Breaking Bad Algorithm

iClicker

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

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

CPSC 100

Computational Thinking

Intro to Programming

Instructor: Parsa Rajabi

Department of Computer Science

University of British Columbia

Agenda

- Learning Goals
- Course Admin
- Follow-up: BB Algorithm
- Intro to Programming

Learning Goals

After this week's lecture, you should be able to:

- Identify the differences between sequential and "breaking bad" algorithms
- Discuss the **difference** between high level, assembly & machine code.
- **Identify and describe** the components of an algorithm
  - (i.e., sequencing, selection, and iteration)
- **Use snap blocks** to represent algorithms
- Be able to **trace** through code using sequences of instructions, variables, loops, and conditional statements in short programs
  - Read carefully: it says be able to **trace** code, **not write code**. In order to help you do this, you will write a small amount of code in lab. You will not, however, be asked to write code on exam.
- Describe in English what a block of *Snap!* code does.

Course Admin

Course Admin

- **Lab #3**
  - Intro to Snap!
  - Due on Thursday, Jan 30 at 11:59pm
- **Post-Class (PC) Quiz #1**
  - Questions? Contact your lab TA.
- **Post-Class (PC) Quiz #2**
  - Only 1 attempt, 60 minutes
  - Due on Sunday, Feb 2 at 11:59pm
- **Project**
  - Milestone 1 - Proposal (5%) **Feb. 12**
  - You should have started by now! If not, start **today**.

Snow day?

- We will keep a very close eye on the weather
- UBC/CS has a process in place and we shall follow their guidance
- You should keep a close eye on **canvas announcements**
- In case it snows AND UBC is still open…
  - We will use our best judgement
  - I will inform you of **any changes**
  - By default, we **WILL** have class, unless stated otherwise
- Top priority: **safety**! Take care of yourself first, then help others.

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

Example of cost counting: Letter A

1 to get to the "A" row

1 to signal the "A" row

1 to get to "A" in the row

+1 to signal "A"

---

4 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

Example of cost counting: Letter N

3 to get to the “I” row
1 to signal the “I” row
6 to get to “N” in the row
+1 to signal “A”

---

11 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 Alg.

- F = 6 total cost
- A = 4 total cost
- N = 11 total cost

21 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

16

Take-home 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

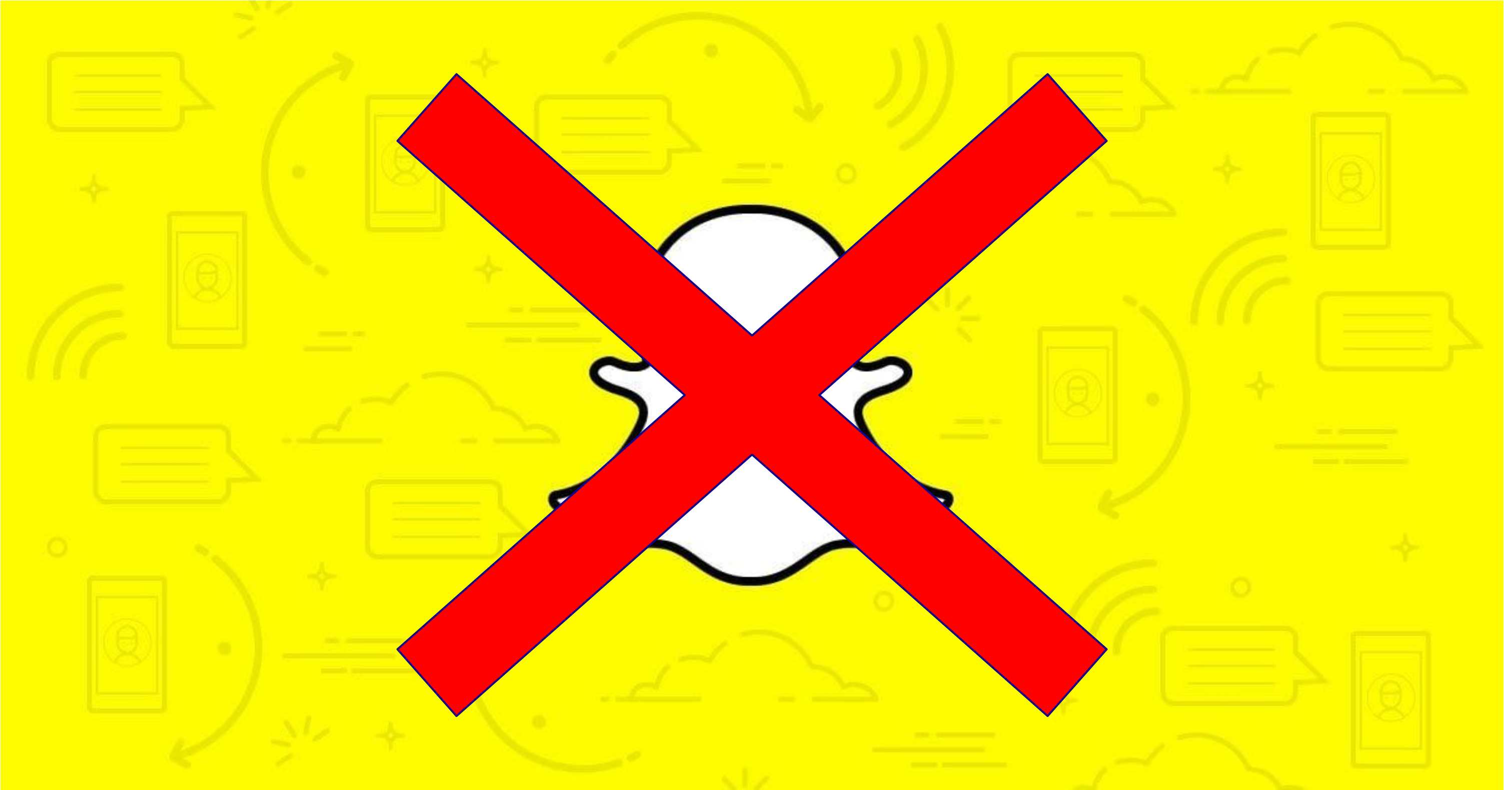
Programming

This is *not* a programming courses

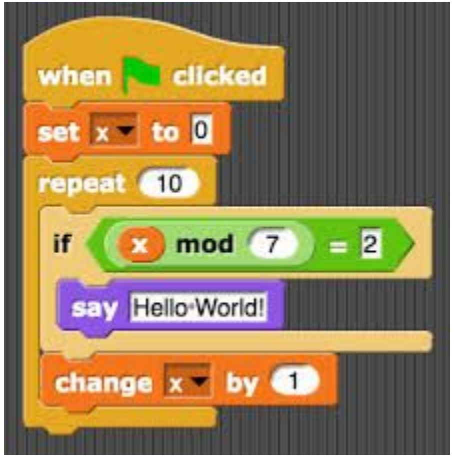
But you do need to *understand* how programs work

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

Snap!



✓



```
when ⚑ 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?

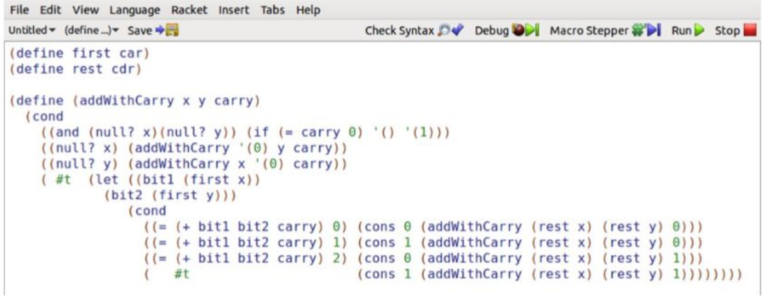
Programs are a way of encoding ***algorithms*** in a precise enough way for computers to understand the instructions.

How do programs work?

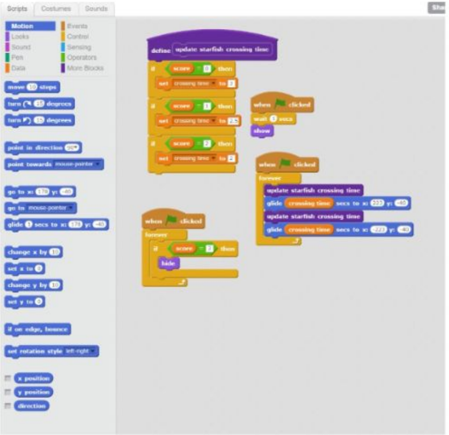
Programs are a way of encoding ***algorithms*** in a precise enough way for computers to understand the instructions.

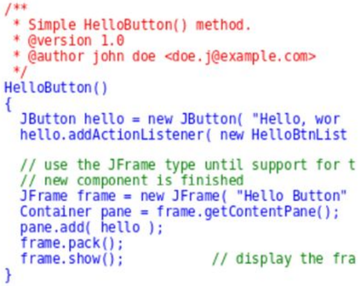
Programmers use a **high level language** like Snap, Scratch, Python, C++, Java, Racket, etc.

These languages may look very different

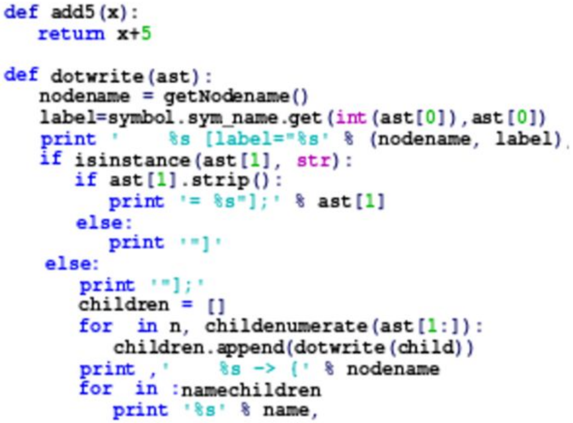


```
(define first car)
(define rest cdr)

(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))))))))
```

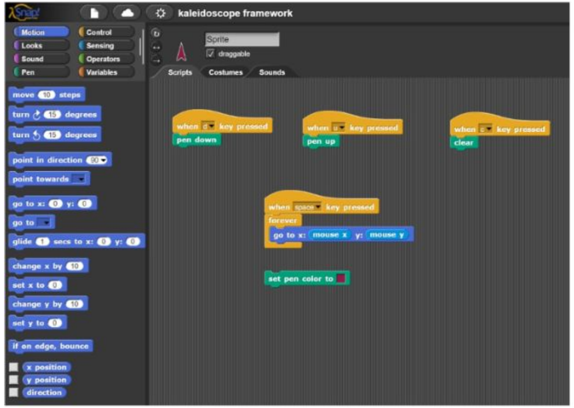


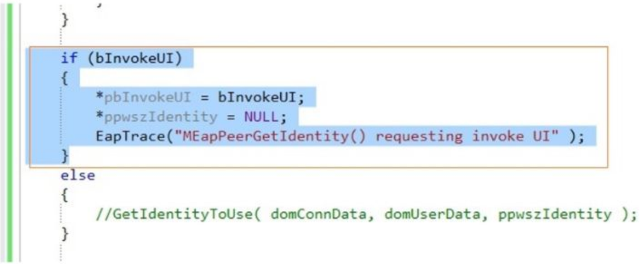


```
/**
 * Simple HelloButton() method.
 * @version 1.0
 * @author john doe <doe.j@example.com>
 */
HelloButton()
{
  JButton hello = new JButton( "Hello, wor
  hello.addActionListener( new HelloBtnList

  // 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,
```





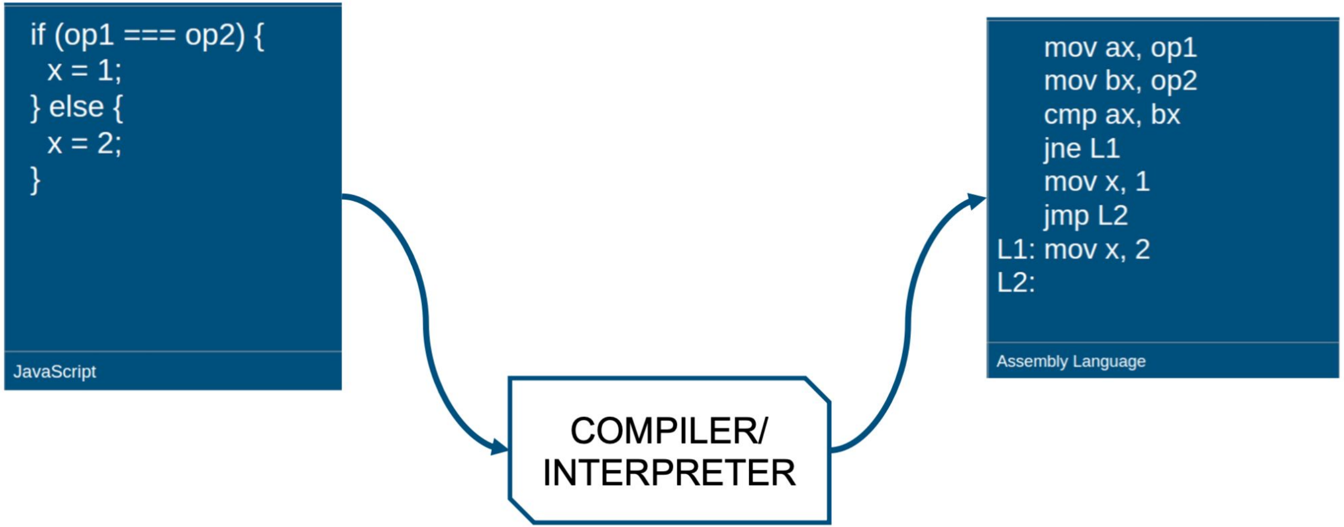
```
    }

    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

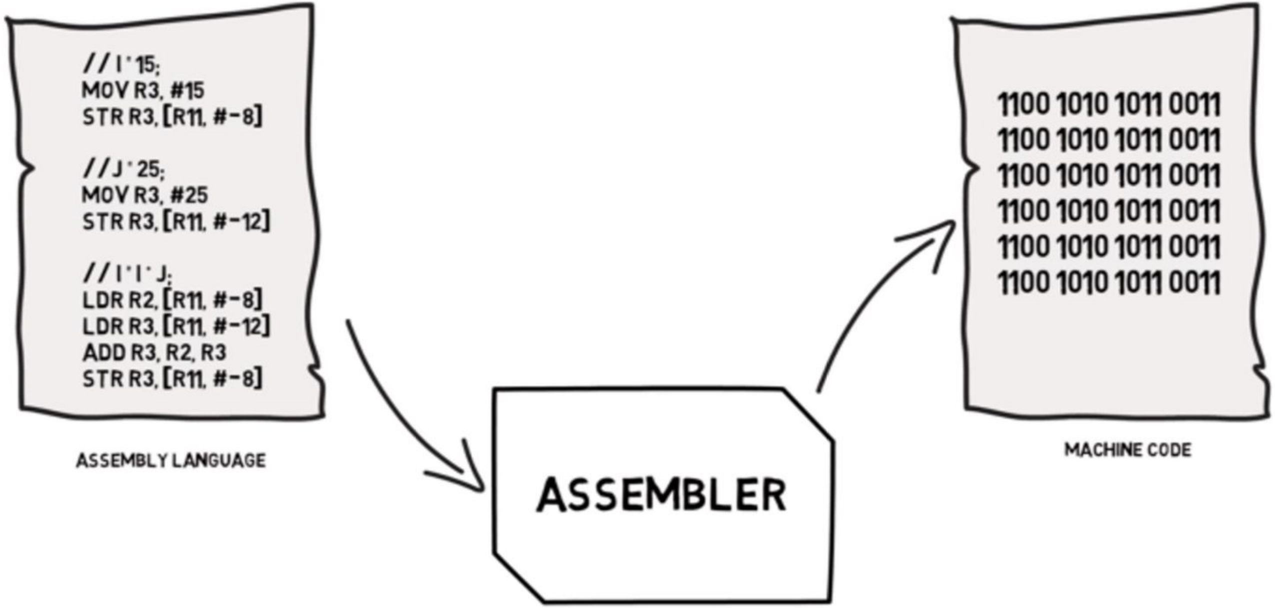
```
    mov ax, op1
    mov bx, op2
    cmp ax, bx
    jne L1
    mov x, 1
    jmp L2
L1: mov x, 2
L2:
```

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]
```

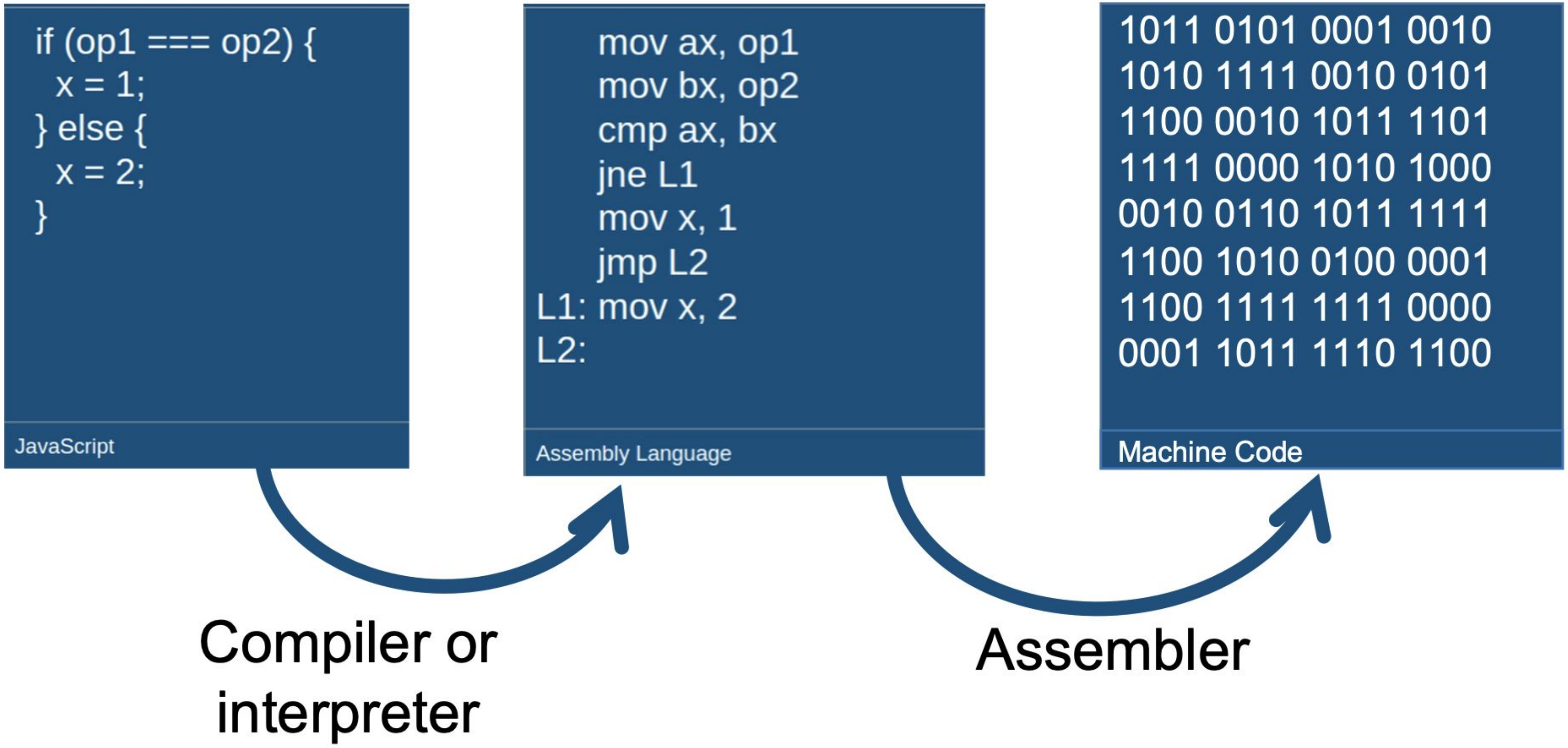
```
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
1100 1010 1011 0011
```

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

37

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.

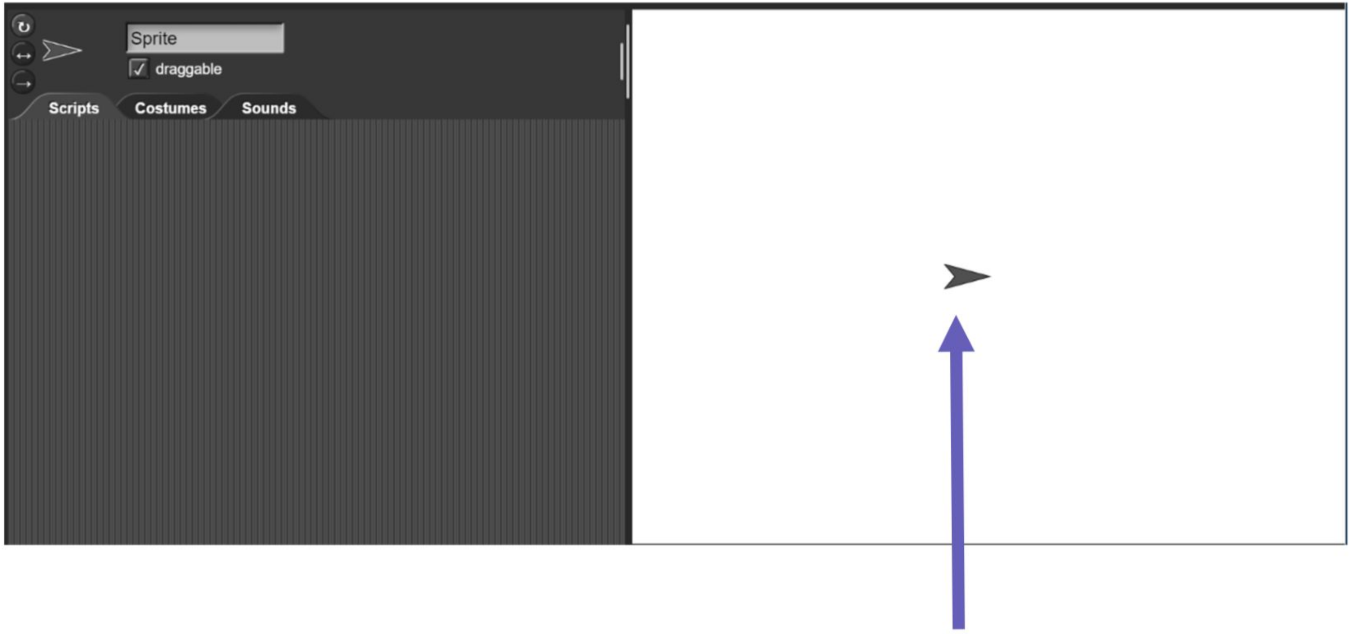
In 1966 it was proved (*structured program theorem*) that any algorithm can be made with only three "ingredients":

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.



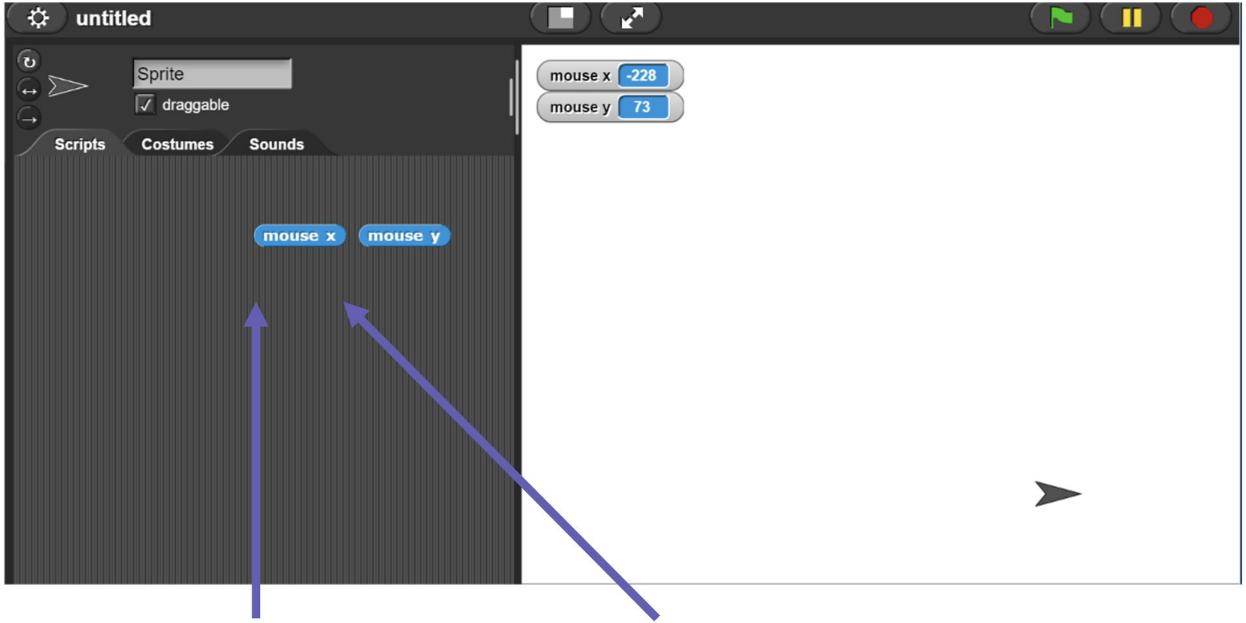
Code area (currently blank)

Sprite

Variables

Intro to Snap!

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

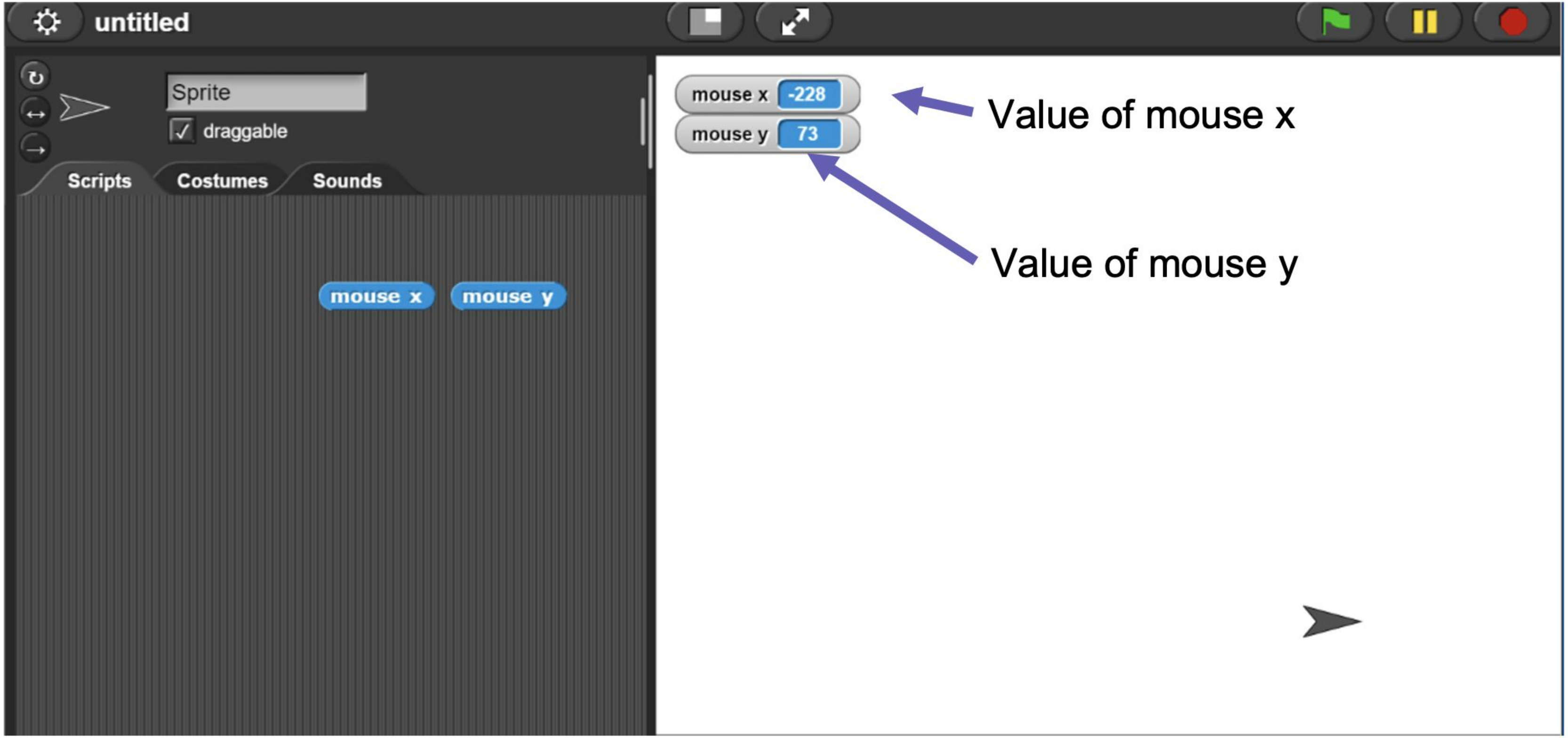


Variable: where on the x axis is the mouse?

Variable: where on the y axis is the mouse?

Intro to Snap!

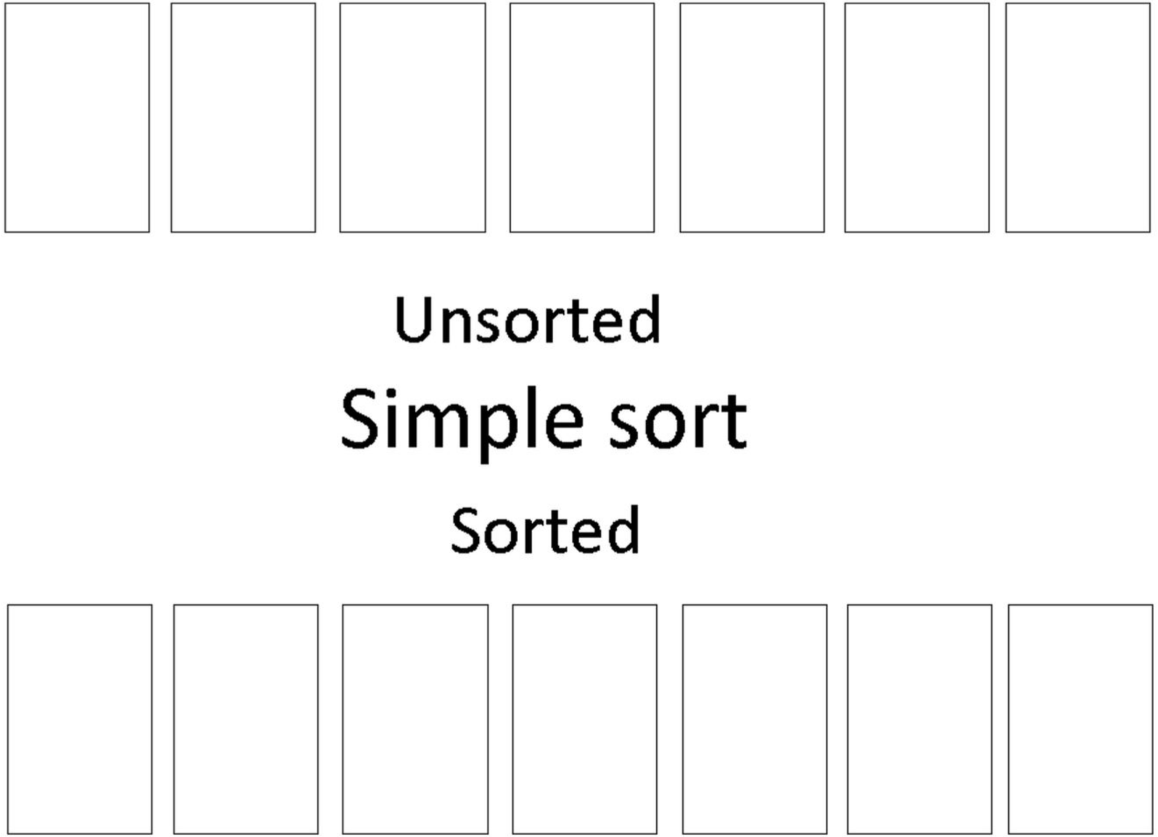
Variables have a **name** and a **value**



Value of mouse x

Value of mouse y

When we sorted cards, each slot was a variable



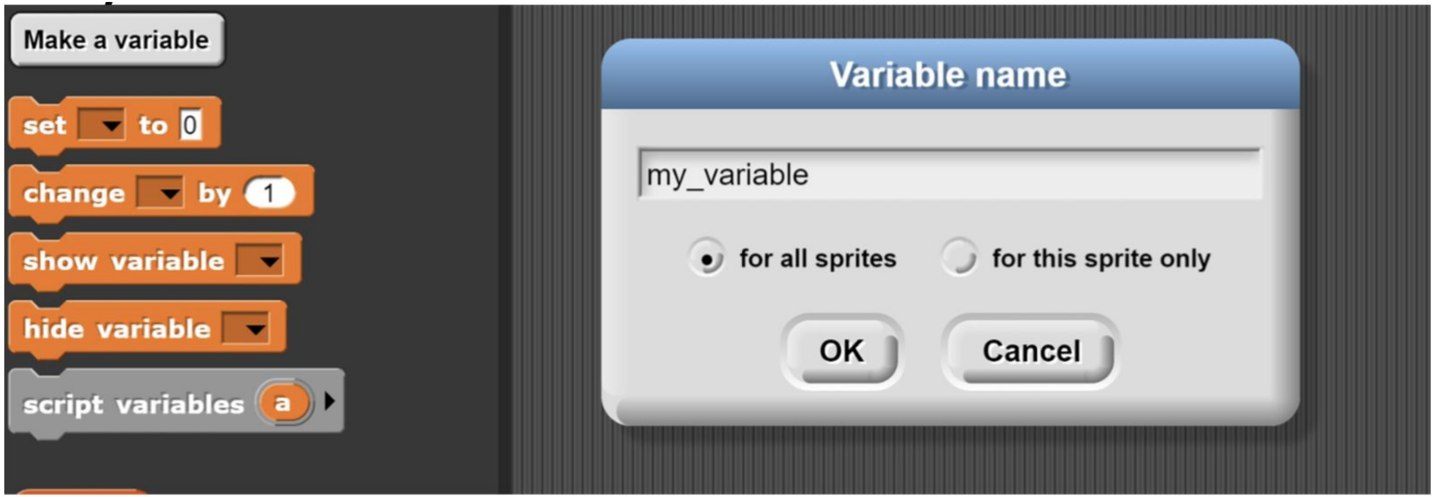
Unsorted

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:



my_variable

● for all sprites ○ for this sprite only

Variables are (usually) in orange. Things that are black writing on white are constants – (The value remains as stated)

51

iClicker

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

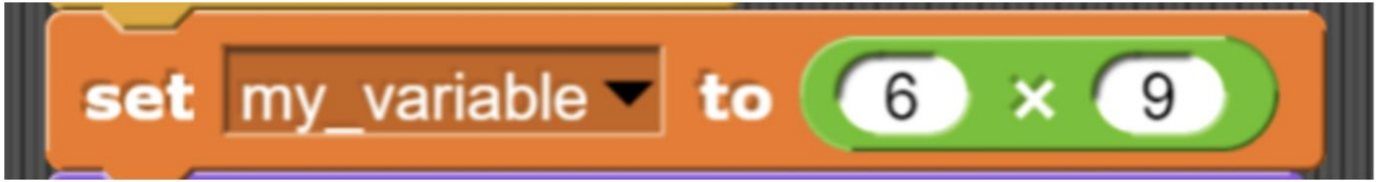
A. 6

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

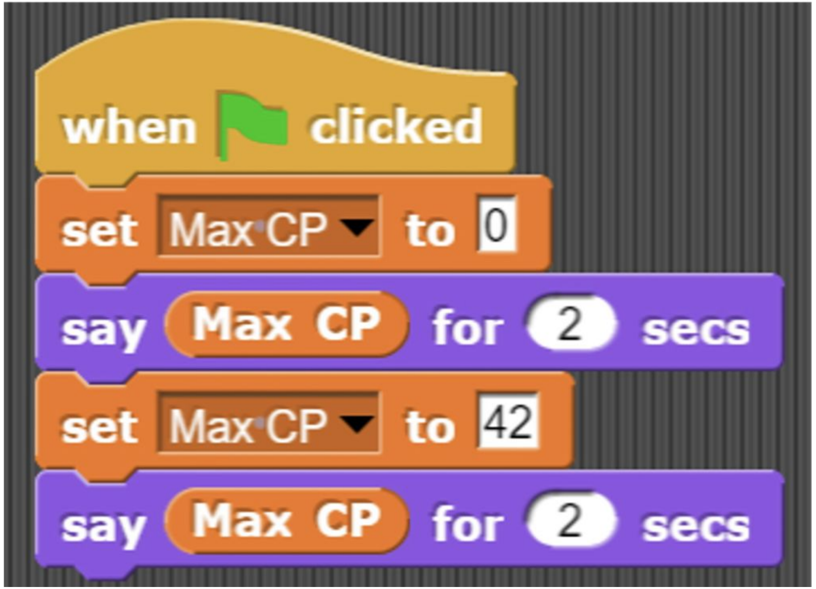
A. 0

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
```

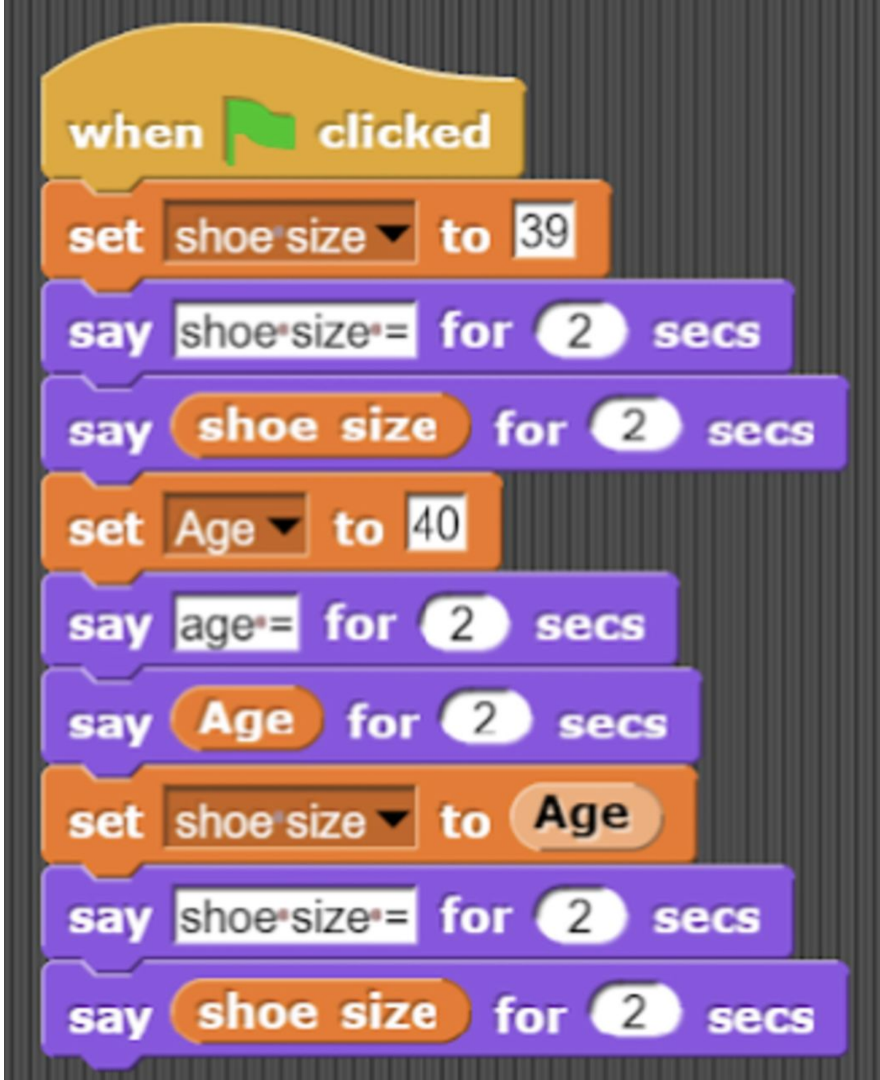
Q: What is the value of “shoe size” after all lines of the program are executed?

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?

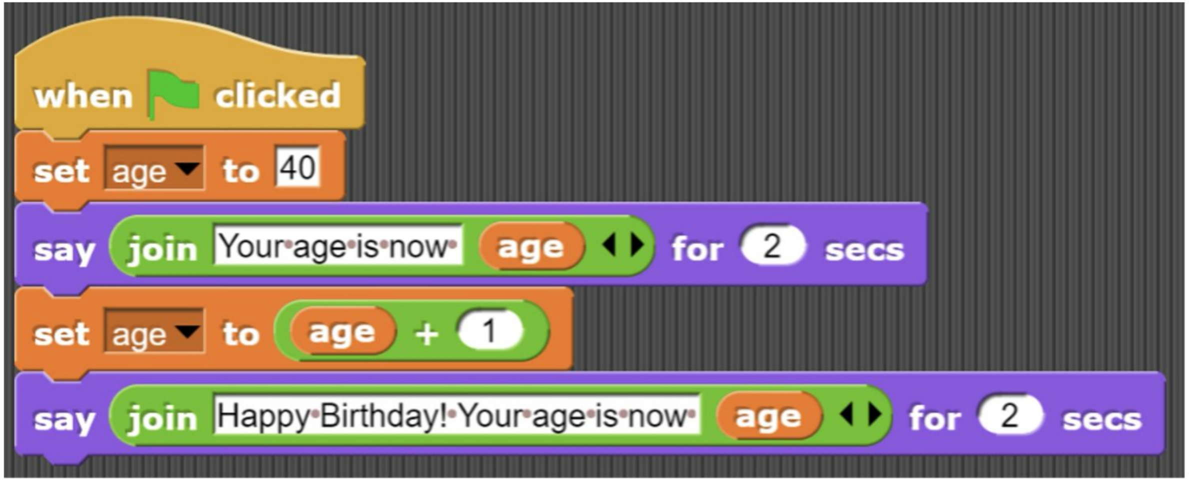
iClicker

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
```

Wrap up

Course Admin

- **Lab #3**
  - Intro to Snap!
  - Due on Thursday, Jan 30 at 11:59pm
- **Post-Class (PC) Quiz #1**
  - Questions? Contact your lab TA.
- **Post-Class (PC) Quiz #2**
  - Only 1 attempt, 60 minutes
  - Due on Sunday, Feb 2 at 11:59pm
- **Project**
  - Milestone 1 - Proposal (5%) **Feb. 12**
  - You should have started by now! If not, start **today**.