Concurrency

Concurrency

Definition: several scripts are executing simultaneously and potentially interacting with each other.



when clicked

set final cs10 grade to D

This is how we assign grades! Based on the Birkahni Theorem, we usually get the grades to average to a B+, though due to the size of the class this semester, the average will be a C+ (jk)

1 Race Condition

Concurrency Issue

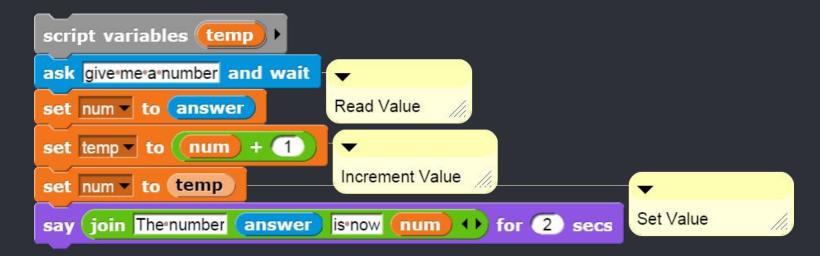
Race Condition

Definition:

When events of a program don't happen in the order that the programmer intended.

Function Definition

Read Value - reads in a value from user inputIncrements Value - increments the value, but does not set itSets Value - sets the value to the incremented version of it



```
when clicked program 1

when clicked program 2
```

```
+program+1+

script variables (temp1)

set temp1 \( \temp1 \)

set temp2 \( \temp2 \)

set temp2 \( \temp2 \)

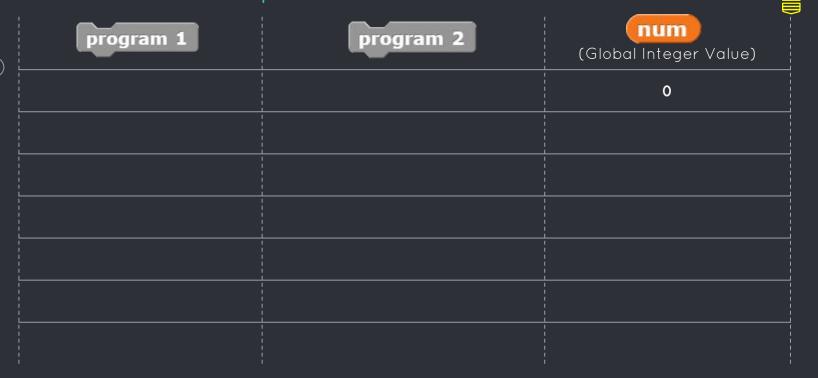
set temp2 \( \temp2 \)

set num \( \temp1 \)

set num \( \temp1 \)

to temp1
```

We have two programs, program 1 and program 2, and a global variable 'num'. Ideally, we want the script in program 1 to run before the script in program 2, but this won't always be the case. We'll look at two scenarios, the first where they run in order (serial), and the second where they don't (race condition).



'num' starts out with value 0

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o

Read Value - read the value of 'num' and sets 'temp1' to that value

program 1	program 2	(Global Integer Value)
		0
set temp1 ▼ to num		O
change temp1 ▼ by 1		0

Increments Value - increases the value of 'temp1' by 1

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		O
change temp1 ▼ by 1		0
set num ▼ to temp1		1

Sets Value - sets 'num' to the value of 'temp1', which is 1

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		0
change temp1 ▼ by 1		o
set num ▼ to temp1		1
	set temp2 ▼ to num	1

Read Value - reads the value of 'num' and sets 'temp2' to that value

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		0
change temp1 ▼ by 1		o
set num ▼ to temp1		1
	set temp2 ▼ to num	1
	change temp2 ▼ by 2	1

Increments Value - increases the value of 'temp2' by 2

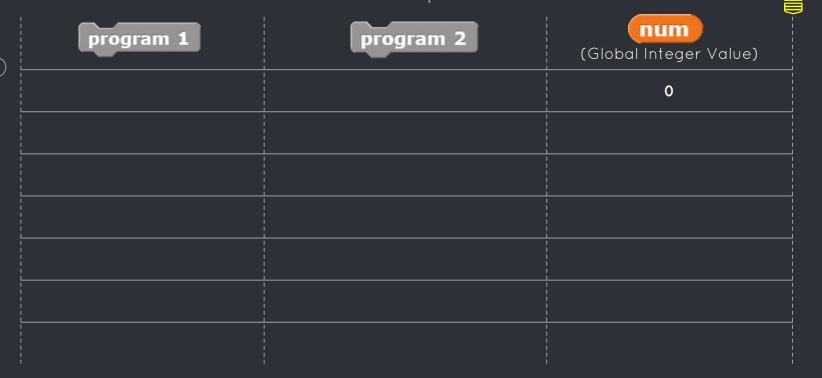
program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		0
change temp1 ▼ by 1		o
set num ▼ to temp1		1
	set temp2 ▼ to num	1
	change temp2 v by 2	1
	set num ▼ to temp2	3

Sets Value - sets 'num' to the value of 'temp2', which is 3

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o
change temp1 ▼ by 1		o
set num ▼ to temp1		1
	set temp2 ▼ to num	1
	change temp2 v by 2	1
	set num ▼ to temp2	3

This is the expected output. We're good here!

What if we interleaved the commands?



'num' starts out with the value 0

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o

Read Value - reads the value of 'num' and sets 'temp1' to that value

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		0
	set temp2 ▼ to num	0

Read Value - reads the value of 'num' and sets 'temp2' to that value

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o
	set temp2 ▼ to num	o
change temp1 ▼ by 1		0

Increments Value - increases the value of 'temp1' by 1

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o
	set temp2 ▼ to num	o
change temp1 by 1		o
	change temp2 ▼ by 2	0

Increments Value - increases the value of 'temp2' by 2

program 1	program 2	num (Global Integer Value)
		o
set temp1 ▼ to num		o
	set temp2 ▼ to num	O
change temp1 ▼ by 1		o
	change temp2 ▼ by 2	O
set num ▼ to temp1		1

Sets Value - sets 'num' to the value of 'temp1', which is 1

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		0
	set temp2 ▼ to num	o
change temp1 ▼ by 1		0
	change temp2 ▼ by 2	0
set num ▼ to temp1		1
	set num ▼ to temp2	2

Sets Value - sets 'num' to the value of 'temp2', which is 2

program 1	program 2	(Global Integer Value)
		o
set temp1 ▼ to num		o
	set temp2 ▼ to num	o
change temp1 by 1		0
	change temp2 ▼ by 2	o
set num ▼ to temp1		1
	set num ▼ to temp2	2

This is NOT the expected output. 'num' is only 2!

Takeaway

Concurrency is great because it allows for tasks to be broken up and completed almost simultaneously. However, you have to be careful how you break up the tasks so you don't get erroneous behavior.

Race Condition Example: Withdrawing

- What if two people were calling withdraw at the same time?
 - e.g., balance = 100 and two withdrew 75 each
 - Can anyone see what the problem *could* be?
 - This is a race condition.

```
+withdraw+amount +

if balance > amount

set balance ▼ to balance - amount

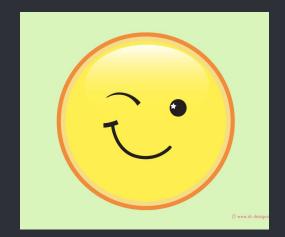
report true

report false
```

In most languages, this is a problem.

- In Scratch, the system doesn't let two of these run at once.

2 Winky Face Problem



Question 13: Your Faaaaace... (5 pts)



You want to draw a face, so you write this serial script that produces the 'winking' face right beside it:





But then you want to simulate what it would be like to parallelize the code and run it on 3 separate 'cores', so you change the serial script above into the following parallel scripts, which all run at the same time.

```
when clicked

wait 1 / pick random 1 to 10 secs

clear

wait 1 / pick random 1 to 10 secs

Draw Left Eye
```

```
when clicked

wait 1 / pick random 1 to 10 secs

clear

wait 1 / pick random 1 to 10 secs

Draw Right Eye
```

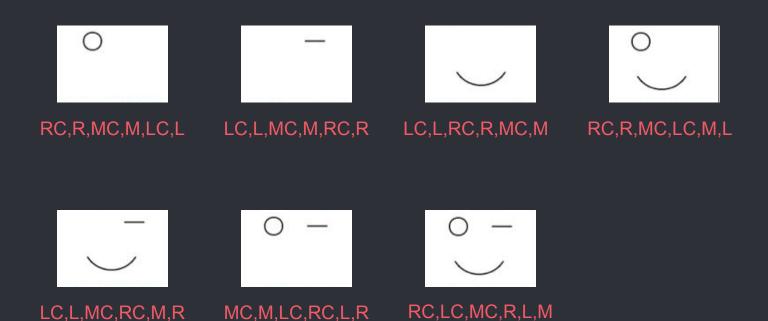


Draw all the faces that could result from running this new parallel code.

Question 13: Your Faaaaace... (5 pts)



These result from interlacing 3 LeftEye / RightEye / Mouth Clear (LC, RC, MC), LeftEye (L), RightEye(R) and Mouth(M).



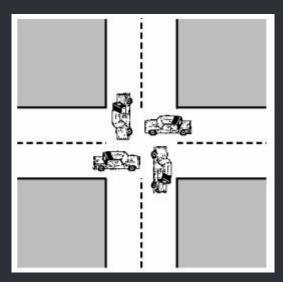
3 Deadlock

Deadlock

Definition:

A situation in which two or more competing actions are each waiting for the other(s) to finish, and thus no one ever finishes.





Deadlock Possible

Deadlock

Deadlock Example



Dining Philosopher Problem



Question 12: Dining Philosophers (5 pts)



Two philosophers (left and right) are having dinner, sitting across from each other. There is a NORTH and a SOUTH chopstick on the table. Each philosopher continually looks down to see if a chopstick is on the table, and tries to grab it; if both are ever grabbed by one person, that person eats, updates HISTORY (a record of what happened) and puts the chopsticks down. Ten seconds after the green flag is clicked, what should HISTORY be?

```
set NORTH chopsticks v to table
                                        set SOUTH chopsticks - to table
                                        set HISTORY v to Started...
                                        broadcast Eat! ▼
       when I receive Eat!
       wait 1 / pick random 1 to 10 secs
       wait until
          NORTH chopsticks = [ight] and SOUTH chopsticks = [ight]
       set HISTORY to join HISTORY right-ate (1)
       set NORTH chopsticks . to table
       set SOUTH chopsticks . to table
when I receive Eatl •
wait 1 / pick random 1 to 10
wait until NORTH chopsticks = left and SOUTH chopsticks = left
set HISTORY to join (HISTORY) left ate (1)
set NORTH chopsticks . to table
set SOUTH chopsticks . to table
```

when clicked

```
when I receive Eat!
wait 1 / pick random 1 to 10
wait until NORTH chopsticks =
set NORTH chopsticks v to left
when I receive Eatl
wait 1 / pick random 1 to 10
wait until (SOUTH chopsticks) = table
set SOUTH chopsticks v to right
when I receive Eatl .
wait 1 / pick random 1 to 10
wait until (SOUTH chopsticks) = table
set SOUTH chopsticks . to left
```

```
wait 1 / pick random 1 to 10

wait until NORTH chopsticks = table

set NORTH chopsticks to right

NORTH

NORTH

Right philosopher
```

SOUTH

when I receive Eat! *

Question 12: Dining Philosophers (5 pts)



```
Started..., Left ate..., Right ate...
                                                                                                                                  when Clicked
            Started..., Right ate..., Left ate...
                                                                                                                                  set NORTH chopsticks . to table
            Started...
                                                                                                                                  set SOUTH chopsticks - to table
                                                                                                                                  set HISTORY v to Started...
                                                                                                                                  broadcast Eat! ▼
                                                when I receive Eat! *
when I receive Eatl •
                                                                                                 when I receive Eat -
wait 1 / pick random 1 to 10 secs
                                                wait 1 / pick random 1 to 10
                                                                                                 wait 1 / pick random 1 to 10 secs
wait until NORTH chopsticks =
                                                wait until ( NORTH chopsticks ) = table
                                                                                                  wait until
set NORTH chopsticks v to left
                                                                                                     NORTH chopsticks = [ight] and SOUTH chopsticks = [ight]
                                                set NORTH chopsticks * to right
                                                                                                  set HISTORY to join HISTORY right-ate (1)
                                                               NORTH
                                                                                                 set NORTH chopsticks . to table
when I receive Eat!
                                                                                                 set SOUTH chopsticks . to table
wait 1 / pick random 1 to 10 secs
                                                                                Right philosopher
wait until (SOUTH chopsticks) = table
                                                  Left philosopher
                                                                                          when I receive Eat! *
set SOUTH chopsticks . to right
                                                                                          wait 1 / pick random 1 to 10 secs
when I receive Eat! *
                                                                                          wait until ( NORTH chopsticks ) = left and ( SOUTH chopsticks ) = left
wait 1 / pick random 1 to 10 secs
                                                                                          set HISTORY to join (HISTORY) left ate (1)
wait until (SOUTH chopsticks) = table
                                                                                          set NORTH chopsticks . to table
set SOUTH chopsticks v to left
                                                              SOUTH
                                                                                          set SOUTH chopsticks . to table
```

Some More Recursion!

Length of List (Recursively!)

Report the length of a list WITHOUT using the length of list block!

Recursive Length of List

Solution

```
+ Recursive + Length + of + List + li
```

Reverse Text (Recursively!)

Write a block that takes in some text, and reports the same text, only reversed!

Solution

```
+ Recursive + Reverse + text + text +
   1 = length of text
report text
eise
                    length of text of text
             letter
report
        join
             Recursive Reverse text all but last letter of text
```

4 See You Next Week!