CPSC 100

Computational Thinking

Sequential Algorithm + Programming

Instructor: Parsa Rajabi

Department of Computer Science

University of British Columbia

Agenda

- Learning Goals
- Course Admin
- Sequential Algorithm
- Intro to Programming

Sequential Algorithm for Signaling Words

Sequential

8 to get to the letter F

+ 1 to signal “F”

---

9 total cost

A	B	C	D	1	2
E	F	G	H	3	4
I	J	K	L	M	N
O	P	Q	R	S	T
U	V	W	X	Y	Z
5	6	7	8	9	0

iClicker

Q: What is the cost to signal the word "FAN"?

A. 26

B. 28

C. 29

[✓ **D. 30**]

E. 31

A	B	C	D	1	2
E	F	G	H	3	4
I	J	K	L	M	N
O	P	Q	R	S	T
U	V	W	X	Y	Z
5	6	7	8	9	0

Breaking Bad Algorithm

https://www.youtube.com/watch?v=pWso-qRaIlM#t=26s

Example of cost counting: Letter F

2 to get to the “E” row
1 to signal the “E” row
2 to get to “F” in the row
+1 to signal “F”

6 total cost

A	B	C	D	1	2
E	F	G	H	3	4
I	J	K	L	M	N
O	P	Q	R	S	T
U	V	W	X	Y	Z
5	6	7	8	9	0

Breaking Bad Algorithm

Q: What is the cost to signal the word “FAN”?

iClicker

A. 17

B. 18

C. 19

D. 20

E. 21

A	B	C	D	1	2
E	F	G	H	3	4
I	J	K	L	M	N
O	P	Q	R	S	T
U	V	W	X	Y	Z
5	6	7	8	9	0

Activity

Activity: Algorithms in Action

Find a word that works better the Sequential way.

Find a word that works better the Breaking Bad way.

Which algorithm is better and why?

Rules:

Both words must be at least 4 letters!

Use the same chart

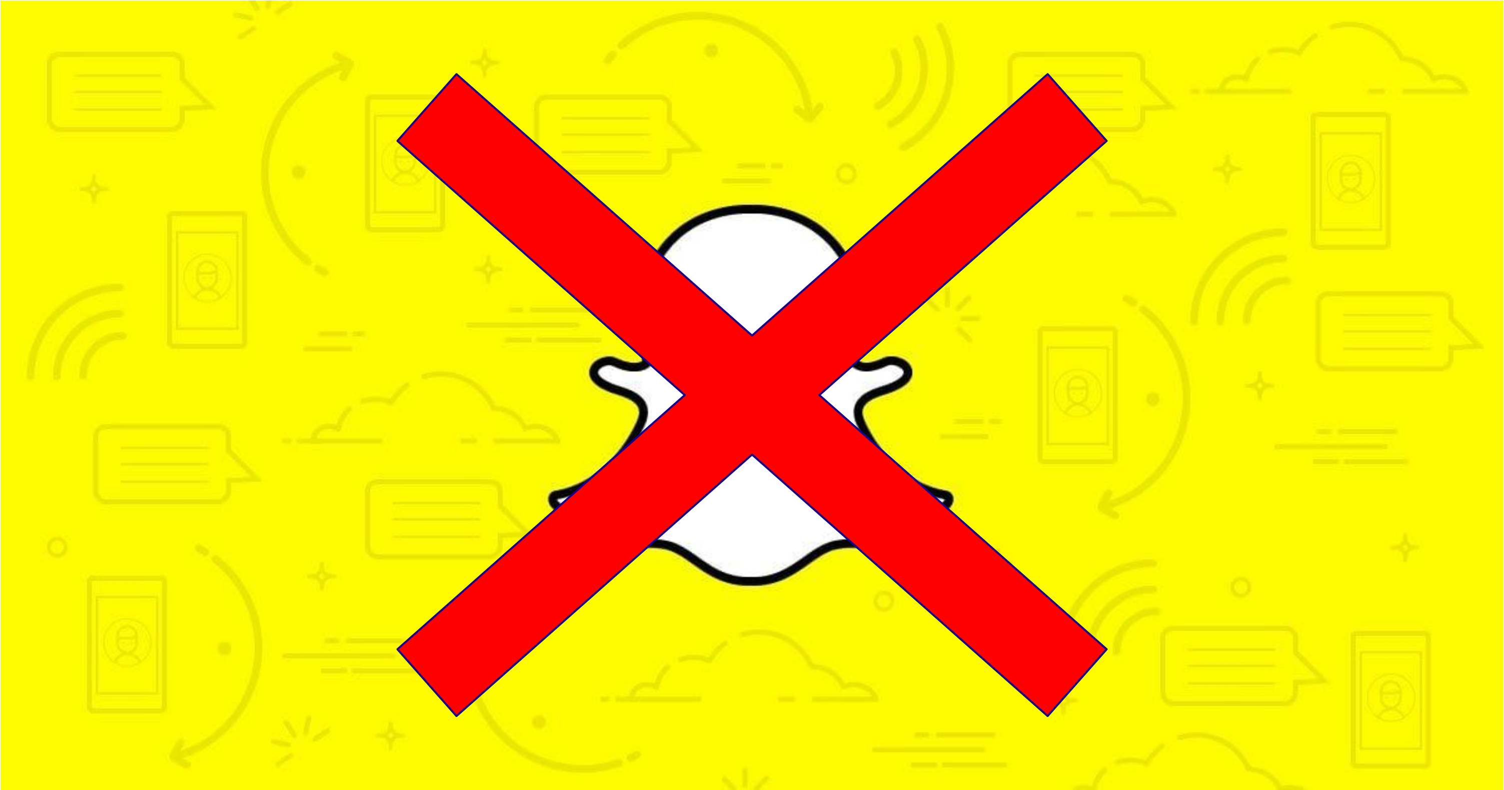
Programming

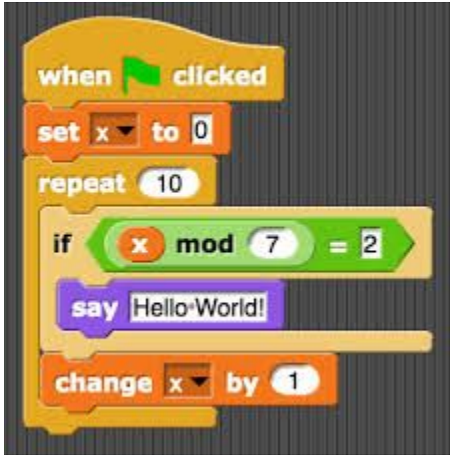
This is *not* a programming courses

But you do need to *understand* how programs work

We’ll cover a small amount of basic concepts in class and you’ll work on a visual language in lab

Snap!





```
when [flag] clicked
set x to 0
repeat 10
  if (x mod 7) = 2
    say Hello World!
  change x by 1
```

From algorithms to code: How do programs work?

How do programs work?

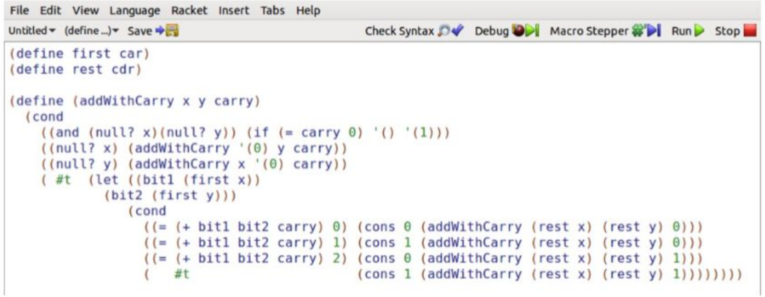
Programs are a way of encoding ***algorithms*** in a precise enough way for computers to understand the instructions.

How do programs work?

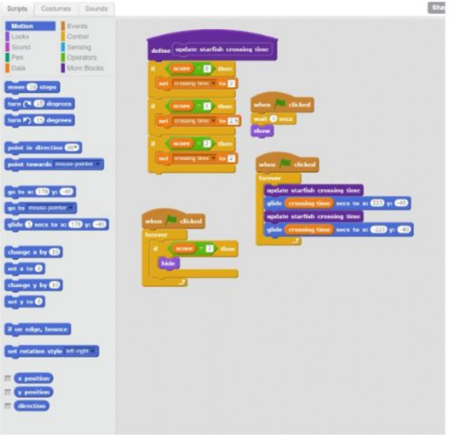
Programs are a way of encoding ***algorithms*** in a precise enough way for computers to understand the instructions.

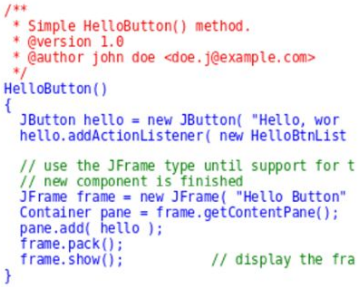
Programmers use a **high level language** like Snap, Scratch, Python, C++, Java, Racket, etc.

These languages may look very different

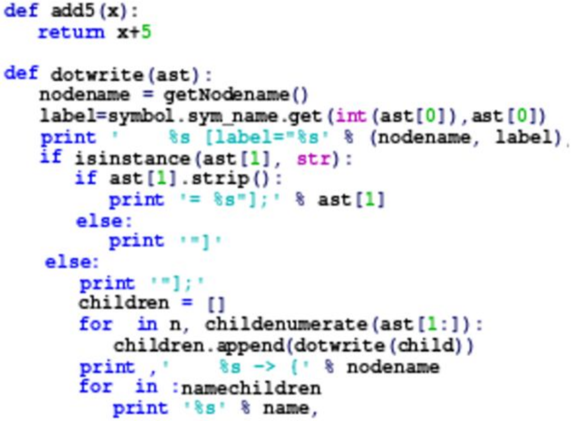

File Edit View Language Racket Insert Tabs Help
Untitled (define ...) Save
Check Syntax Debug Macro Stepper Run Stop

```
(define first car)
(define rest cdr)

(define (addWithCarry x y carry)
  (cond
    ((and (null? x)(null? y)) (if (= carry 0) '() '(1)))
    ((null? x) (addWithCarry '(0) y carry))
    ((null? y) (addWithCarry x '(0) carry))
    ( #t  (let ((bit1 (first x))
               (bit2 (first y)))
                  (cond
                    ((= (+ bit1 bit2 carry) 0) (cons 0 (addWithCarry (rest x) (rest y) 0)))
                    ((= (+ bit1 bit2 carry) 1) (cons 1 (addWithCarry (rest x) (rest y) 0)))
                    ((= (+ bit1 bit2 carry) 2) (cons 0 (addWithCarry (rest x) (rest y) 1)))
                    (   #t                     (cons 1 (addWithCarry (rest x) (rest y) 1))))))))
```

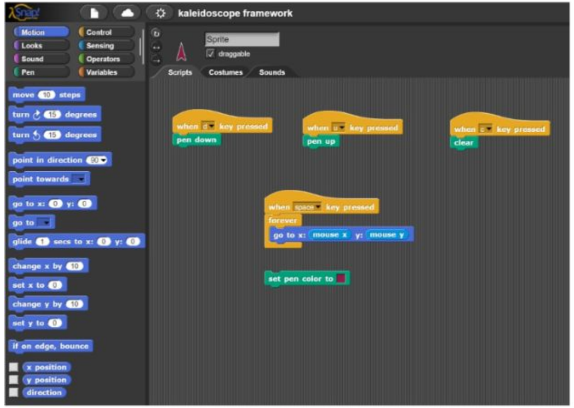


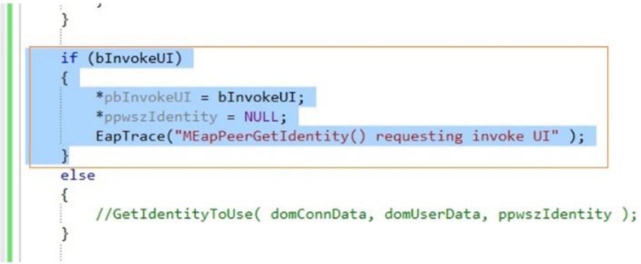


```
/**
 * Simple HelloButton() method.
 * @version 1.0
 * @author john doe <doe.j@example.com>
 */
HelloButton()
{
  JButton hello = new JButton( "Hello, wor
  hello.addActionListener( new HelloBtnList

  // use the JFrame type until support for t
  // new component is finished
  JFrame frame = new JFrame( "Hello Button"
  Container pane = frame.getContentPane();
  pane.add( hello );
  frame.pack();
  frame.show();            // display the fra
}
```



```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
     else:
        print '"];'
        children = []
        for  in n, childenumerate(ast[1:]):
            children.append(dotwrite(child))
        print ,'    %s -> {' % nodename
        for  in :namechildren
            print '%s' % name,
```





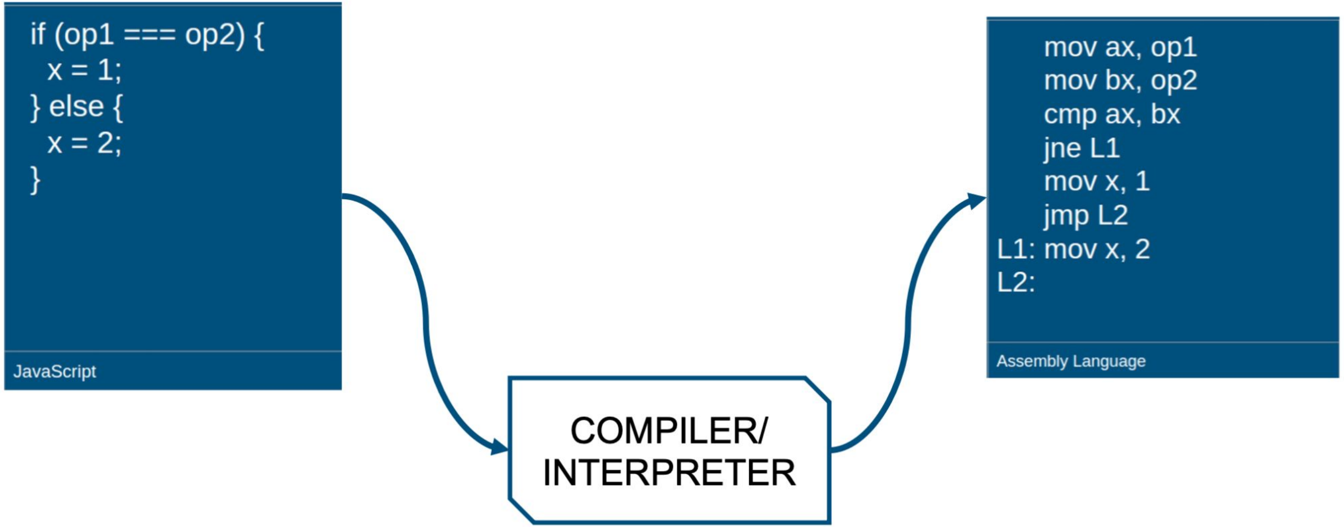
```
    }
    if (bInvokeUI)
    {
        *pbInvokeUI = bInvokeUI;
        *ppwszIdentity = NULL;
        EapTrace("MEapPeerGetIdentity() requesting invoke UI" );
    }
    else
    {
        //GetIdentityToUse( domConnData, domUserData, ppwszIdentity );
    }
```

From high to low level programming

Compiler/Interpreter

Compiler / Interpreter

Take a high level language and translates it into something that looks about the same, regardless of which high level language is used.



```
if (op1 === op2) {
  x = 1;
} else {
  x = 2;
}
```

JavaScript

COMPILER/
INTERPRETER

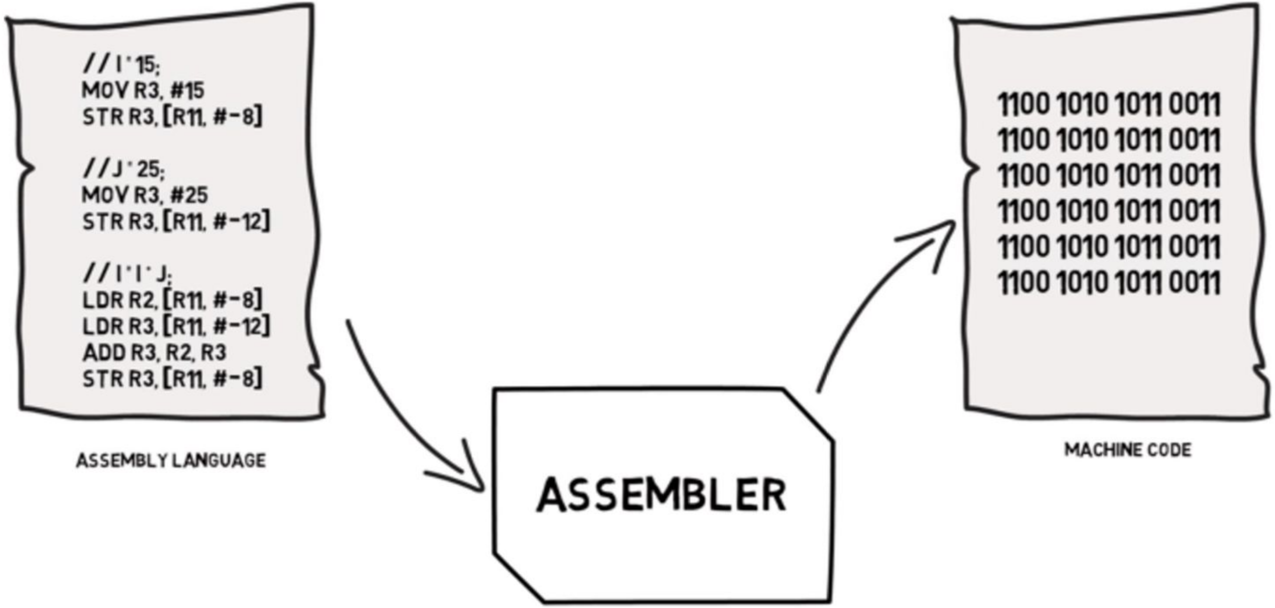
```
    mov ax, op1
    mov bx, op2
    cmp ax, bx
    jne L1
    mov x, 1
    jmp L2
L1: mov x, 2
L2:
```

Assembly Language

Assembler

Assembler

An **assembler** translates from Assembly language to Machine Code



```
// I · 15;
MOV R3, #15
STR R3, [R11, #-8]

// J · 25;
MOV R3, #25
STR R3, [R11, #-12]

// I · I · J;
LDR R2, [R11, #-8]
LDR R3, [R11, #-12]
ADD R3, R2, R3
STR R3, [R11, #-8]
```

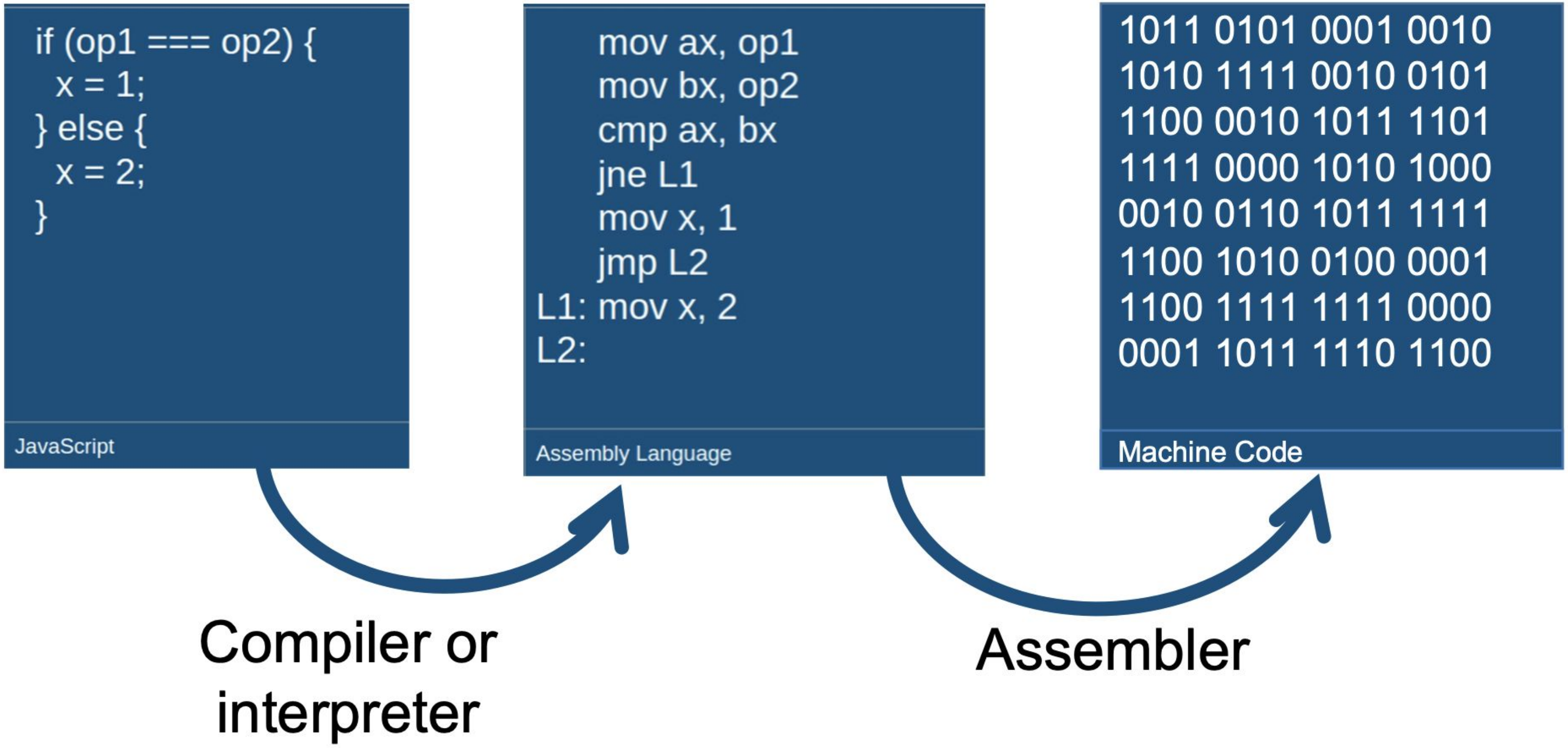
```
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
```

Differences

High-level programming languages enable us to **write programs** that are **portable** across **different** machines. They are **closer** to human languages.

Assembly language is **specific** to a particular computer architecture and operating system.

Machine code consists of binary (0/1) or hexadecimal (e.g. 7B316) instructions that a computer can respond to directly.



```
if (op1 === op2) {
  x = 1;
} else {
  x = 2;
}
```

JavaScript

```
       mov ax, op1
       mov bx, op2
       cmp ax, bx
       jne L1
       mov x, 1
       jmp L2
L1: mov x, 2
L2:
```

Assembly Language

```
1011 0101 0001 0010
1010 1111 0010 0101
1100 0010 1011 1101
1111 0000 1010 1000
0010 0110 1011 1111
1100 1010 0100 0001
1100 1111 1111 0000
0001 1011 1110 1100
```

Machine Code

Compiler or interpreter

Assembler

Bread Making Algorithm

Components of an Algorithm - Bread Making

Repeat 10 times:

1. Preheat oven (400° C)
2. Combine ingredients in bowl to form dough
3. Put dough into bread pan
4. If ingredients contain yeast, allow to sit at room temperature for 1 hour
5. Put bread pans into preheated oven and bake for 30 minutes

Recall

Algorithms

Algorithms

Recall

An ***algorithm*** describes a sequence of steps that is:

1. Unambiguous
   - No “assumptions” are required to execute the algorithm
   - The algorithm uses precise instructions
2. Executable
   - The algorithm can be carried out in practice
3. Terminating
   - The algorithm will eventually come to an end, or halt

Components of an Algorithm

An ***algorithm*** is a precise, systematic method for producing a specified result.

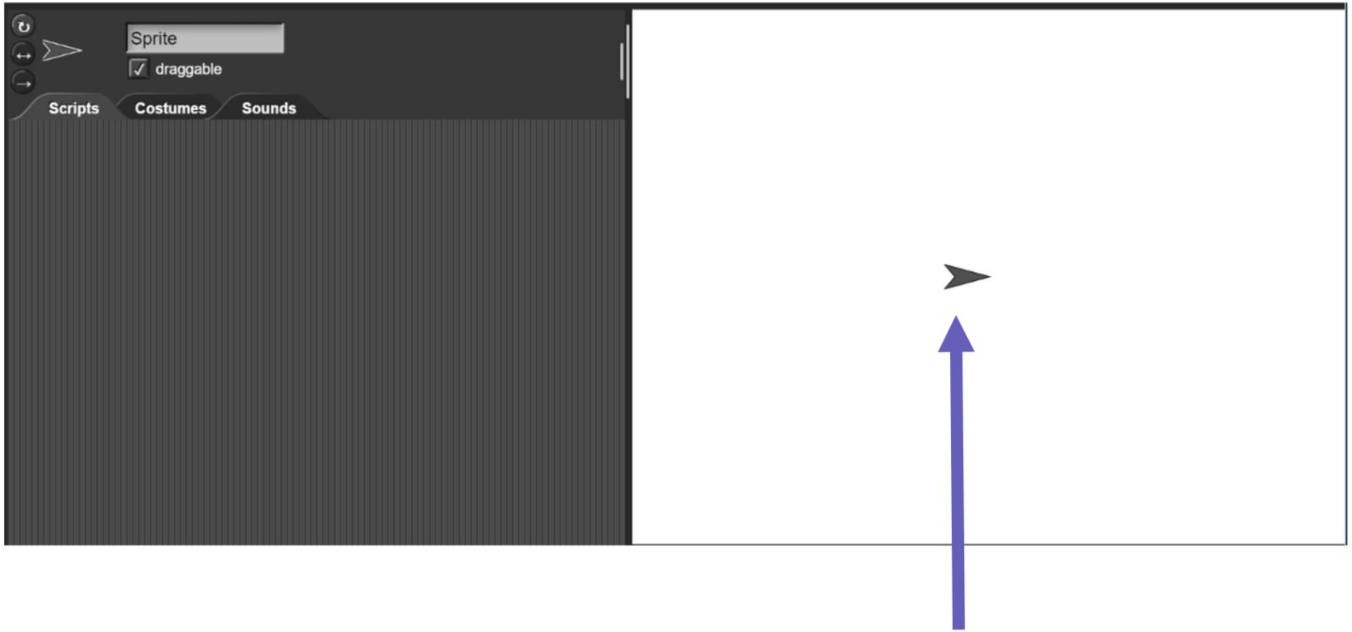
In 1966 it was proved that any algorithm can be made with only three “ingredients”:

1. Sequencing
2. Selection
3. Iteration

Intro to Snap!

Intro to Snap!

We call our screen our “**stage**”. “Things” we add are called **sprites**. A sprite is an object you can move on a larger scene.

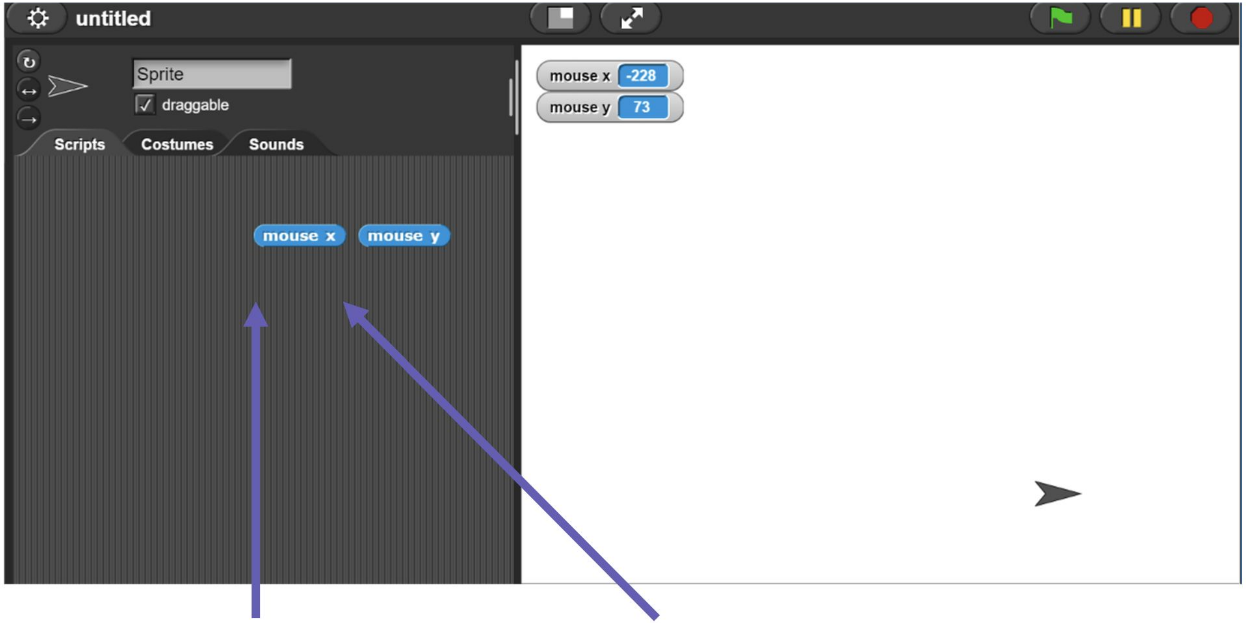


Code area (currently blank)

Sprite

Intro to Snap!

Most things that we need to keep track of, we track with ***variables*** (named quantities)

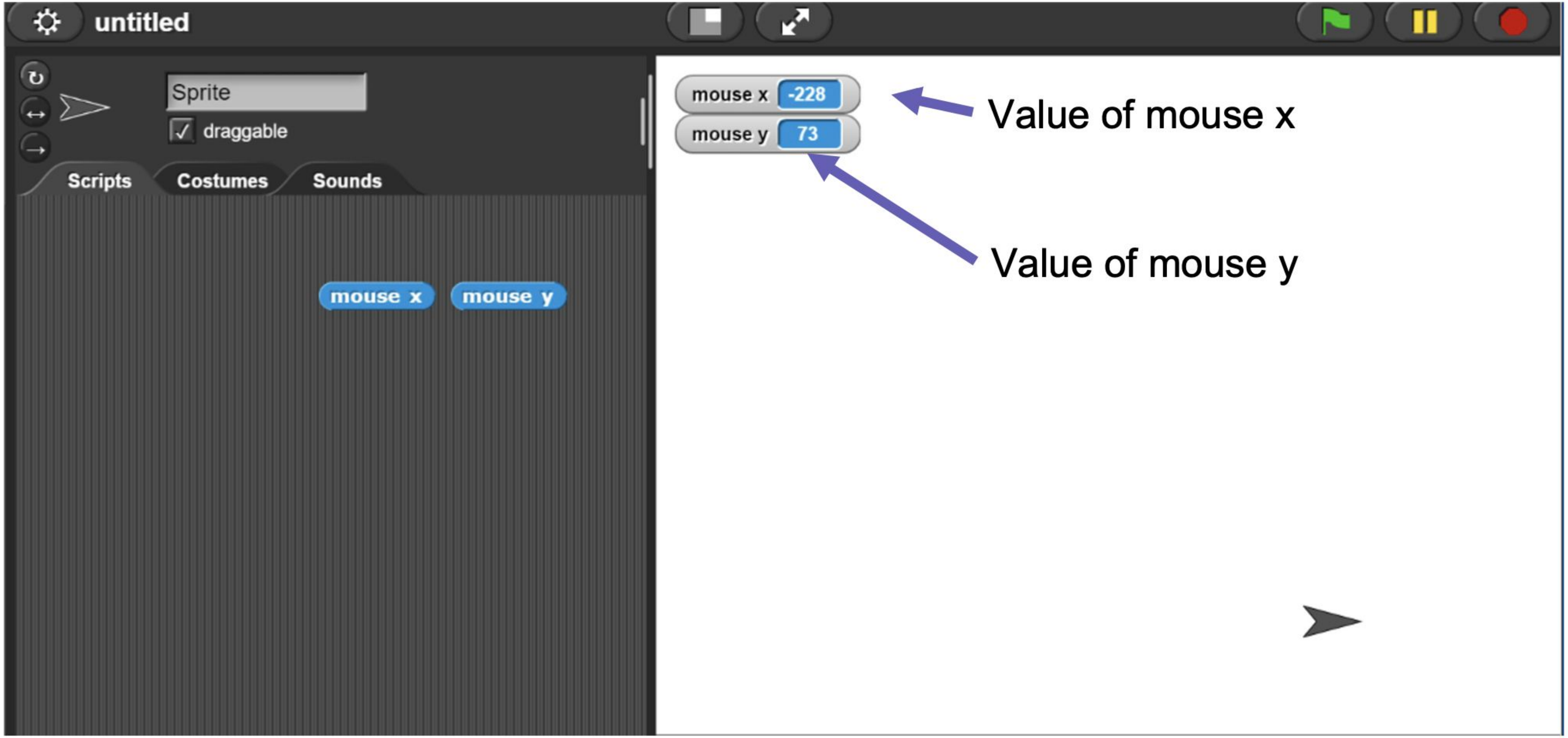


Variable: where on the x axis is the mouse?

Variable: where on the y axis is the mouse?

Intro to Snap!

Variables have a **name** and a **value**

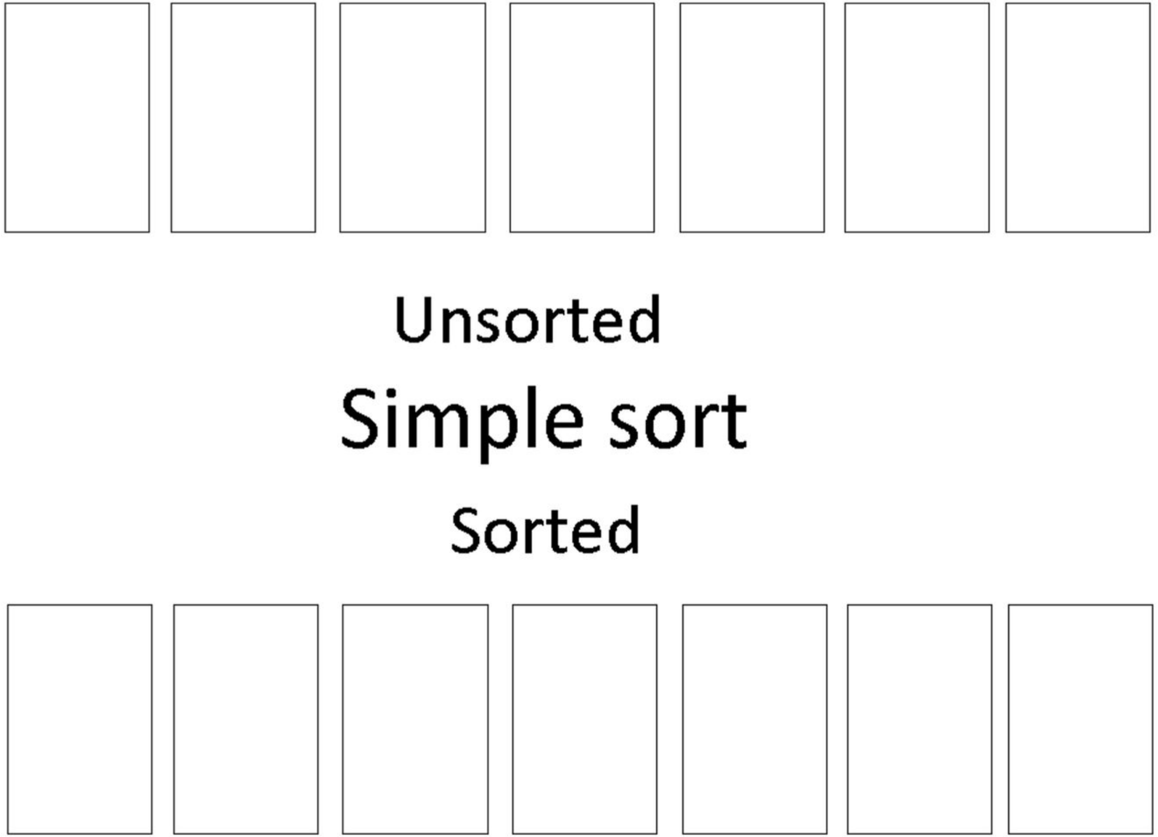


Value of mouse x

Value of mouse y

Variables

When we sorted cards, each slot was a variable



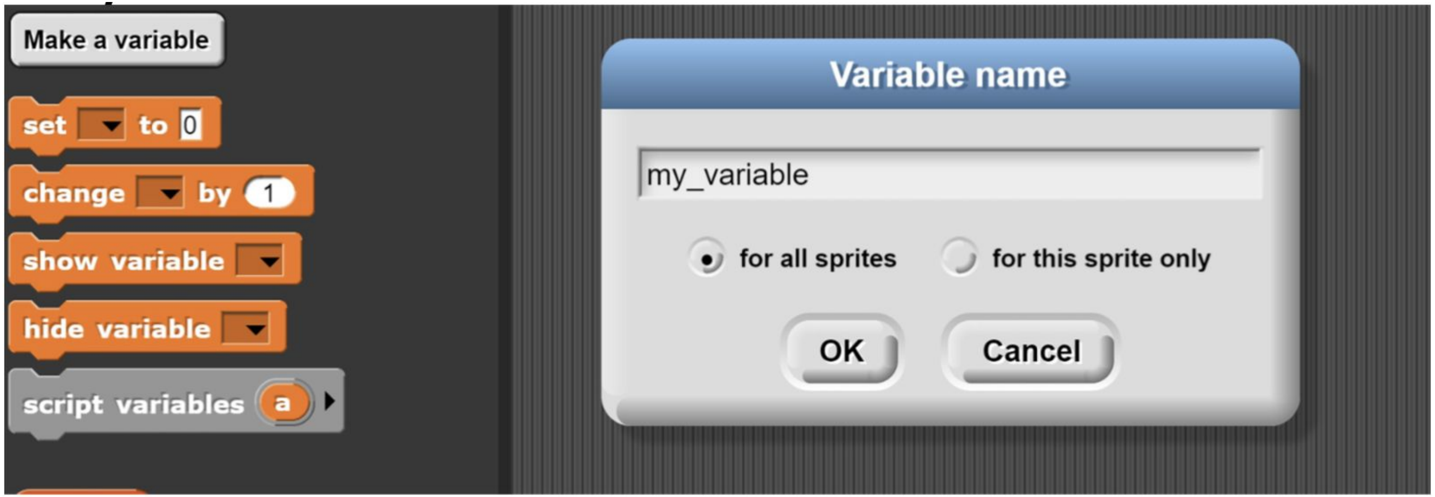
Unsorted

Simple sort

Sorted

We can use variables in our code!

Some variables are built in (e.g., “answer” is the answer to a question in Snap). You can make your own variables:



Variable name

my_variable

● for all sprites ○ for this sprite only

Variables are (usually) in orange. Things that are black writing on white are constants – actually that value

Q: What is the value of “my_variable” after the following code is run?

iClicker

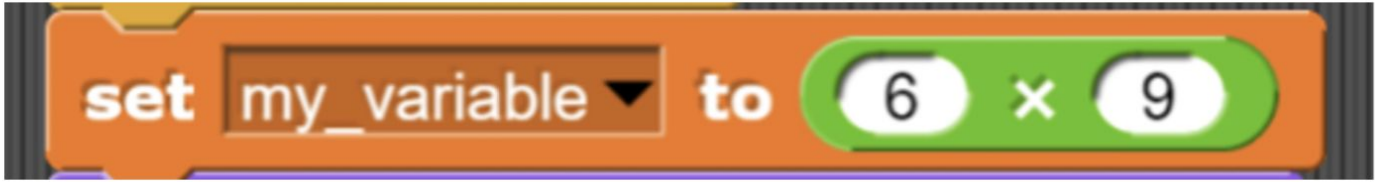
A. 6

B. 9

C. 42

D. 54

E. None of the above



```
set my_variable ▾ to (6 × 9)
```

Recall

Components of an Algorithm

1. Sequencing
2. Selection
3. Iteration

Recall

Components of an Algorithm

1. **Sequencing**
2. Selection
3. Iteration

Sequencing

Instructions are executed in the specified order

Repeat 10 times:

1. Preheat oven (400° C)
2. Combine ingredients in bowl to form dough
3. Put dough into bread pan
4. If ingredients contain yeast, allow to sit at room temperature for 1 hour
5. Put bread pans into preheated oven and bake for 30 minutes

Sequencing

Order matters

Programs will execute exactly in the order that’s given:

1. A
2. B
3. C

If we assign values to variables, they’ll set one value after another after another.

Q: What is the value of “Max CP” after all lines of this program are executed?

iClicker

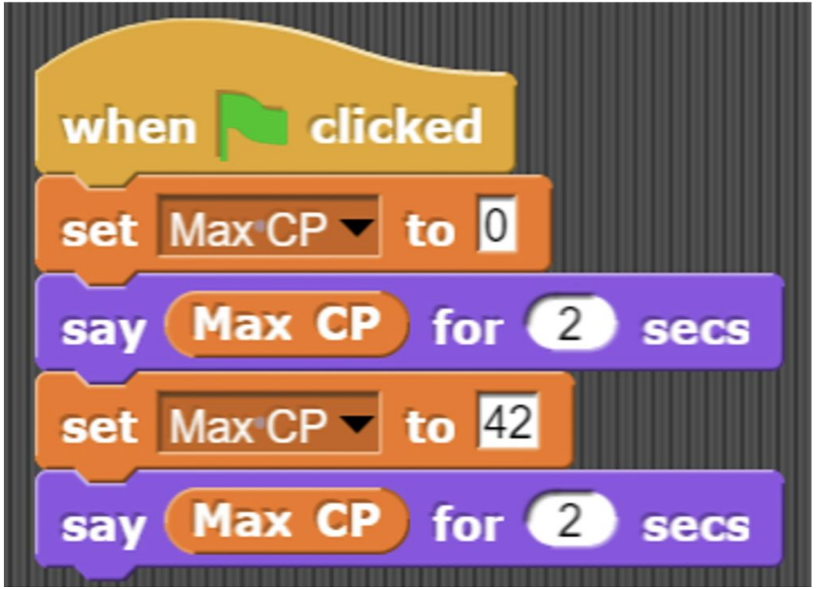
A. 0

B. 2

C. 42

D. 042

E. None of the above



```
when ⚑ clicked
set Max CP ▾ to 0
say Max CP for 2 secs
set Max CP ▾ to 42
say Max CP for 2 secs
```

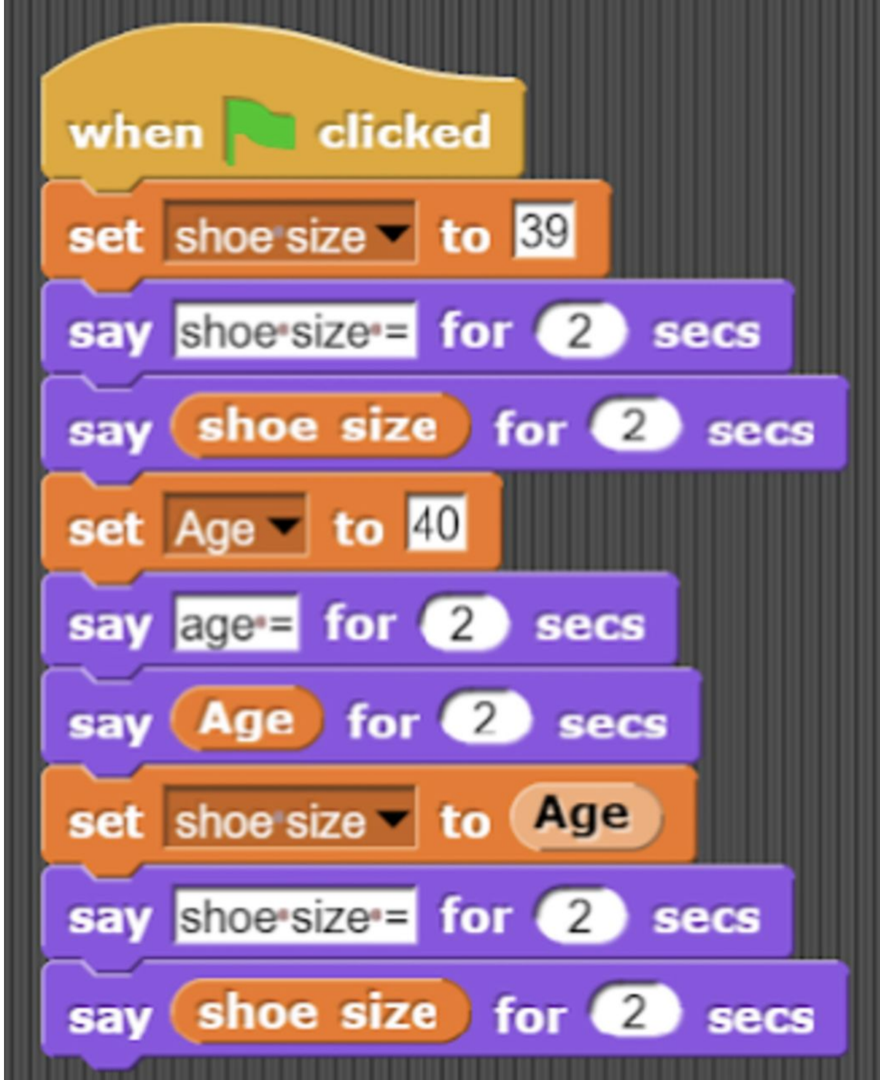
Q: What is the value of “shoe size” after all lines of the program are executed?

A. 2

B. 39

C. 40

D. 3940



```
when ⚑ clicked
set shoe size ▾ to 39
say shoe size = for 2 secs
say shoe size for 2 secs
set Age ▾ to 40
say age = for 2 secs
say Age for 2 secs
set shoe size ▾ to Age
say shoe size = for 2 secs
say shoe size for 2 secs
```

Mutation

Mutation

Process of **changing the state** or data of an **object** after it has been created.

Repeat 10 times:

1. Preheat oven (400^{o} C)
2. **Combine ingredients in bowl to form dough**
3. Put dough into bread pan
4. If ingredients contain yeast, allow to sit at room temperature for 1 hour
5. Put bread pans into preheated oven and bake for 30 minutes

Q: What is the value of “age” after all lines of this program are executed?

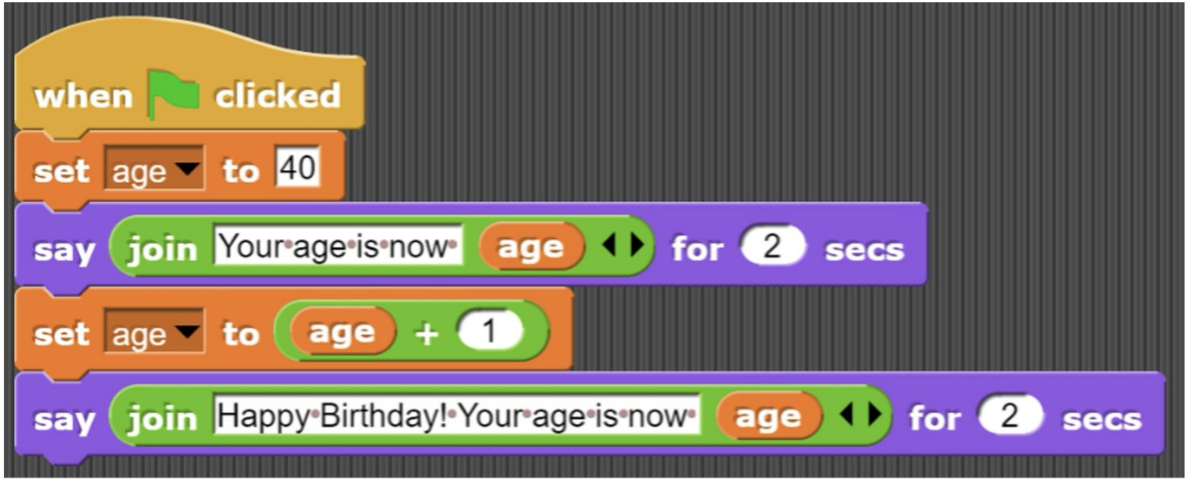
iClicker

A. 1

B. 40

C. 41

D. 401



```
when 🏴 clicked
set age ▾ to 40
say join Your·age·is·now· age ◀▶ for 2 secs
set age ▾ to age + 1
say join Happy·Birthday!·Your·age·is·now· age ◀▶ for 2 secs
```

Components of an Algorithm

1. ~~Sequencing~~
2. **Selection**
3. Iteration

Selection

Allows the algorithm to select which instructions to execute

(depending on conditions)

Repeat 10 times:

1. Preheat oven (400° C)
2. Combine ingredients in bowl to form dough
3. Put dough into bread pan
4. **If ingredients contain yeast, allow to sit at room temperature for 1 hour**
5. Put bread pans into preheated oven and bake for 30 minutes

Selection → Conditionals

Conditionals allow for different results, depending on input.

Generally this looks like “**if**” with a possible “**else**”:

Real World:

If you eat your dinner

Then you may have some ice cream

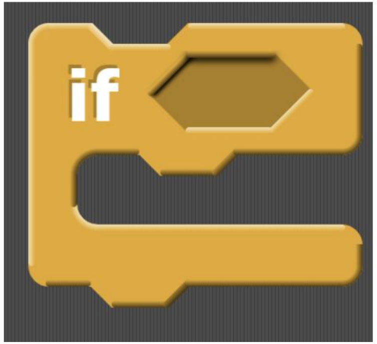
Else (you may have referred to this step as "otherwise" in Lab 2)

You may only have fruit

Selection → Conditionals

If ingredients contains yeast

    allow to sit at room temp. for 1hr

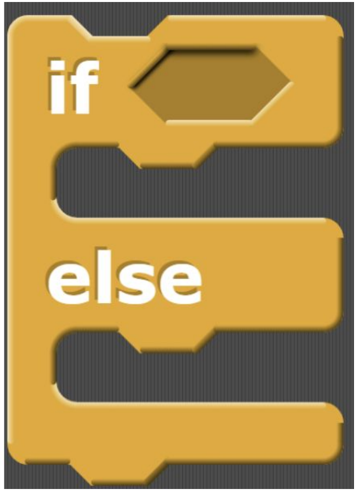


If it's snowing

    Wake-up 6:30am

Else

    Wake-up at 7am



Q: What's the output after we press run?

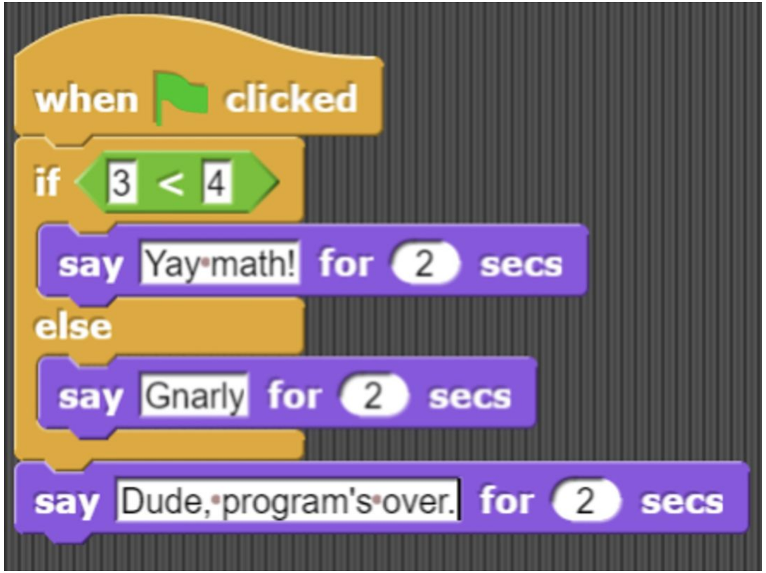
iClicker

A. It only says “Yay math!”

B. It says “Yay math!” Then…
it says “Gnarly” Then…
it says “Dude, program’s over.”

C. It says “Gnarly” Then…
it says “Dude, program’s over.”

D. It says something else



```
when [flag] clicked
if <3 < 4>
  say Yay math! for 2 secs
else
  say Gnarly for 2 secs
say Dude, program's over. for 2 secs
```

Q: What is the output if you press the “P” key?

iClicker

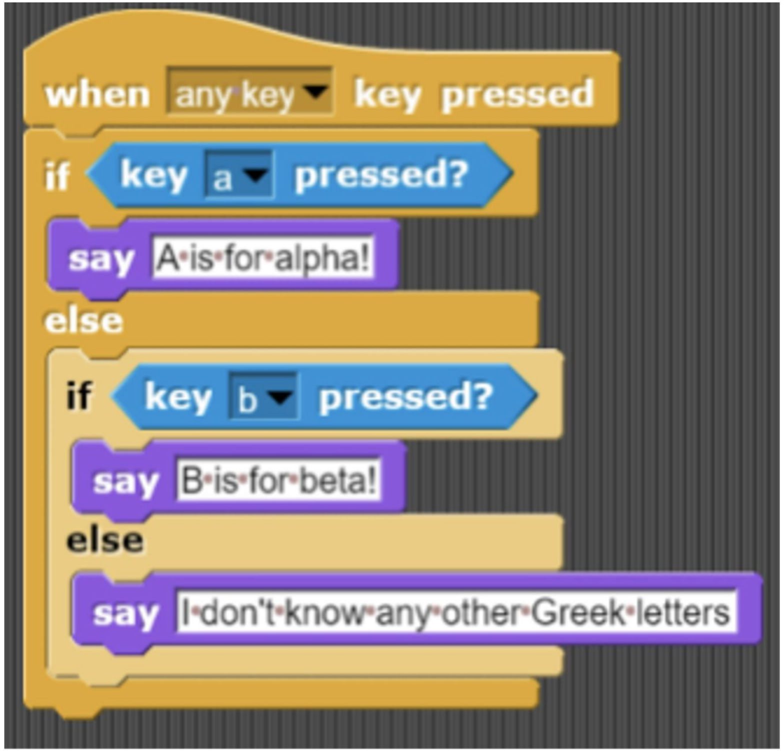
A. “A is for alpha”

B. “B is for beta”

C. “I don’t know any other Greek letters”

D. The program crashes

E. No output



```
when any key key pressed
if key a pressed?
    say A is for alpha!
else
    if key b pressed?
        say B is for beta!
    else
        say I don't know any other Greek letters
```

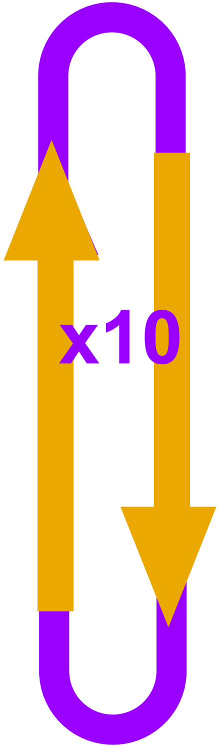
Recall

Components of an Algorithm

1. ~~Sequencing~~
2. ~~Selection~~
3. **Iteration**

Iteration

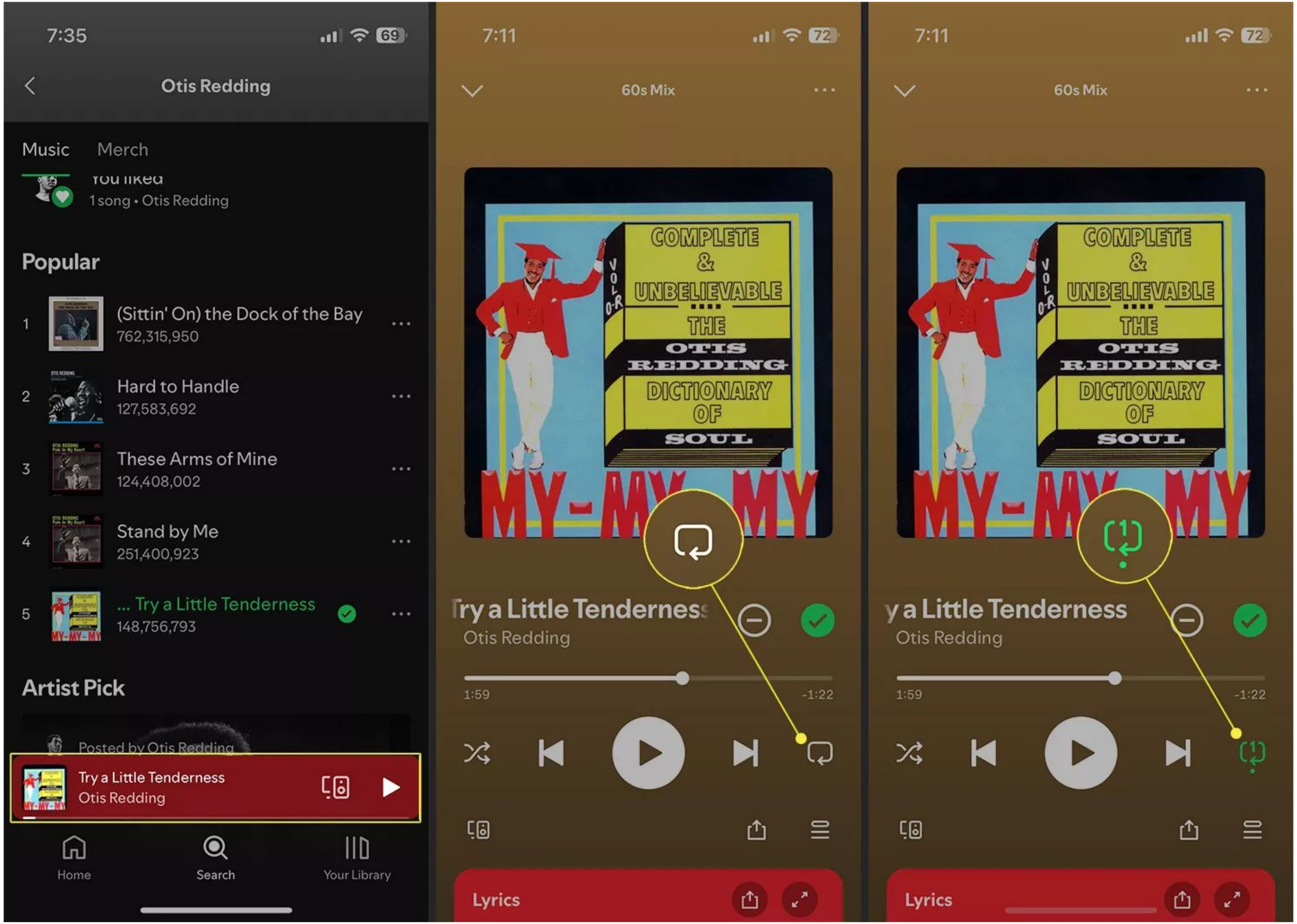
Allows the algorithm to repeat instructions.



Repeat 10 times:

1. Preheat oven (400° C)
2. Combine ingredients in bowl to form dough
3. Put dough into bread pan
4. If ingredients contain yeast, allow to sit at room temperature for 1 hour
5. Put bread pans into preheated oven and bake for 30 minutes

LIFE GUARD
2 FT 6 IN 0.8 M
2 FT 6 IN 0.8 M



7:35

Otis Redding

Music Merch

You liked
1 song • Otis Redding

Popular

1	(Sittin' On) the Dock of the Bay	762,315,950
2	Hard to Handle	127,583,692
3	These Arms of Mine	124,408,002
4	Stand by Me	251,400,923
5	... Try a Little Tenderness	148,756,793

Artist Pick

Posted by Otis Redding

Try a Little Tenderness
Otis Redding

Home Search Your Library

7:11

60s Mix

Try a Little Tenderness
Otis Redding

1:59 -1:22

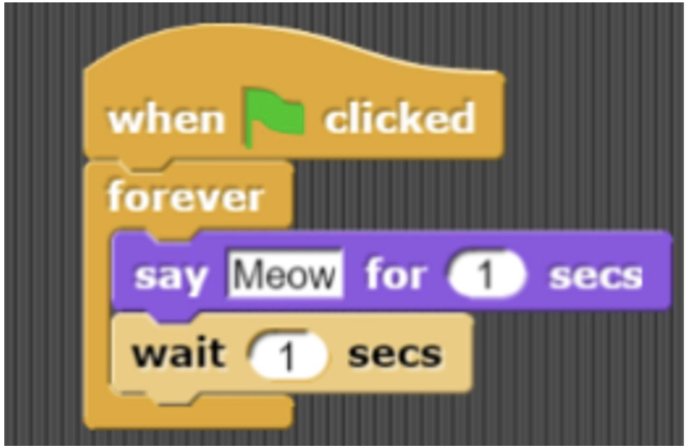
Lyrics

Source/ Guide

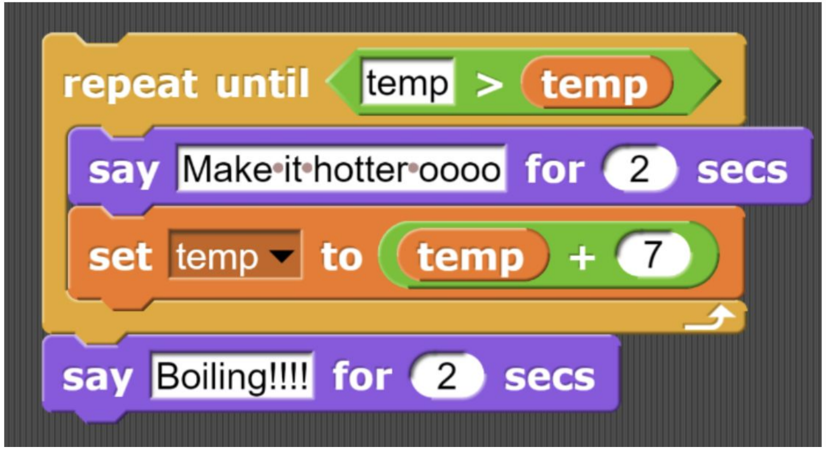
Iteration

What if you want to do a task over and over again?

A **loop** allows you to do the same task over & over again, sometimes with a **stopping** condition, sometimes **forever!**



```
when ⚑ clicked
forever
    say Meow for 1 secs
    wait 1 secs
```



```
repeat until temp > temp
    say Make·it·hotter·oooo for 2 secs
    set temp to temp + 7
say Boiling!!!! for 2 secs
```

Placeholder for iClicker

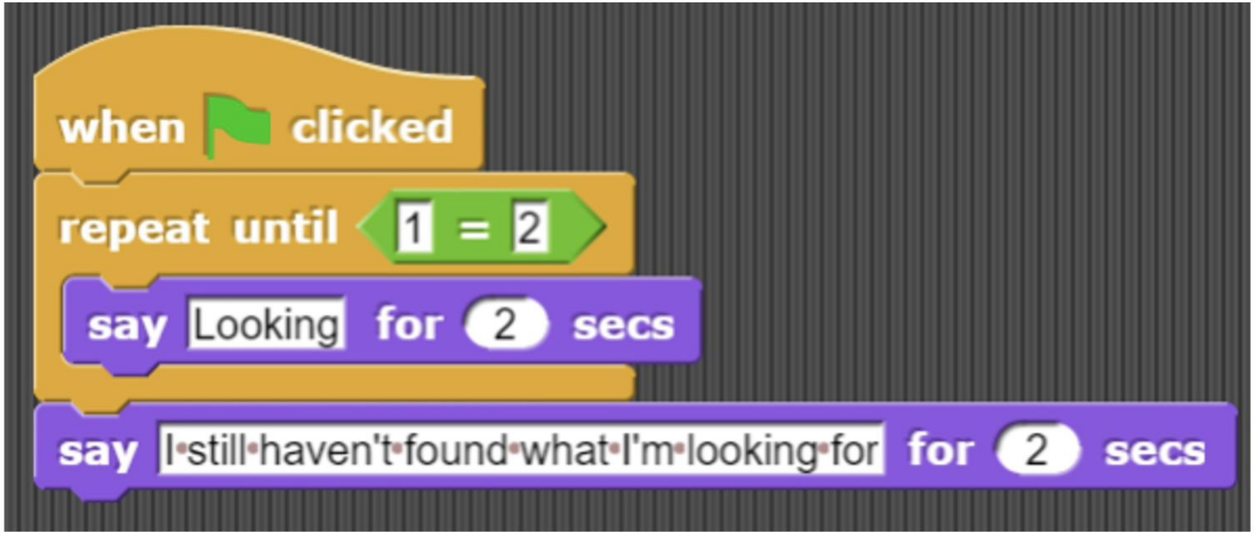
Q: Will this program ever say “I still haven’t found what I’m looking for”?

iClicker

A. Yes

B. No

C. Sometimes

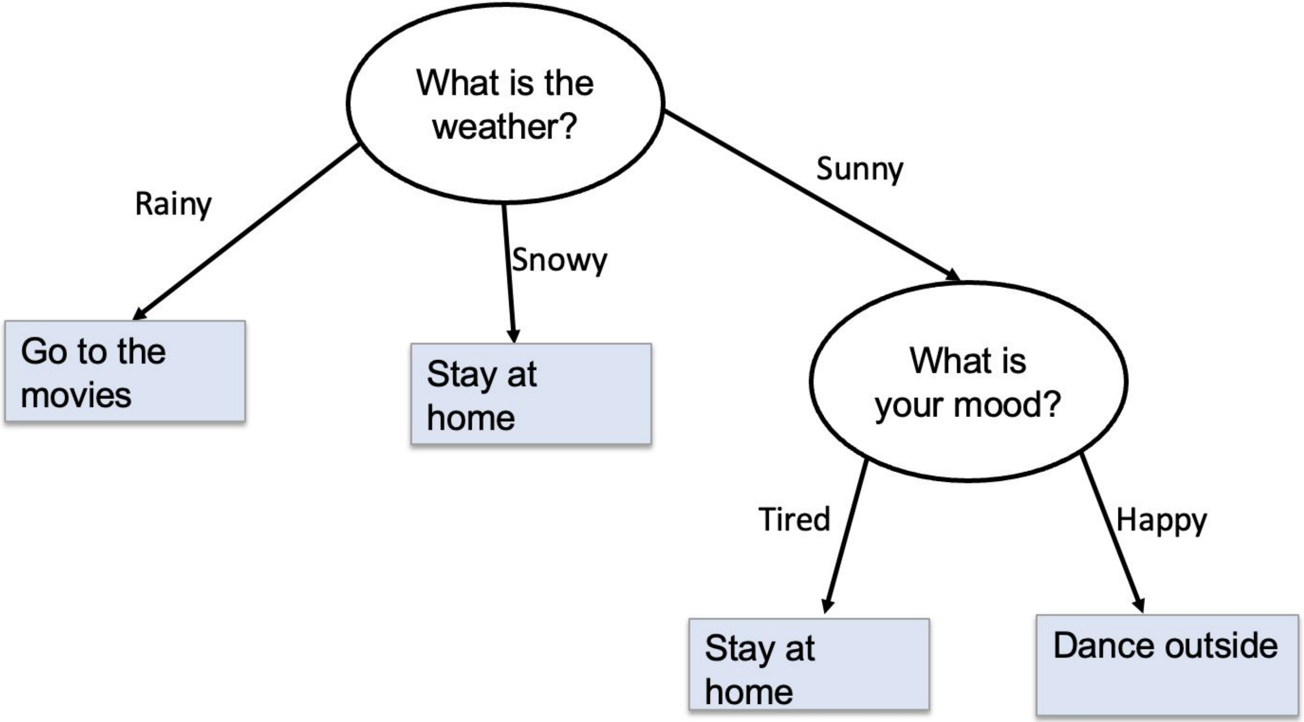


```
when ⚑ clicked
repeat until <(1) = (2)>
    say [Looking] for (2) secs
say [I still haven't found what I'm looking for] for (2) secs
```

Activity

Activity: Conditionals in Snap!

Convert the following decision tree to a Snap Block program



Wrap up