CPSC 100
Computational Thinking

Intro to Programming

Instructor: Firas Moosvi
Department of Computer Science
University of British Columbia

Agenda

- Recap: Breaking Bad Algorithm
- Course Admin
- Learning Goals
- Intro to Snap!
- Components of an Algorithm

Recap

Breaking Bad Algorithm

Q: What is the cost to signal the letter "A"?

iClicker

A. 1
B. 2
C. 3
D. 4
E. 5

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

Example of cost counting: Letter F

2 to get to the "E" row
1 to signal the "E" row
1 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 “N”

---

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

Course
Admin

Course Admin

- **Test 3** window will start on Monday
  - Sorry! I need some more time to set up the Test! Extra apologies to the 4 of you who booked on Saturday & Sunday.

- **Project**
  - We'll start talking about the Project on Monday next week, I'll adjust the course schedule accordingly

Course Admin

- **Test 2 is being graded**, grades will hopefully be released mid-next week!

- **Lab 2** grades will be released late today!

- **Learning Log 2** is released!

Learning Goals

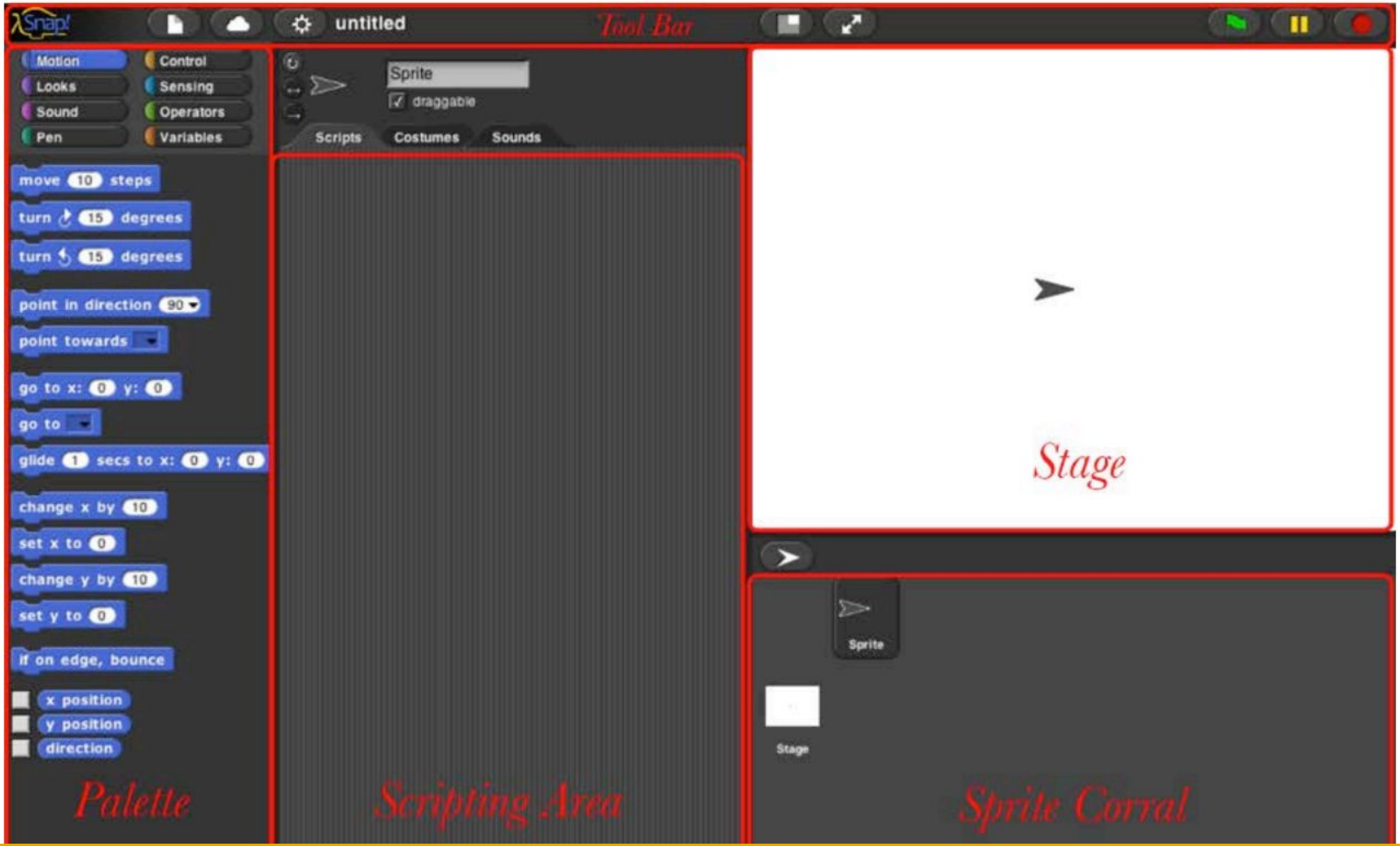
After this week's lecture, you should be able to:

- Learn about Snap! and its interface
- **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.
- **Identify and describe** the components of an algorithm
  - (i.e., sequencing, selection, and iteration)

Intro to Snap!

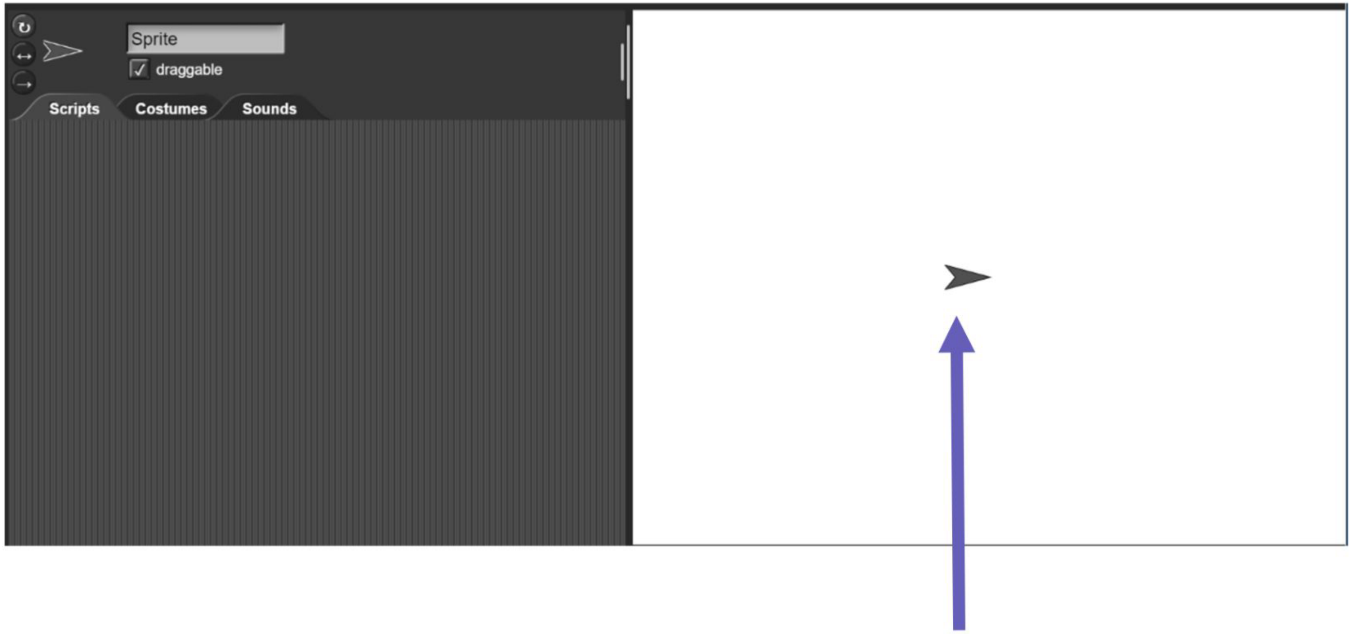
Credit to Brendan Dilloughery for the Intro Snap slides!

SNAP! Interface



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.

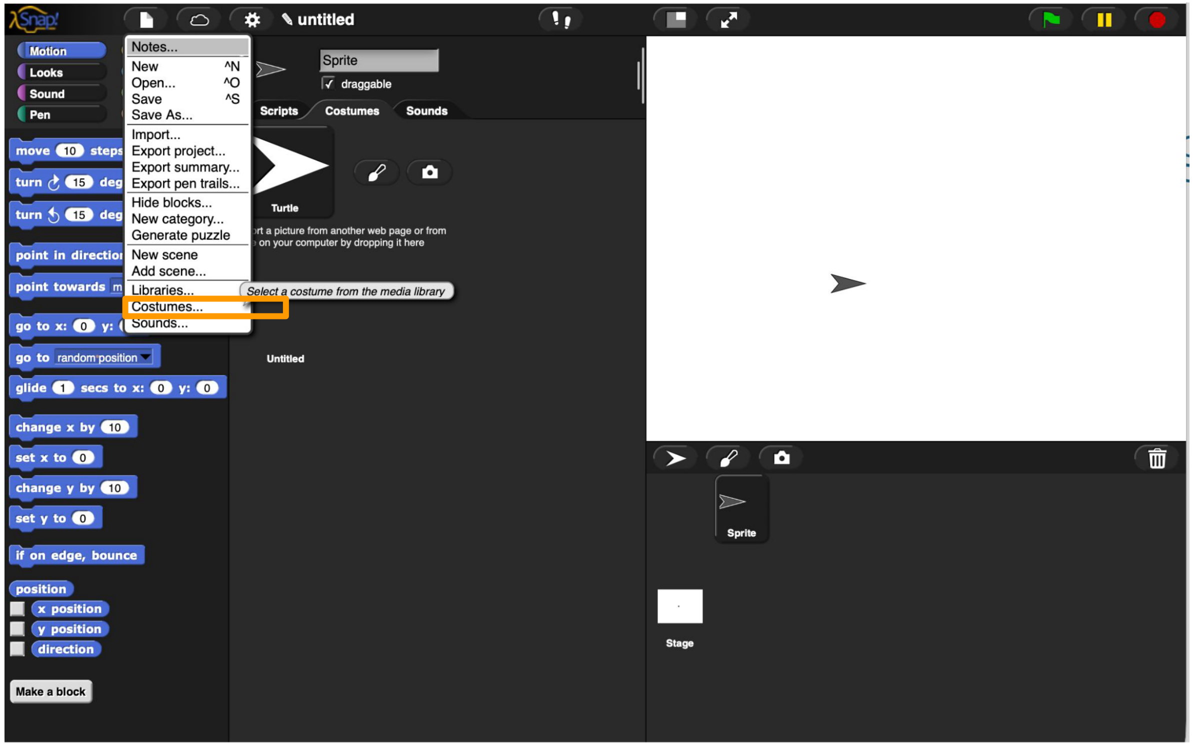


Code area (currently blank)

Sprite

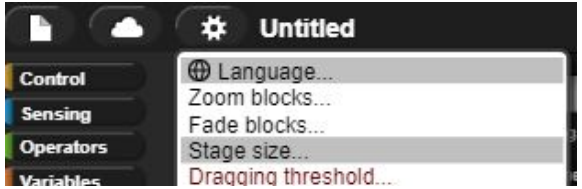
Intro to Snap!

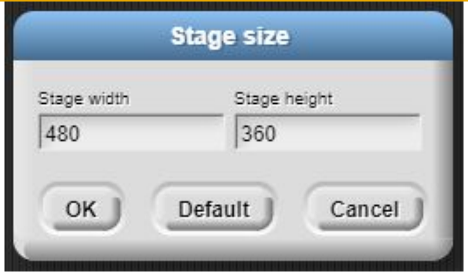
Let's change our Sprite's costume:



Stage

(aka the background)



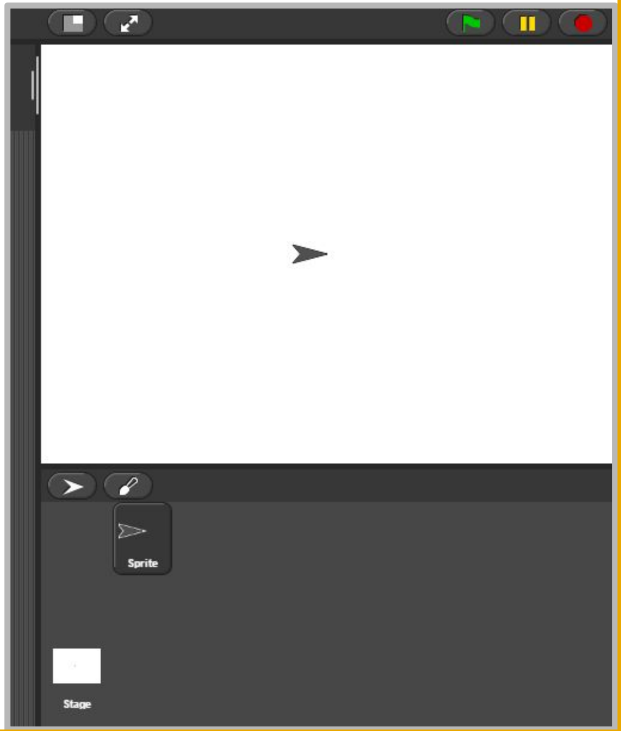


Stage width: 480
Stage height: 360

The stage/background is 480 pixels by 360 pixels.
These can be called either pixels or steps.
The stage size can be changed in settings.

The locations on the stage are just like x- and y-coordinates in math. The center of the stage is (0,0).

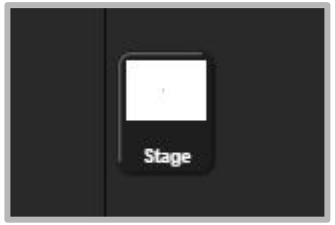
The top right corner is (240,180), the bottom right corner is (240,-180) ... and so on for each corner.

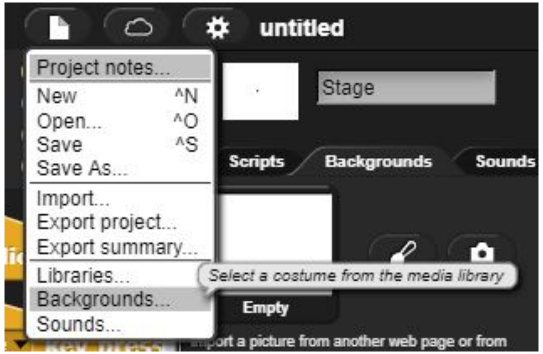


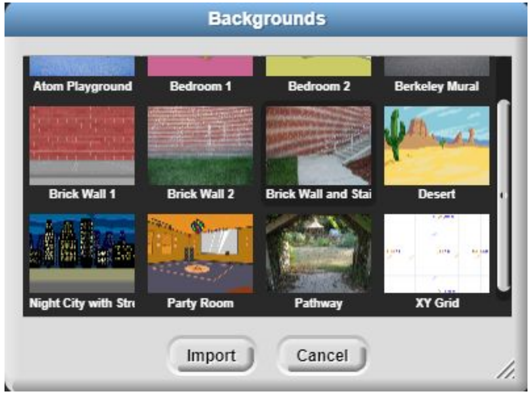
Helpful Background

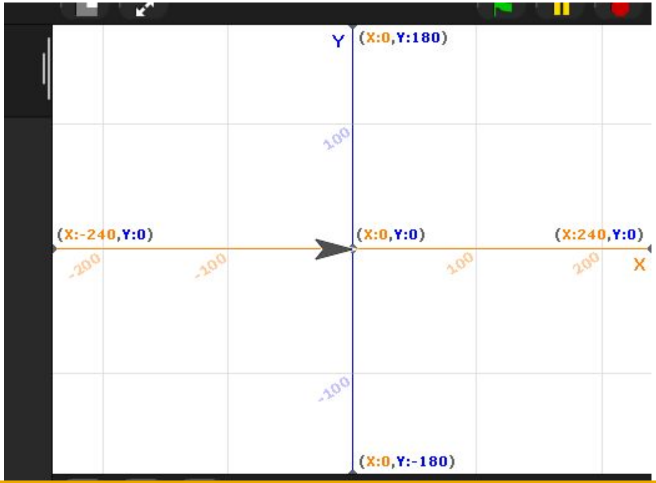
(aka the XY coordinate plane)

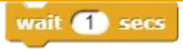
1. Select the "Stage" in the bottom right of the screen.
2. Go to the top menu, and select:
   File Menu → Backgrounds …
3. Select "XY Grid", which is the last one.



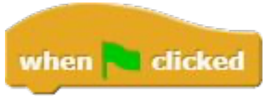


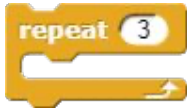


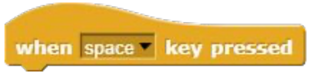




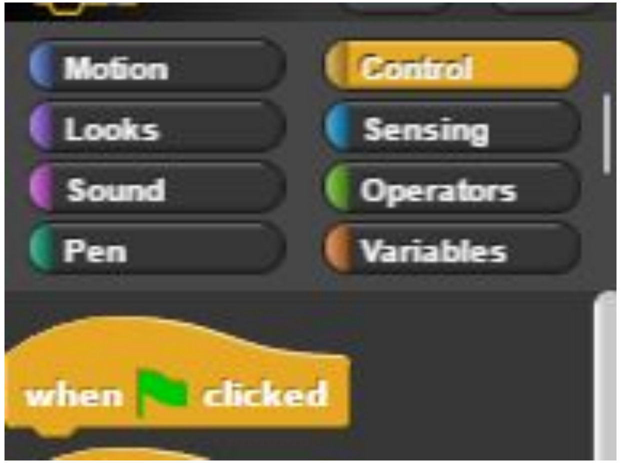
Control Blocks





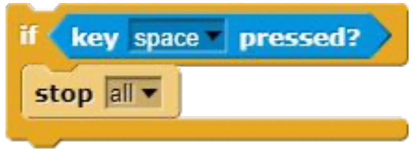


Control blocks are used to manage how your program runs. These control when portions of code -- start, stop, repeat, etc

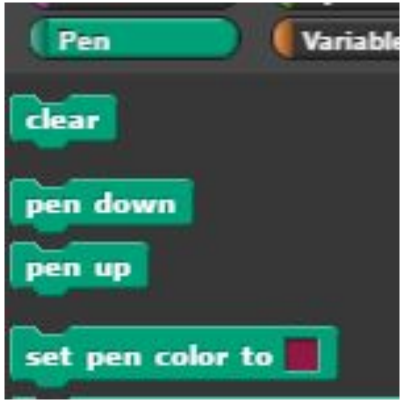


Some simple examples:

1. “When green flag is clicked, run _____”
2. “Repeat ________ 3 times”
3. “When spacebar is pressed, stop ____”



Pen Blocks



Use the Pen to draw.

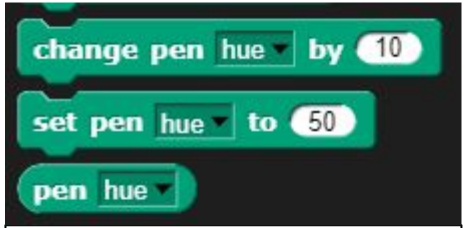
To draw with the pen, the main commands will be to:

1. Put the pen down (start writing)
2. Move (this is explained in the next slide - blue “motion” category)
3. Pick the pen up (stop writing)
4. Clear (delete everything written by the pen)

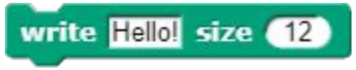
Pen Settings

You can also set the pen color (0-199) and pen size (width in pixels, 1-256).

More Pen Blocks

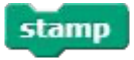


Use these blocks to set and change the pen's ...
Hue
Saturation
Brightness
Transparency



This block draws the words/numbers in the rectangular area.

Note: Your sprite is used to draw the letters. The letters will begin at your position and go to the right. Your sprite will end up to the right of the last letter.



This block draws an image of your sprite, at the current location with the pen.

Example: If your sprite is a yellow cat, it draws a yellow cat at your current position. That can't can't be changed later - only erased using the "clear" block.

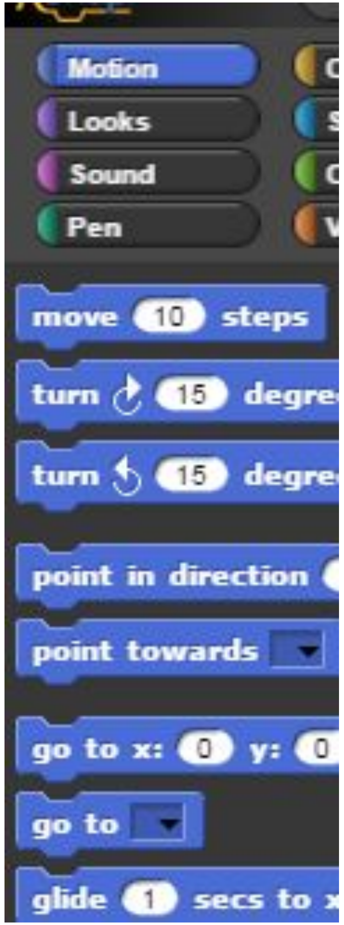
Motion Blocks

Use Motion to define your location and movements.

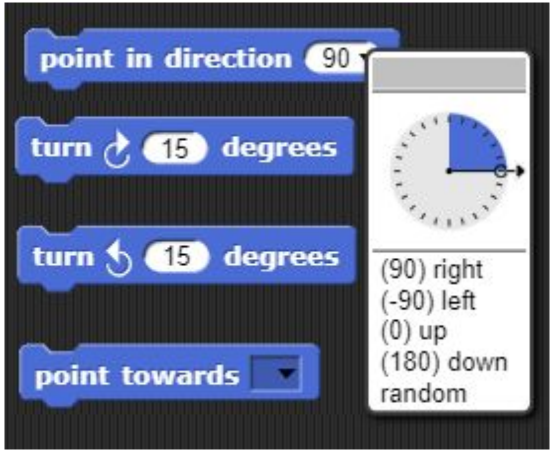
These motion blocks control the movement of sprites around the screen.

<u>As a general rule:</u>
<u>Never leave assumptions in your code!</u>
Always start with commands like “Go to (location)” and “Point in direction _____”.

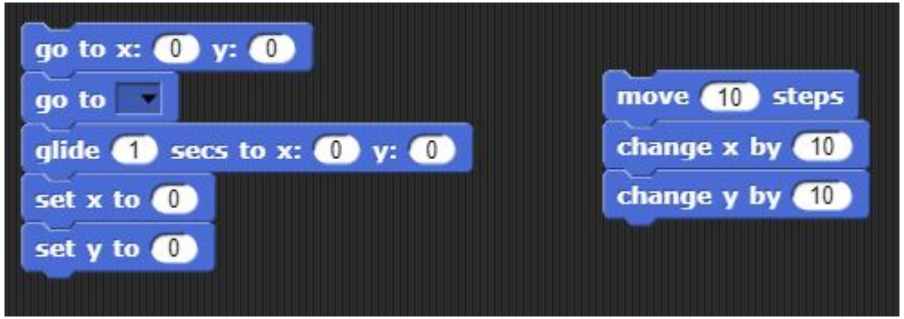
This greatly helps avoid glitches later on when you are using multiple collections of code together.



Motion Blocks



Direction - These are in degrees, with 0 pointing up. Positive degrees turn clockwise, and negative counter-clockwise.
You can also use the “Point towards ____” block to point towards another sprite.



Go / Move - “go to” or “set” blocks take you to specific locations.
“Move” and “change” blocks move you a number of pixels from your current position.

Note: The glide block locks you into that motion, and no other code in the script runs until you finish gliding.

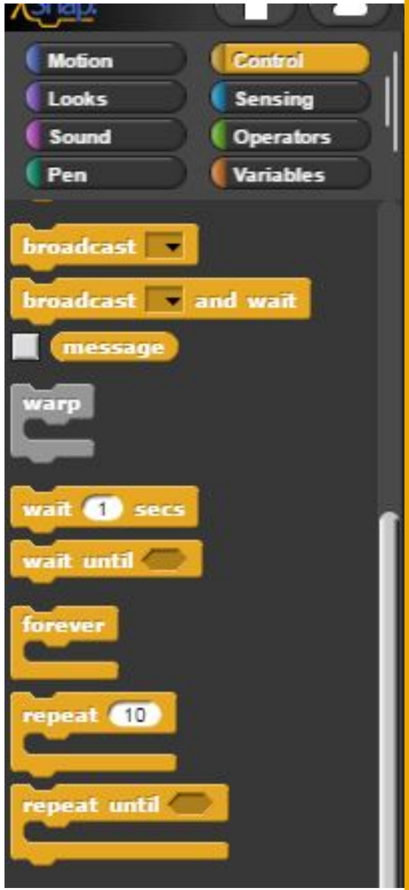
Repetition

We will use Control Blocks to repeat blocks of code.

These commands are the bread and butter of programming. They do not simply say “run ___”, but rather say when and how many times a block of code runs. It also controls when the code stops. Under what conditions? For how long?

These commands, along with clever data manipulation are what make up a lot of programming.

Otherwise, every single thing you wanted a computer to do, you would have to manually program each step. With repetition you can tell the computer … “start doing this, and keep going until you are done”.

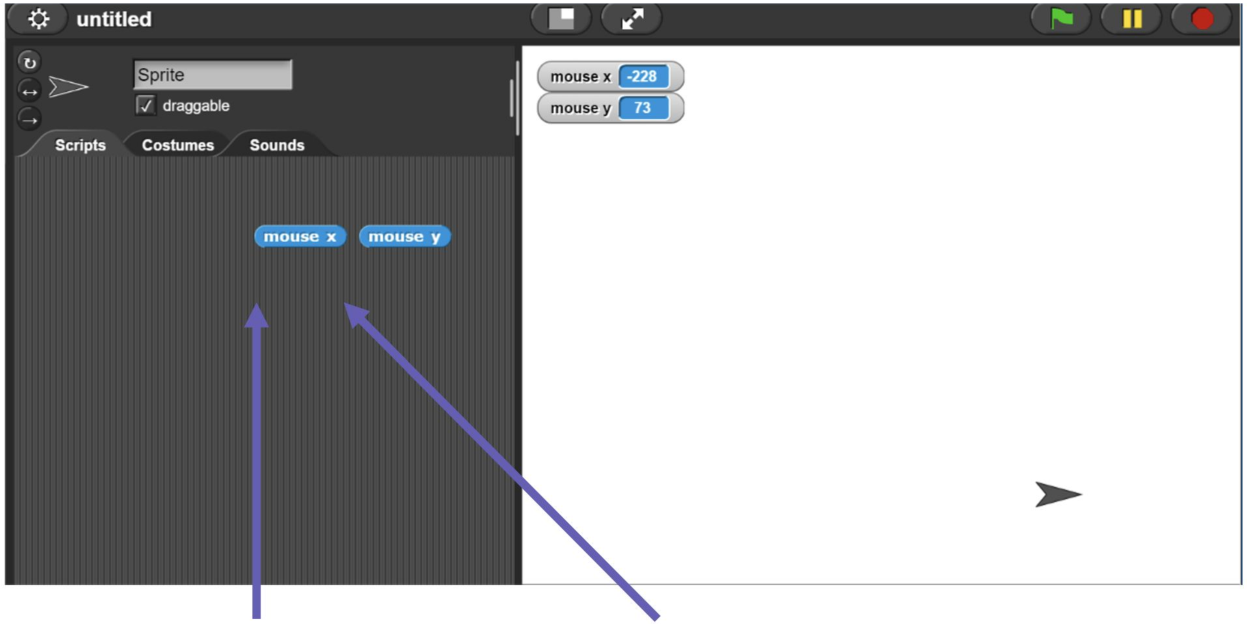


28

Variables

Intro to Snap!

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

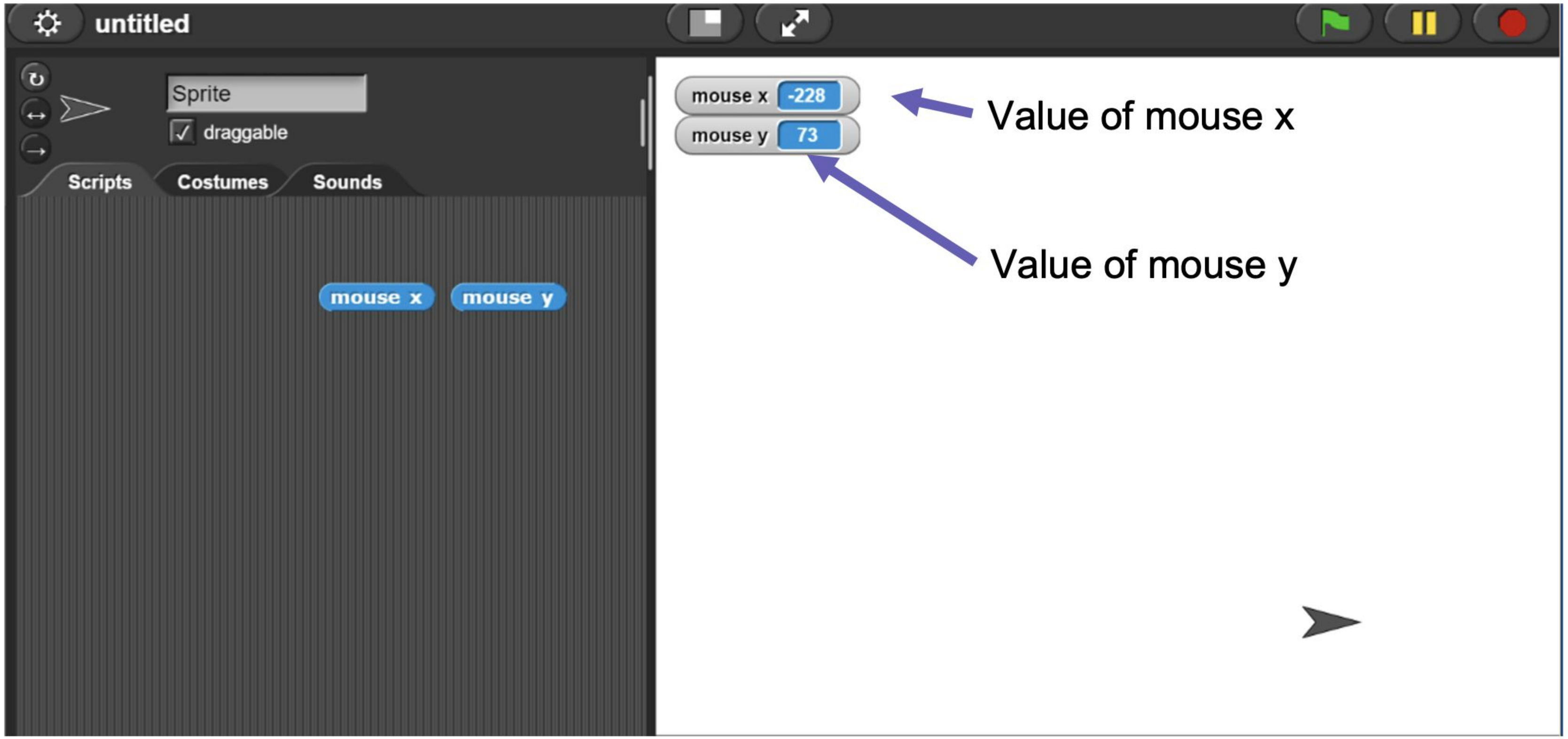


Variable: where on the x axis is the mouse?

Variable: where on the y axis is the mouse?

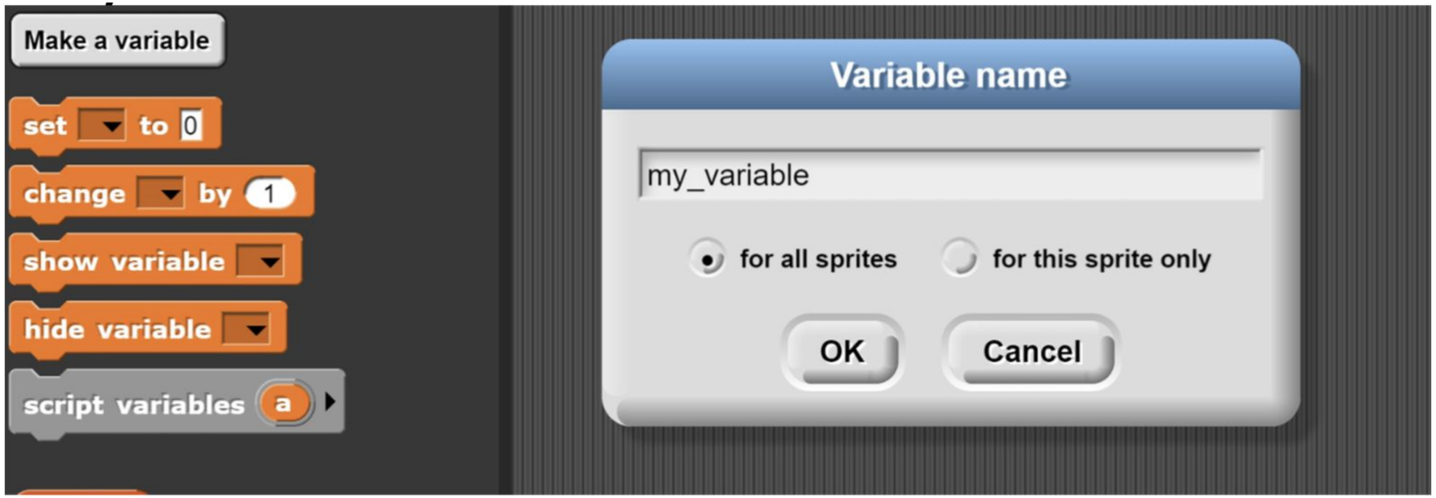
Intro to Snap!

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



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)

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

iClicker

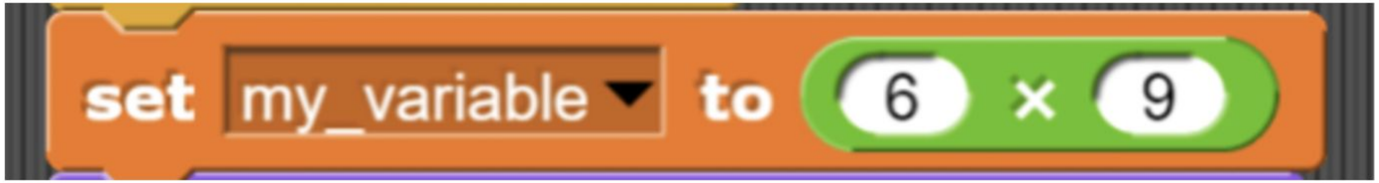
A. 6

B. 9

C. 42

D. 54

E. None of the above



```
set my_variable to (6 × 9)
```

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

iClicker

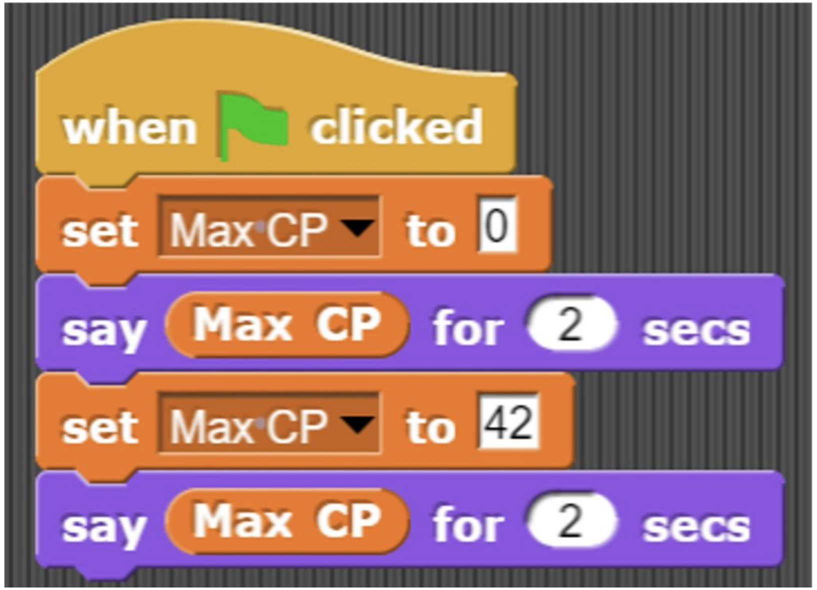
A. 0

B. 2

C. 42

D. 042

E. None of the above



```
when [flag] clicked
set Max CP to 0
say (Max CP) for 2 secs
set Max CP to 42
say (Max CP) for 2 secs
```

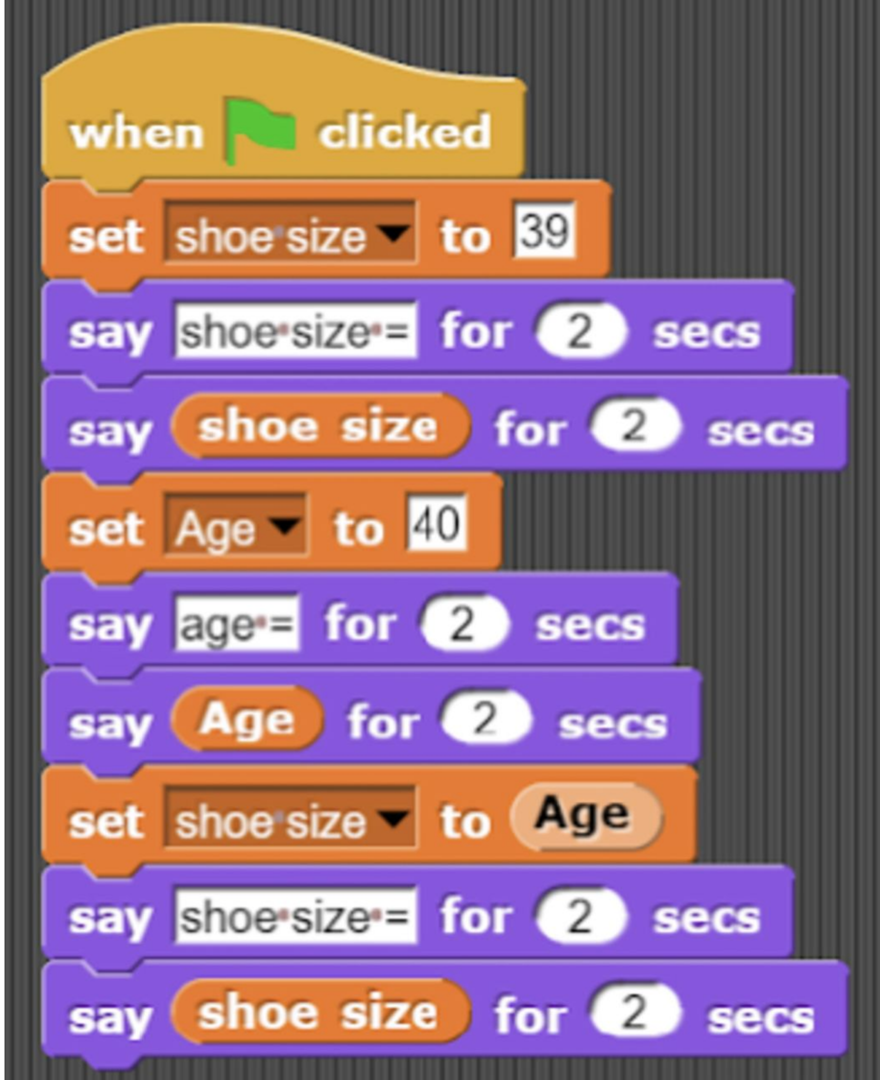
Q: What is the value of “shoe size” after all lines of the program are executed?

A. 2

B. 39

C. 40

D. 3940



```
when [flag] 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?

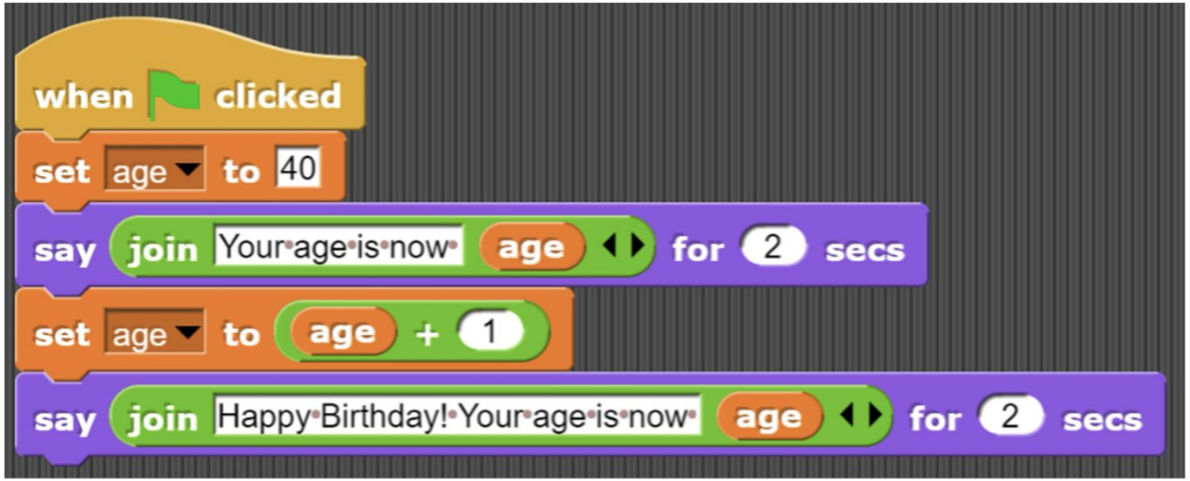
iClicker

A. 1

B. 40

C. 41

D. 401



```
when [flag] 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
```

Recall

Algorithms

Recall

Algorithms

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

Bread Making Algorithm

Components of an Algorithm - Bread Making

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

Task: In this algorithm, where do you see the three components?

Wrap up