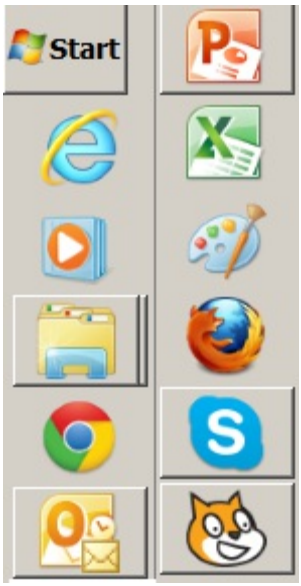
PLTW COMPUTER SCIENCE

Activity 1.1.3

Branching and Iteration

Introduction



How do computers do such amazing things? How do people write complicated content like Microsoft® Office® or Adobe® Photoshop® software? Even the most complicated software or cell phone app is comprised of very simple steps that occur one at a time.

MATERIALS

- Computer with Internet and Flash
- Scratch™ account or Scratch Offline Editor
- Camera (built-in or external)

Resources

1.1.3 sourceFiles.zip

Algorithms: Branching and Iteration in Scratch

Procedure

Scratch lets you create graphics **sprites** that move on the **stage**. Each sprite in Scratch has its own **scripts**, **costumes**, and **sounds**. The stage also has its own script, backgrounds, and sounds.

In this activity you will create a sprite costume by drawing on a photograph. Then you will write a script to make the sprite respond to the keyboard. You will learn how to use “if-else” blocks to make branches in an algorithm. You will also learn how to make a program **iterate**. To iterate means to do over and over again.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

1. Form pairs as instructed by your teacher. The professional greeting is an extremely important part of the professional skills practiced in this curriculum. No matter how/if you already greeted your partner earlier, take a moment and practice your introductory greeting with your partner. A professional greeting includes:

    - Squaring your shoulders and knees with the person you are greeting and using good posture
    - Making eye contact
    - Smiling
    - Using an enthusiastic, interested voice
    - Saying something pleasant, like “Nice to meet you,” or “I am glad to be working with you,” or “I look forward to working with you.”

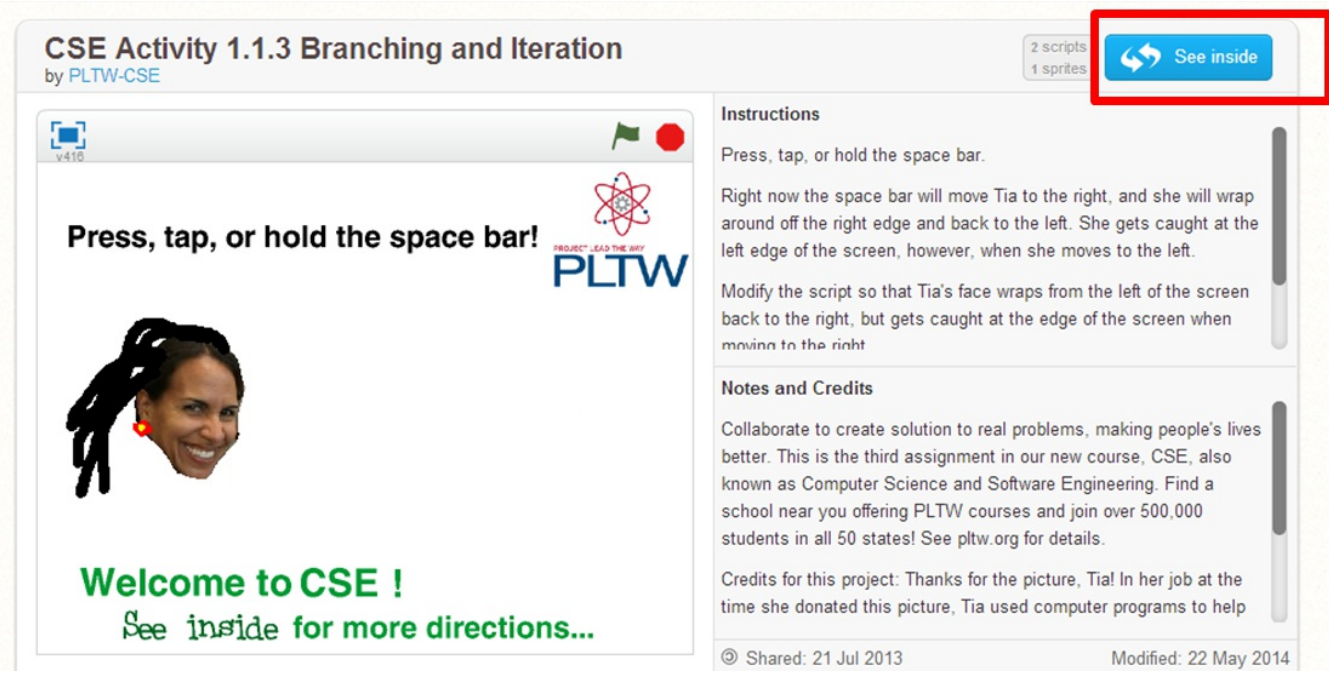
2. Each individual within the partnership should launch a Web browser. In the first part of this activity, you will each work on your own computer but are encouraged to go the same pace, helping each other along the way. Later in the activity, you will work together on one computer to **pair program**, alternating who is “driving” with the keyboard and mouse.

3. Navigate to the Scratch project at **http://scratch.mit.edu/projects/11520529/**. Click the green flag.

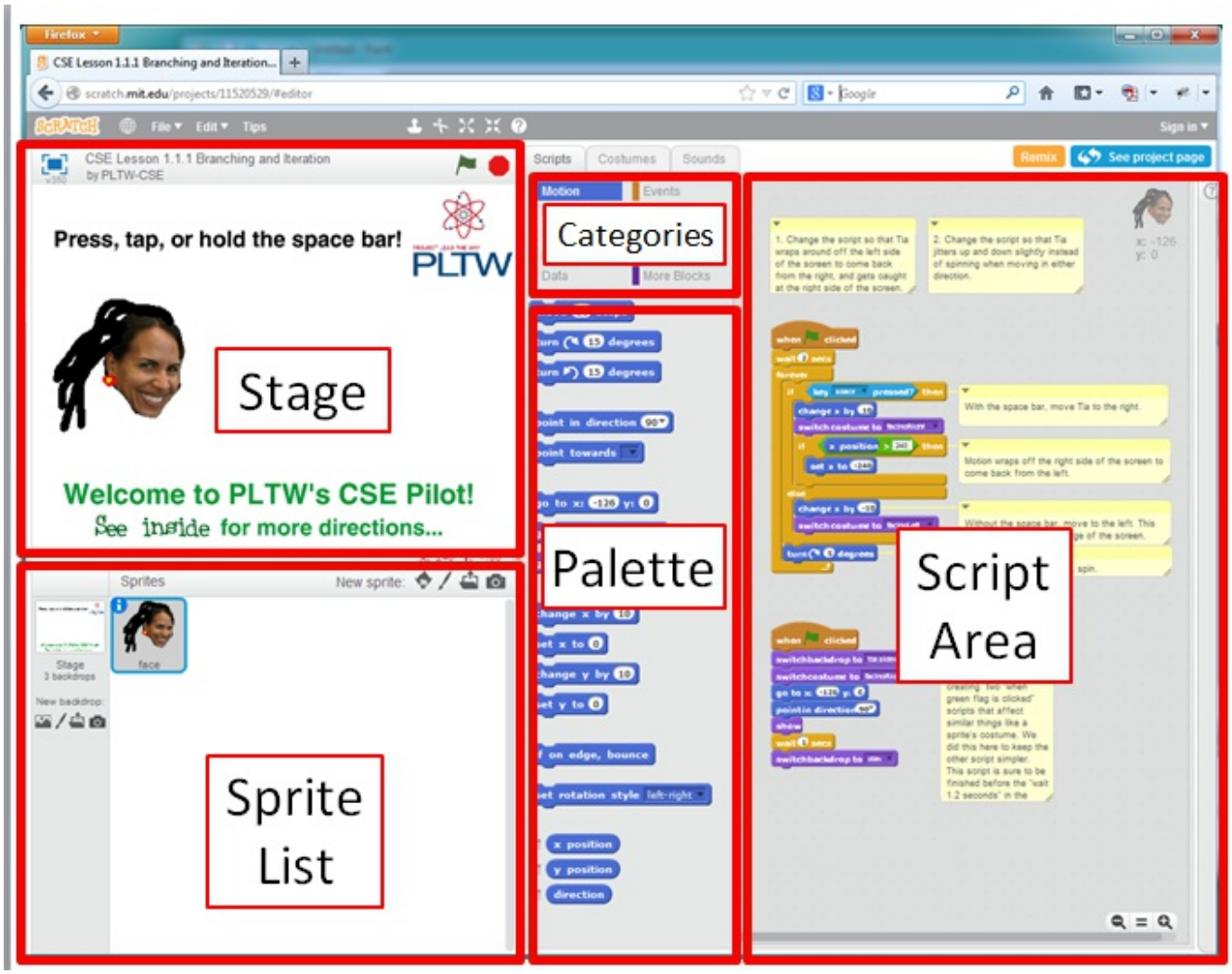
4. Try the space bar and observe the effect. Hold down the space bar and observe Tia moving

off the right side of the screen. Let go and watch the left side.

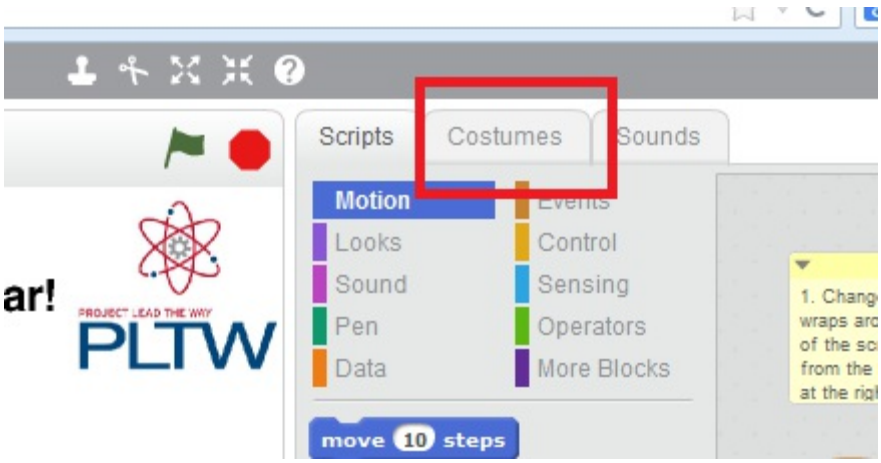
5. Scratch is a web-based Integrated Development Environment. You can create projects by remixing the work of others. Select **See Inside**.



6. Inside the project, the environment starts you out in the Scripts view. Notice these five parts of the Scripts view.

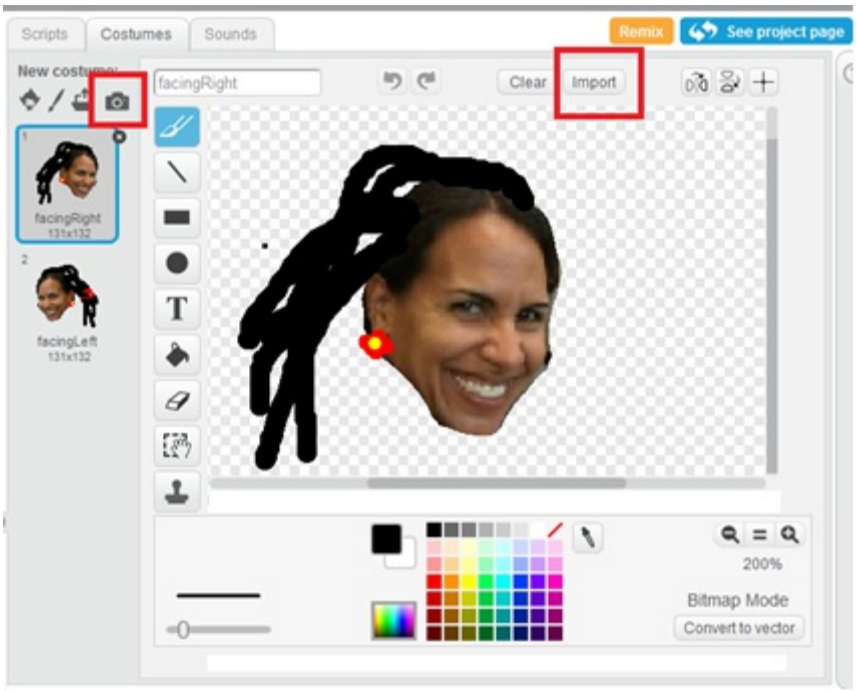


7. Inside the project you can create a picture of yourself as a new costume of this sprite or as the first costume of a new sprite. Select the **Costumes** tab as shown below to modify this sprite's costumes.

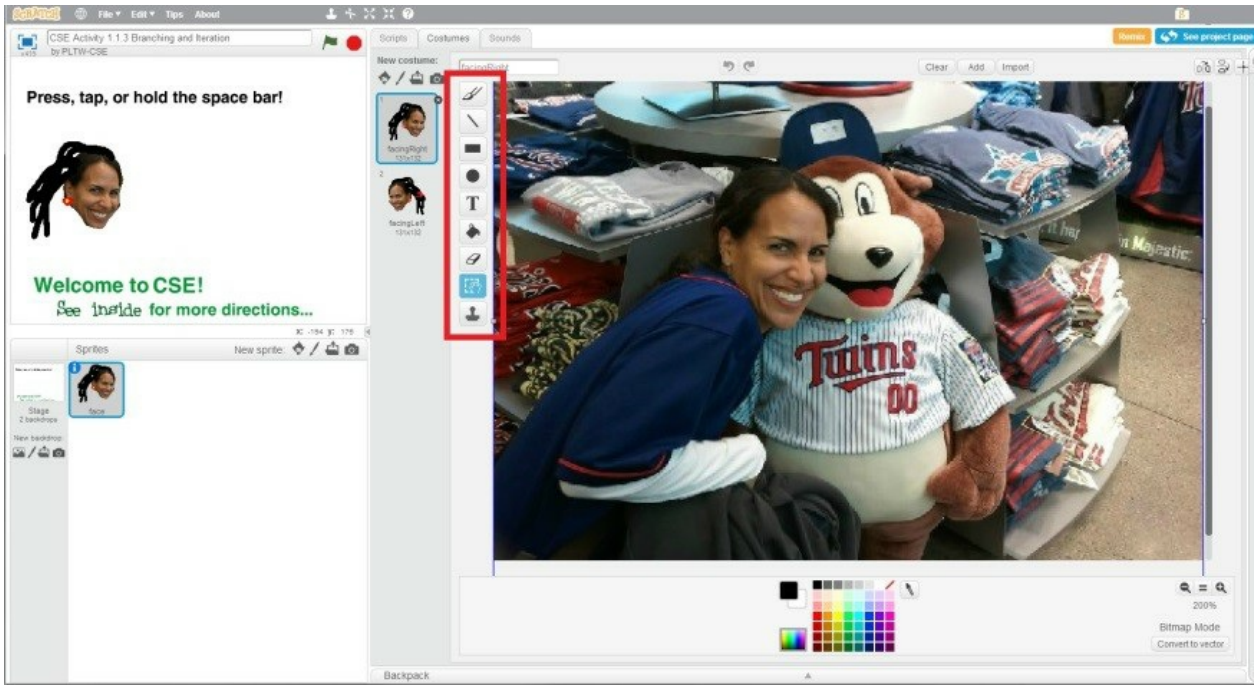


- If you have a camera attached to your computer, select the **camera** icon and **Allow** Flash Player to access your camera and microphone. When you see yourself, select **Save**.

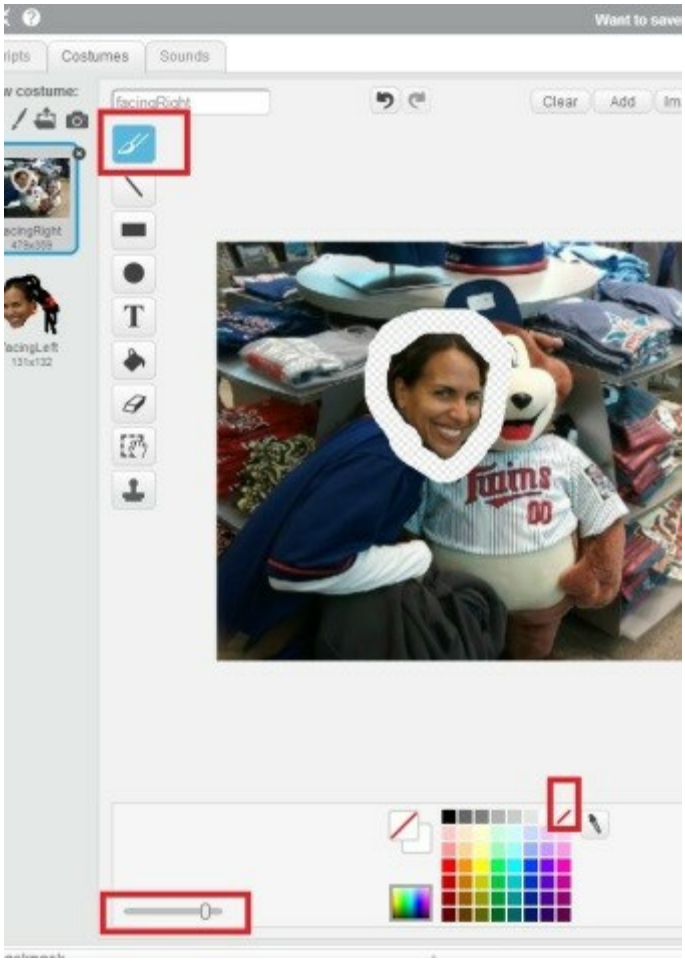
- If you don't have a camera attached to your computer, you can skip the previous step and use the **Import** button to choose an image stored on your computer instead.



- Edit the costume with the costume editor using any of the tools shown in the toolbar.

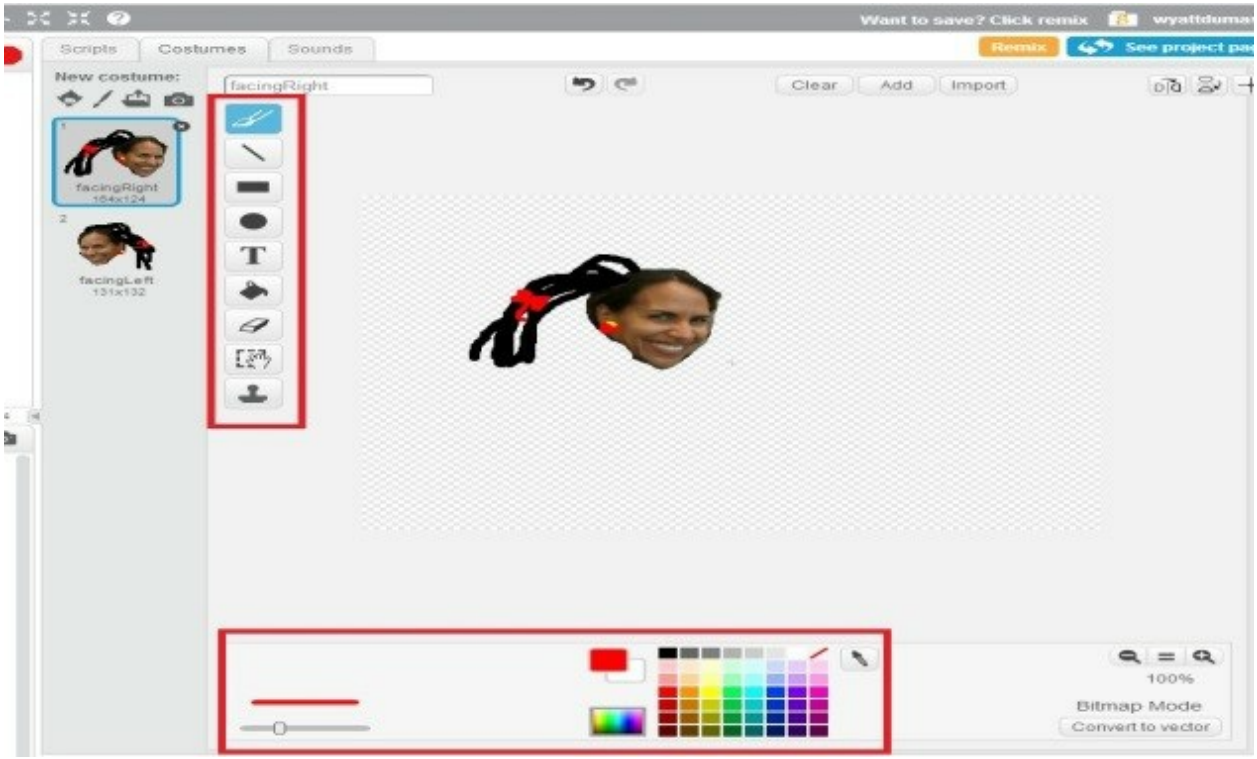


- Erase unwanted parts of the picture. Here we've picked the brush tool, changed the width of the brush, and chosen the “transparent” color.

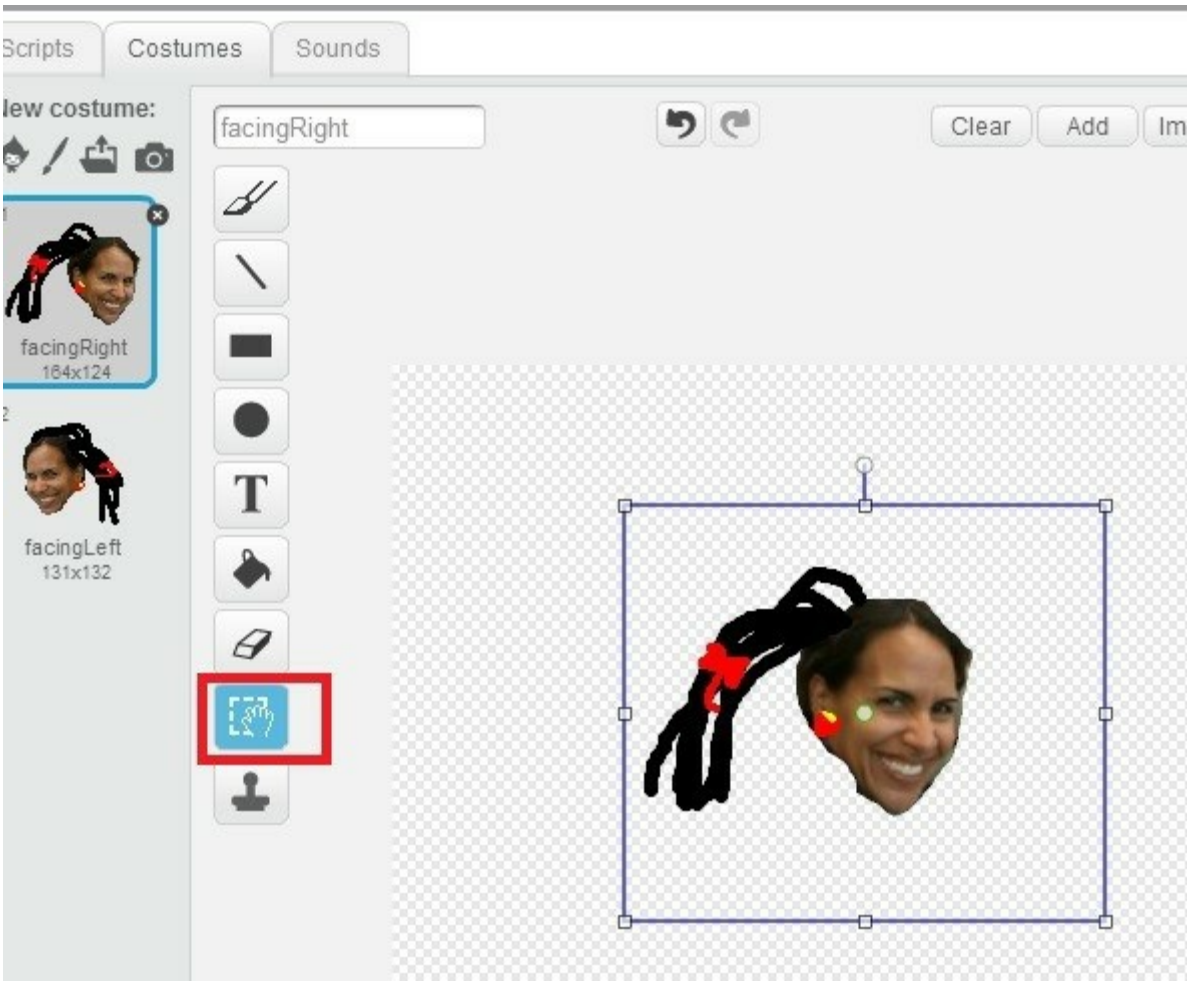


- Once you have removed the background around your face, draw as you wish on the

picture.

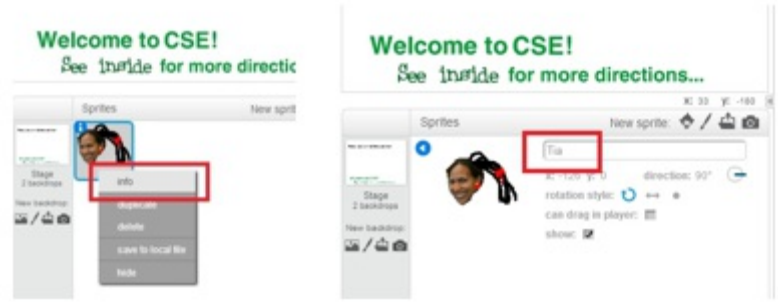


- The pixel at the center of this image will be used for the (x, y) coordinate of the entire sprite. It is also the location that the costume will use as the center of rotation if you rotate the costume. Make sure the image is centered by using the **Select** tool and dragging the visible portion of the image into the middle of the canvas.



- Using the following instruction, make a duplicate costume facing the other way. Duplicate the costume. Reflect the costume across a vertical axis. Rename the costumes `facingLeft` and `facingRight` or something else descriptive.

- Delete the other costumes if you haven't already by right-clicking on them and selecting **Delete**.
- Name your sprite. If you modified the `face` sprite, you can rename it by right-clicking on that sprite and selecting **info**.



8. Access the documentation as follows.

    - The help screens and the reference guide can be accessed through the help menu by clicking either of the question mark icons shown below.

    - Right-click any block to obtain help with that block.

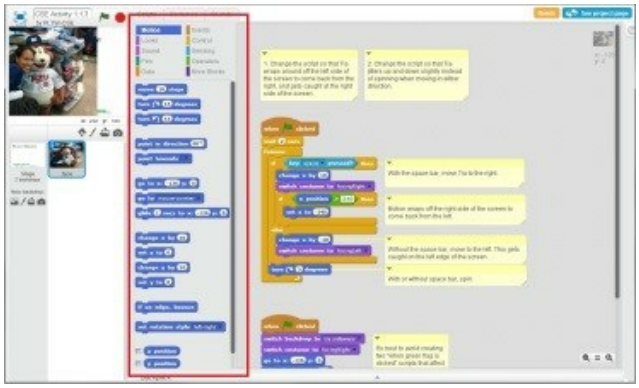
Algorithms: Branching and Iteration in Scratch

9. Create your first script! Every script belongs to one sprite or to the stage, and we've started you off with two scripts belonging to the `face` sprite. You can right-click a script or comment to delete it. Every script starts with a “cap” block and then proceeds one block at a time.

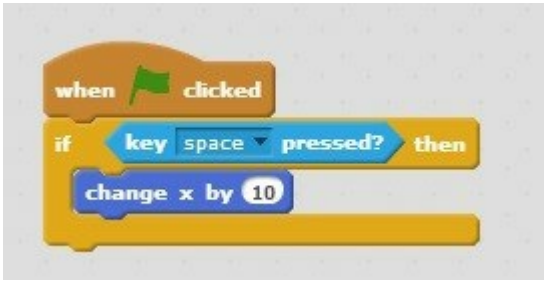
    - In the Scripts tab for your new sprite, drag blocks from the palette to the scripts area as shown here to create the script shown. Since each block is color coded, you can always determine the category of any block shown. To delete a block, drag it back to anywhere on the palette. Test a script by double clicking it or by creating the **event** described by the cap block (here, clicking the green flag).

      Always program a little at a time, testing frequently. Even professional programmers make lots of mistakes, so they test frequently to catch errors before they pile up and become difficult to find.

    - The hexagon-shaped blocks are called **Boolean expressions**. They are either true or false. An **if then** structure evaluates a Boolean expression once. If the expression is true, the script **executes** the blocks inside the `if then` block. If the expression is false, the `if then` block skips the blocks inside the `if then` block. Either true or false, the `if then` block only happens once. Then the script goes on to the next block.



`face` sprite:

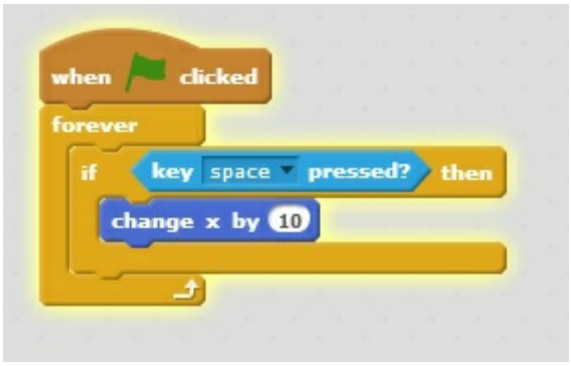


Try holding down the space bar while clicking the green flag. What happens and why?

- Try clicking the green flag and then pressing the space bar. What happens and why?

10. Create the script shown using a `forever` loop. You can pull blocks apart by dragging the lower block away.

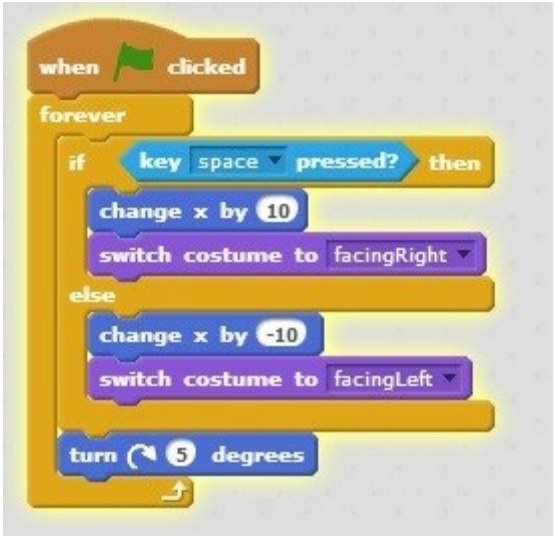
    `face` sprite:



    Try clicking the green flag and then repeatedly pressing or holding the space bar. What happens and why?

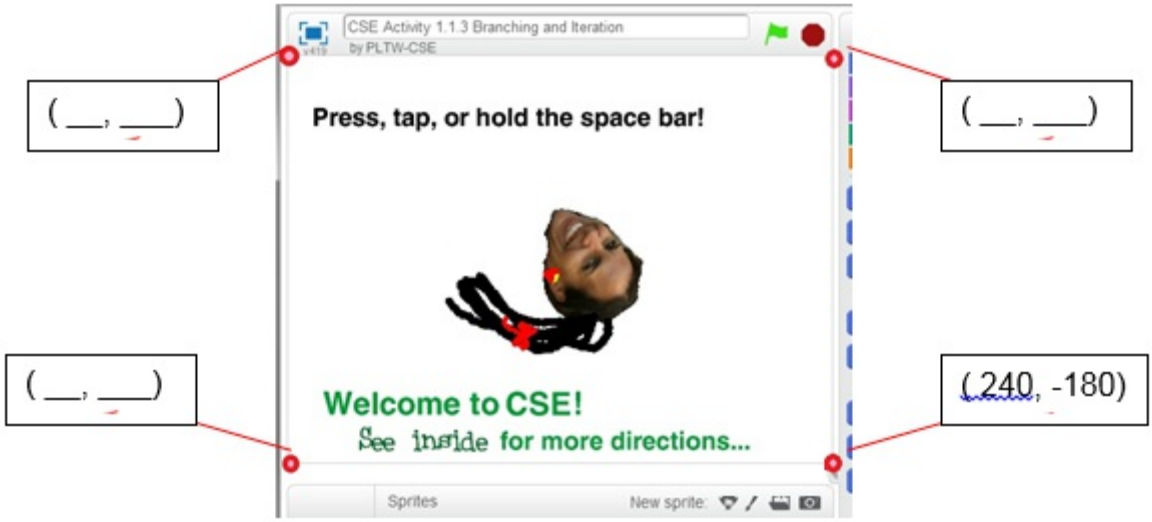
11. Create the script shown. Replace your `if then` block with the `if then else` block as shown.

    `face` sprite:



What happens and why?

12. Note the coordinates of the mouse pointer on the stage. Explore to determine the x- and y-coordinates of the left, right, bottom, and top of the stage. Record the coordinates of the corners of the stage here.



13. Add blocks to make the sprite “wrap around” when moving to the right, as shown.

- Use the greater than operator.

- Fill in the two blanks in the greater than operator so that if the sprite's x-position is to the right of the stage's right edge, the sprite goes to the stage left edge. Use either of these two options:

- Think, discuss, and then write: Why do both of the scripts above perform a “wrap

`face` sprite:

`face` sprite:

OR

around" motion to the right?

- Would it work to put the new piece (boxed in red) inside the "else" part of the `if then else` block? Why or why not? If it would not wrap motion to the right, what would happen instead?

- This is a more challenging but important question. Discuss it with your partner carefully and write down your thoughts. One of the two scripts shown above makes the computer "think" about (evaluate) the Boolean expression `x-position>240` on occasions when it really doesn't need to. That version of the script is slower than the other one. Which script is slower? When is it slower?

14. Working in pairs, modify your script so that the sprite wraps around when it reaches the left side of the screen.

- One partner in your pair should stop working at their computer and become the "navigator." The navigator will look at the screen while their partner "drives." The driver controls the mouse and keyboard. The navigator thinks about how the program will work, advising and coaching the driver. The navigator can point at their partner's screen but should not touch the keyboard or mouse. The driver should stop and listen to the navigator. You can flip a coin to decide who gets to navigate first. Switch every 10-15 minutes. Together, modify both of your programs.

- Once you've changed your code a little bit, test it. Usually it won't do what you think, and you have to figure out why it is doing what it is doing. That is a **bug**. Fixing the parts of a program that don't work right is **debugging**. Even professional programmers write programs with a lot of bugs and spend a lot of their time debugging. It can be fun! The more frustrating it is to find and understand the bug, the more satisfying it is when you figure it out - like a puzzle!

15. Save your program as “1.1.3*yourName*” in a folder specifically for your work, as instructed by your teacher. Each partner should save their own copy. To turn the assignment in, take a screen shot of your script and paste it here.

Conclusion

1. Why is it a good idea to write programs in small pieces and check how well they work after each piece is written?

2. Professional programmers usually have to write a piece of a program several times before they get it right. Think of other tasks in life that require you to figure things out. How is programming similar or different?

3. Reflect on how well you worked together when pair programming. What could you do next time to make a partnership function even better?

4. The introduction to this activity said, “Even the most complicated software or cell phone app is made out of very simple steps that happen one at a time.” That is not quite true. Each **central processing unit core** (CPU core) can process one instruction at a time. A quad-core computer processes four instructions at once. But any computer can run more programs “at once” than the number of processor cores it has. How do you think a dual-core computer, for instance, can make it look and feel like it is running ten or more programs at once?

Algorithms: Branching and Iteration in Scratch

The movie and the slide show contain the same material: Choose which media you prefer to learn about branching and iteration.

> Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

> Refer to your downloadable resources for this video.