without using map. Instead, use iteration!



Build that Function!

```
+ keep + function: + function  $\lambda$  + to + list: + input-list : +
```

```
script variables output ▶
```

```
set output ▼ to list ▶
```

```
for each item in input-list
```

```
if call function with inputs item ◀▶
```

```
add item to output
```

```
report output
```

Build that Function!

- Objective: Create the combine block from scratch. You should be invoking a function on every item from a list. Do this without using map. Instead, use iteration!

combine function:



to list:



10

Build that Function!

+ combine + function: + function λ + to + list: + input-list : +

script variables output ▶

set output ▼ to

for each item in input-list

set output ▼ to call function with inputs output item ◀▶

report output