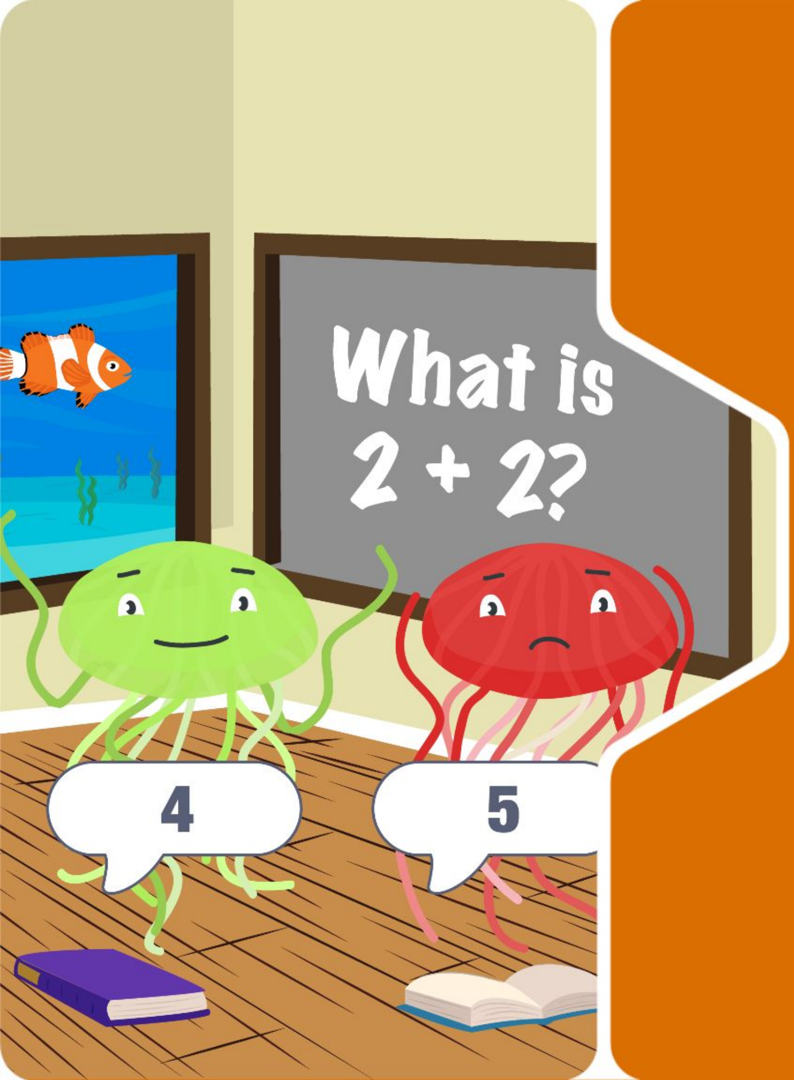
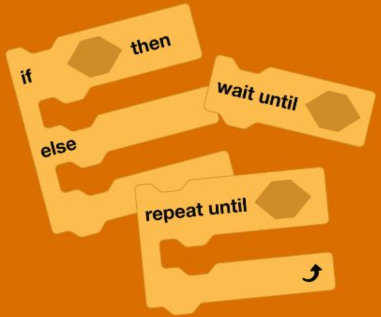
LEVEL UP WITH SCRATCH WORKSHOP SERIES

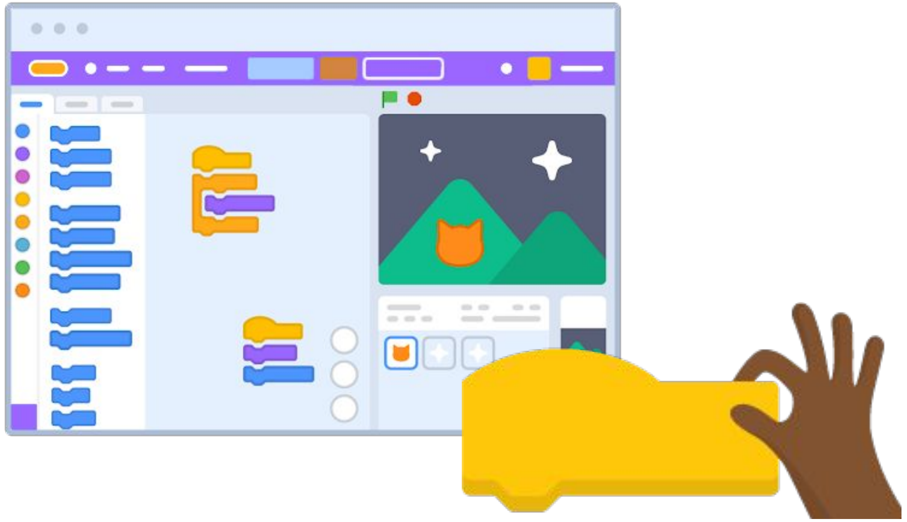
Create Dynamic Scratch Projects with the Power of Conditional Statements





Session Overview

- Brief Introduction to Creative Learning with Scratch
- Static vs Dynamic
- Conditional Statements
- A-MAZE-ing Conditions
- Compare, Contrast
- Create a Math Game
- Use Operators
- Level Up the Math Game
- Conditional Storytelling
- The Unexpected
- Nesting Statements
- Sequence Matters
- Get Help Setting Your Condition
- Your Face as the Controller
- Wrap Up - Debugging and Reflection



Facilitator: Maren Vernon

Scratch Learning Resource Designer
@algorithmar and @scratchlycaterton

SCRATCH™ FOUNDATION

Learning Goals

- Identify good use-cases for conditional statements in Scratch projects
- Hypothesize, experiment, and observe the differences between conditional statement blocks
- Debug unexpected user behavior and sequence order
- Remix our starter projects and personalize
- Reflect on finalized projects and the creative process with peers
- Communicate and share projects with your learning community and the greater Scratch online community

Getting Started

Click “Create” or log in to your free account to save projects.

go to: scratch.mit.edu

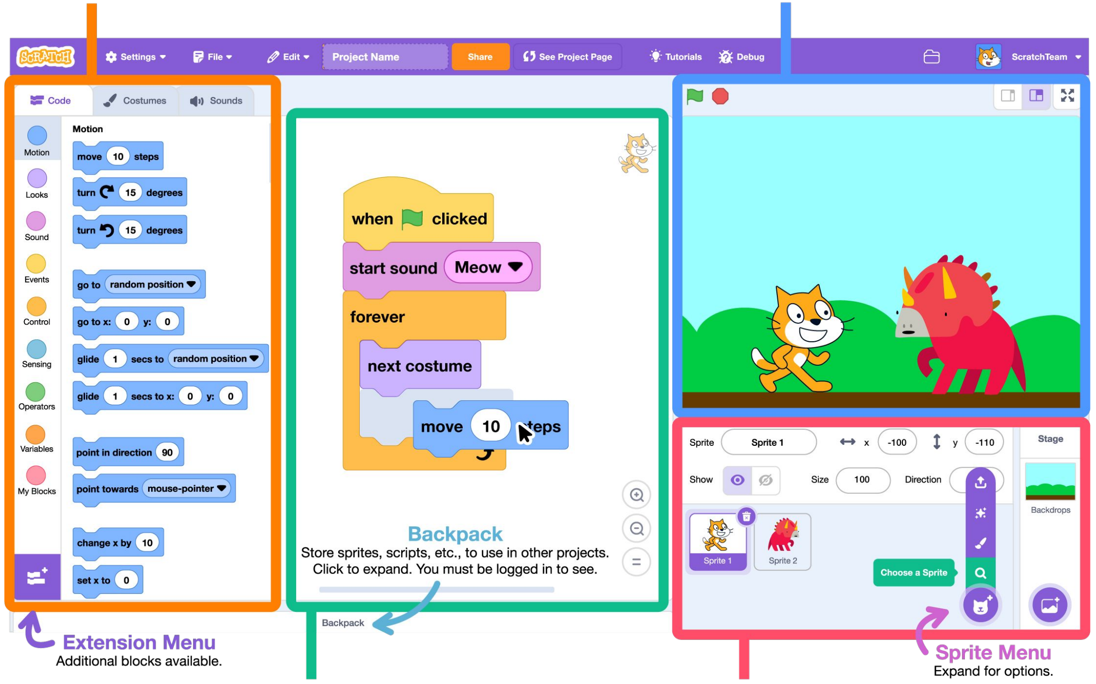
Set your language and block color mode.

Choose a sprite. Drag and drop code blocks to create a script.

scratchfoundation.org/learn/learning-library/getting-started

Block Palette
Blocks for coding your projects.

The Stage
Where your creations come to life.



Backpack
Store sprites, scripts, etc., to use in other projects. Click to expand. You must be logged in to see.

Extension Menu
Additional blocks available.

Sprite Menu
Expand for options.

Coding Area/Script Area
Drag in blocks and snap them together.

Sprite Area
Click the thumbnail of a sprite to select it.

Creative Learning

As facilitators, we want to support **playful learning and tinkering mindset values** so that participants can:

- Engage playfully in **projects** that are meaningful to them and elicit joy
- Collaborate with **peers** to experiment, share, and celebrate ideas
- Develop a mindset that is **comfortable with the discomfort** of getting stuck
- Develop a mindset that thinks critically about **strategies for getting unstuck**

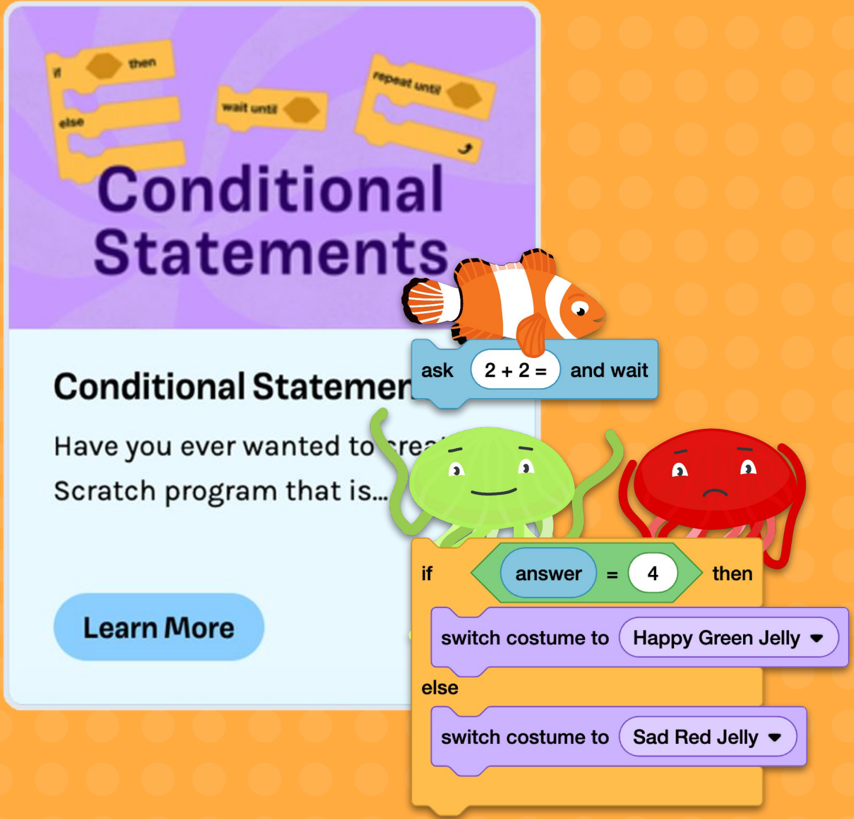
scratchfoundation.org/learn/learning-library/scratch-creative-learning-philosophy

Scratch's Creative Learning Philosophy

Scratch pioneered block-based programming, enabling young...

Let's Imagine...

What will you create?



Conditional Statemen

Have you ever wanted to crea
Scratch program that is...

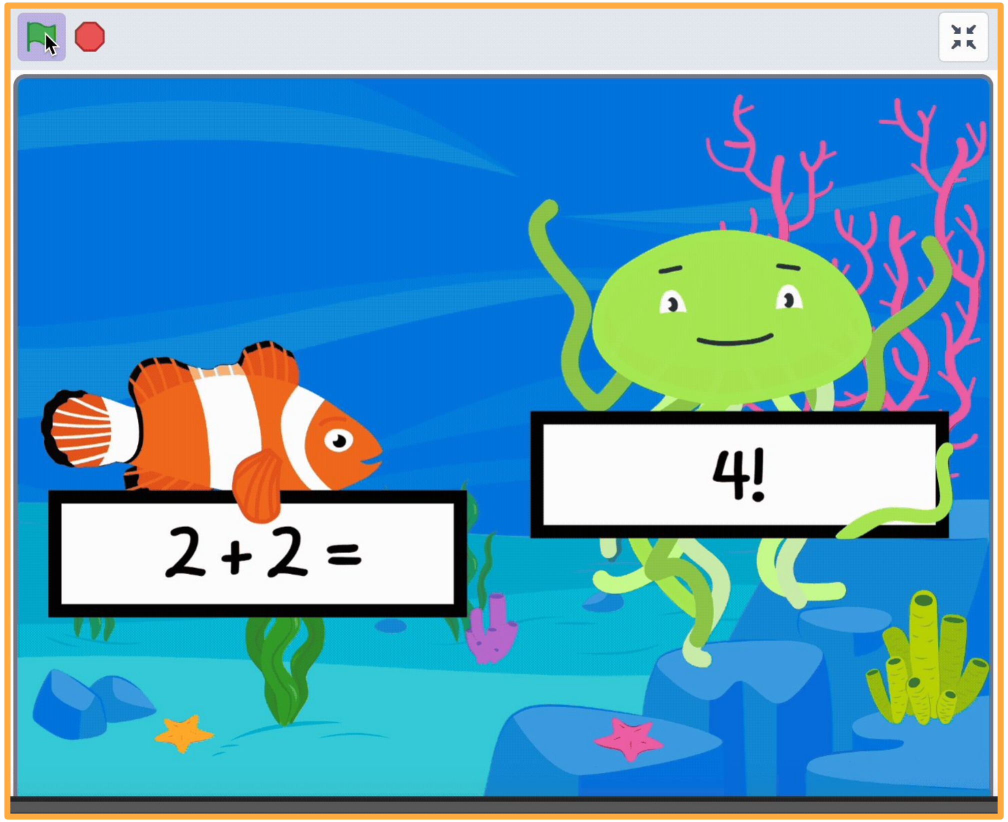
```
ask (2 + 2 =) and wait
if <(answer) = (4)> then
    switch costume to (Happy Green Jelly ▾)
else
    switch costume to (Sad Red Jelly ▾)
```

Static vs Dynamic

Some Scratch programs are **static**: the outcome is fixed and the same thing happens each time.

Some are **dynamic**: they are capable of action or change each time they are run.

In order to create dynamic programs, the programmer can use conditional statement blocks to give instructions on how the project should respond in different circumstances.



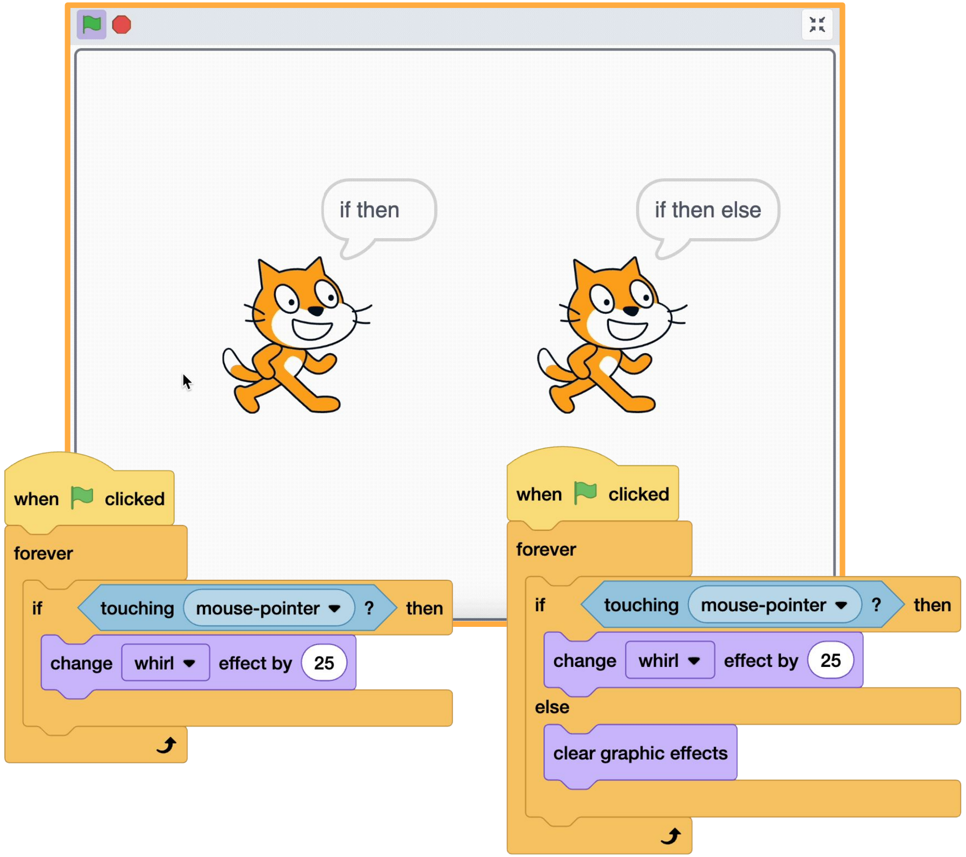
Conditional Statements

Many students first encounter the **computational concept of conditionals** in the context of a game (like a chase game or a pong game), when they want to adjust an outcome based on if sprites are touching or if a sprite is clicked or touched with a mouse, etc.

Conditional blocks can be found in the Control blocks category.

A conditional statement can be **true or false**.

Example projects: 1010683073 and 818527419
More on Pong and Chase Games



```
when [flag] clicked
forever
    if <touching (mouse-pointer ▾) ?> then
        change (whirl ▾) effect by (25)
```

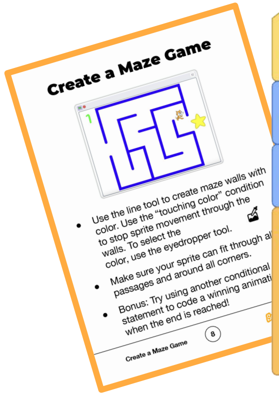
```
when [flag] clicked
forever
    if <touching (mouse-pointer ▾) ?> then
        change (whirl ▾) effect by (25)
    else
        clear graphic effects
```

Starter Project

A-MAZE-ing Conditions

See our "Maze Starter" project and our coding cards. There is often more than one way to arrive at a solution. Debug, iterate, celebrate variety!

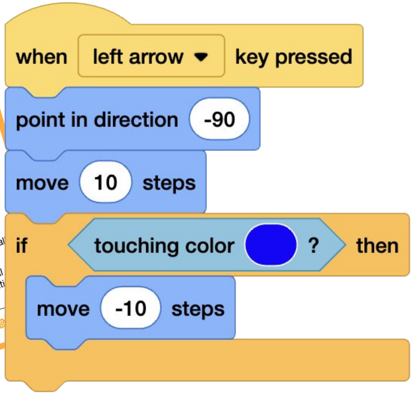
Platform example: 1215320440



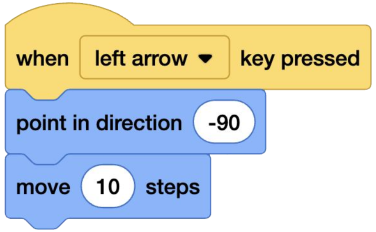
Create a Maze Game

- Use the line tool to create maze walls with color. Use the "touching color" condition to stop sprite movement through the walls. To select the color, use the eyedropper tool.
- Make sure your sprite can fit through al passages and around all corners.
- Bonus: Try using another conditional statement to code a winning animati when the end is reached!

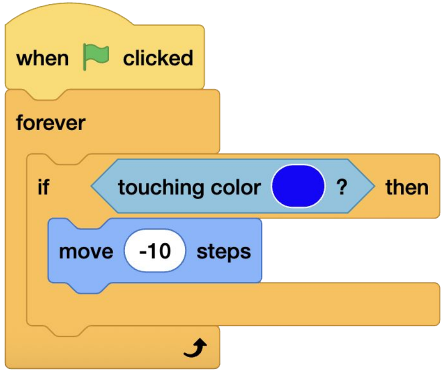
Create a Maze Game 8



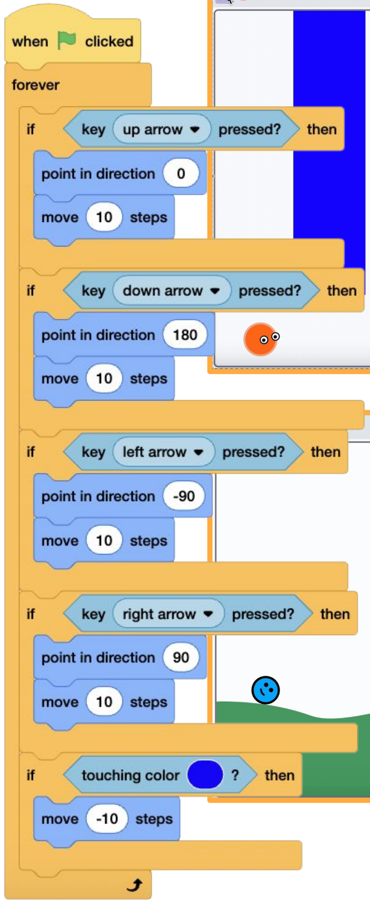
```
when [left arrow ▾] key pressed
point in direction (-90)
move (10) steps
if <touching color [●] ?> then
    move (-10) steps
```



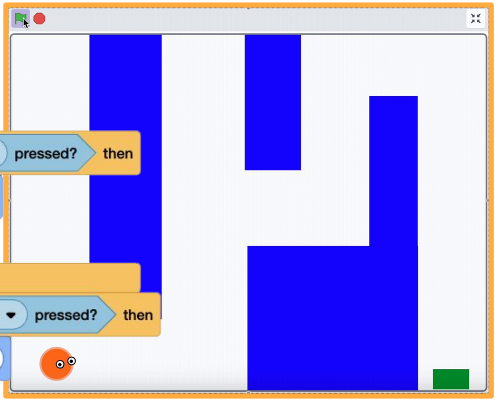
```
when [left arrow ▾] key pressed
point in direction (-90)
move (10) steps
```

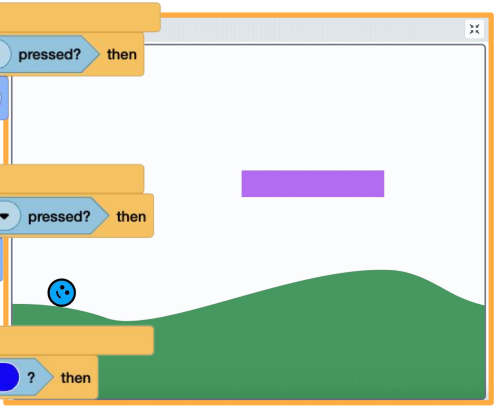


```
when ⚑ clicked
forever
    if <touching color [●] ?> then
        move (-10) steps
    ↻
```



```
when ⚑ clicked
forever
    if <key [up arrow ▾] pressed?> then
        point in direction (0)
        move (10) steps
    if <key [down arrow ▾] pressed?> then
        point in direction (180)
        move (10) steps
    if <key [left arrow ▾] pressed?> then
        point in direction (-90)
        move (10) steps
    if <key [right arrow ▾] pressed?> then
        point in direction (90)
        move (10) steps
    if <touching color [●] ?> then
        move (-10) steps
    ↻
```





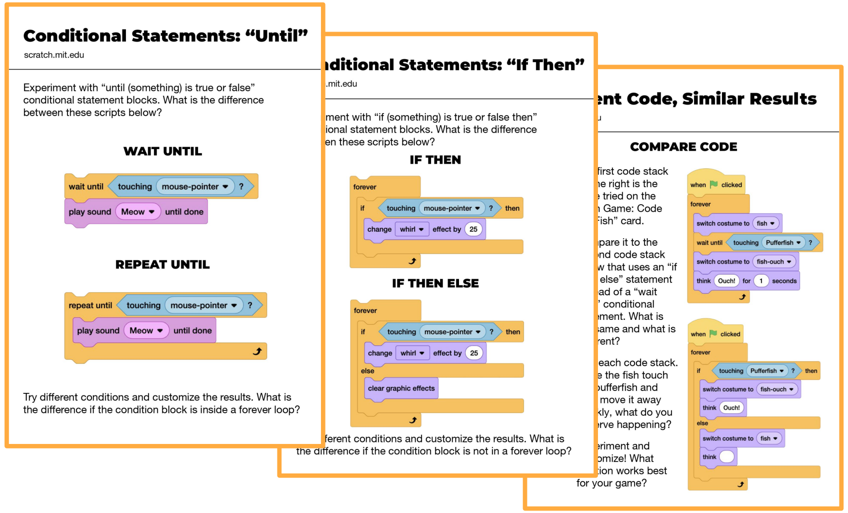
SCRATCH FOUNDATION

Compare, Contrast

See our student-facing coding cards on Conditional Statements for a few example experiments with "wait until" versus "repeat until (something) is true or false," and "if (something) is true or false then" versus "if then else."

Often more than one solution to get a similar result. Other times, using one block over another can produce wildly different results.

Start hands off: What is your hypothesis of what will happen? Experiment! What did you observe?



Conditional Statements: "Until"

scratch.mit.edu

Experiment with "until (something) is true or false" conditional statement blocks. What is the difference between these scripts below?

WAIT UNTIL

```
wait until <touching (mouse-pointer ▾) ?>
play sound (Meow ▾) until done
```

REPEAT UNTIL

```
repeat until <touching (mouse-pointer ▾) ?>
    play sound (Meow ▾) until done
end
```

Try different conditions and customize the results. What is the difference if the condition block is inside a forever loop?

ditional Statements: "If Then"

.mit.edu

ment with "if (something) is true or false then" ional statement blocks. What is the difference en these scripts below?

IF THEN

```
forever
    if <touching (mouse-pointer ▾) ?> then
        change (whirl ▾) effect by (25)
    end
end
```

IF THEN ELSE

```
forever
    if <touching (mouse-pointer ▾) ?> then
        change (whirl ▾) effect by (25)
    else
        clear graphic effects
    end
end
```

ferent conditions and customize the results. What is the difference if the condition block is not in a forever loop?

ent Code, Similar Results

u

COMPARE CODE

first code stack he right is the tried on the Game: Code Fish" card.

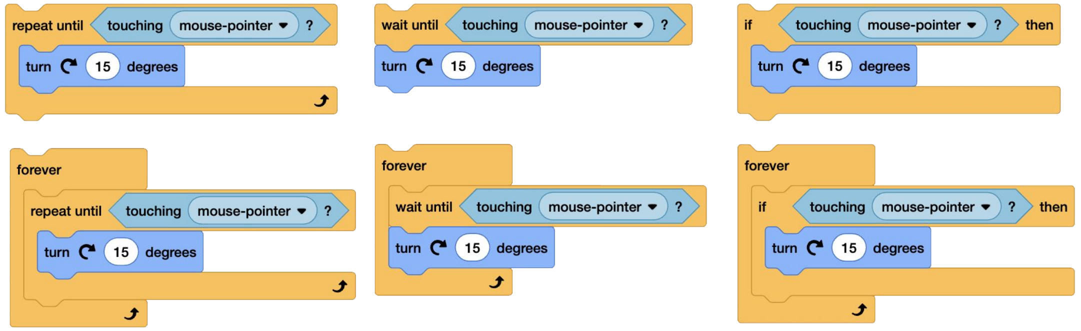
pare it to the nd code stack w that uses an "if else" statement ad of a "wait ' conditional ment. What is same and what is rent?

each code stack. e the fish touch pufferfish and move it away kly, what do you rve happening?

eriment and omize! What ion works best for your game?

```
when flag clicked
forever
    switch costume to (fish ▾)
    wait until <touching (Pufferfish ▾) ?>
    switch costume to (fish-ouch ▾)
    think (Ouch!) for (1) seconds
end
```

```
when flag clicked
forever
    if <touching (Pufferfish ▾) ?> then
        switch costume to (fish-ouch ▾)
        think (Ouch!)
    else
        switch costume to (fish ▾)
        think ()
    end
end
```



```
repeat until <touching (mouse-pointer ▾) ?>
    turn ↻ (15) degrees
end
```

```
wait until <touching (mouse-pointer ▾) ?>
turn ↻ (15) degrees
```

```
if <touching (mouse-pointer ▾) ?> then
    turn ↻ (15) degrees
end
```

```
forever
    repeat until <touching (mouse-pointer ▾) ?>
        turn ↻ (15) degrees
    end
end
```

```
forever
    wait until <touching (mouse-pointer ▾) ?>
    turn ↻ (15) degrees
end
```

```
forever
    if <touching (mouse-pointer ▾) ?> then
        turn ↻ (15) degrees
    end
end
```

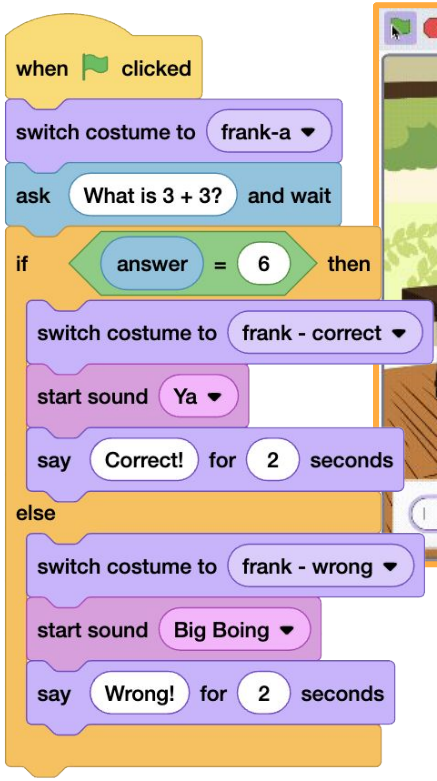
Achievery Unit

Create a Math Game

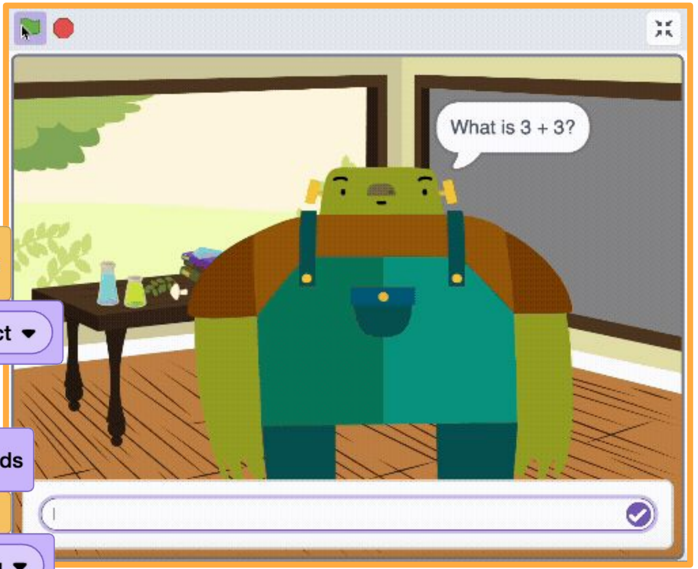
Check out our Achievery Unit “Create a Math Game” and our starter project “Math Game.”

Start by creating a simple conditional statement that checks the “answer” to a question posed with the “ask” block and responds accordingly.

Remix sprites to change expressions and animations (find tips for remixing sprites in our Sprite Creation resources).



```
when [flag] clicked
switch costume to (frank-a ▾)
ask (What is 3 + 3?) and wait
if <(answer) = (6)> then
    switch costume to (frank - correct ▾)
    start sound (Ya ▾)
    say (Correct!) for (2) seconds
else
    switch costume to (frank - wrong ▾)
    start sound (Big Boing ▾)
    say (Wrong!) for (2) seconds
```



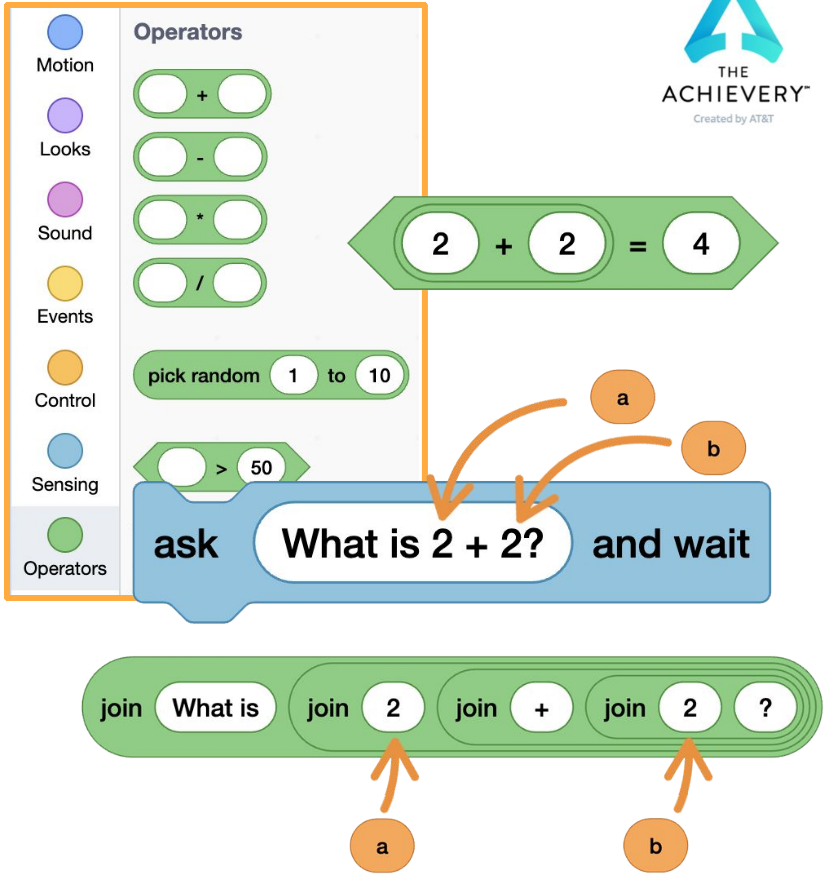
Achievery Unit

Use Operators

To level up this game, let's start by decomposing the question you want to ask:

- What parts of that question would change with each new question and what parts would remain the same? (For example, where do numbers appear in that question?) How can you represent changeable elements?
- How would you write an equation to calculate an answer using Operator blocks?

See our Achievery Unit "Arithmetic Operators" to learn more.



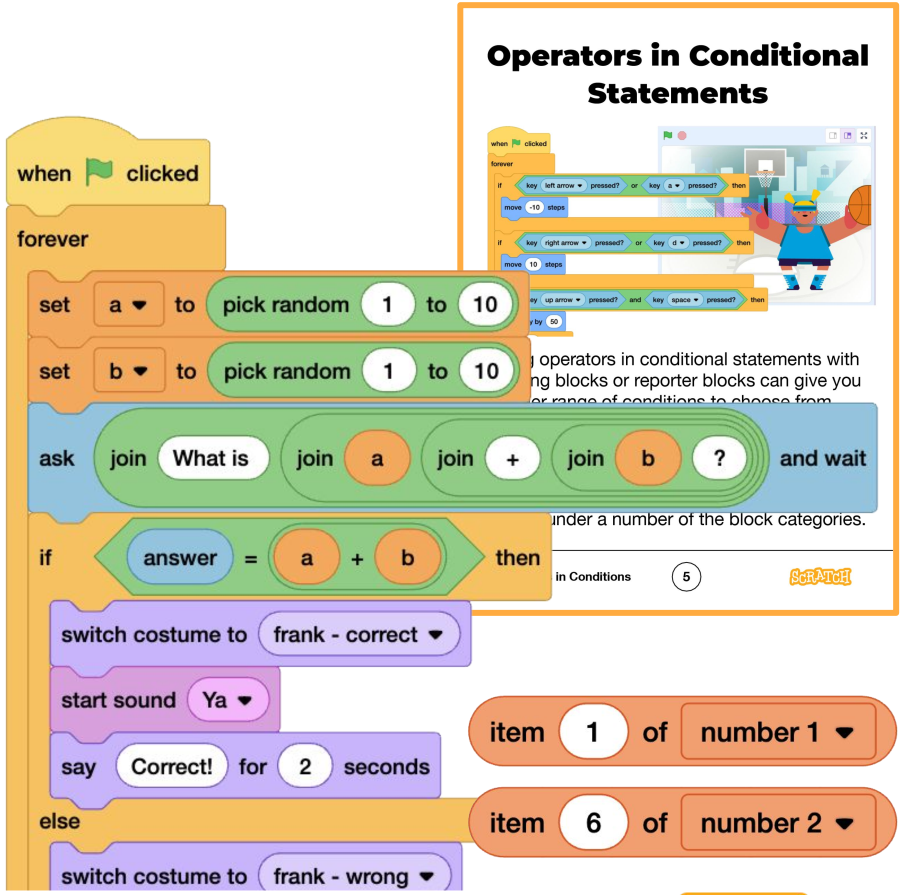
Level Up the Math Game

Rather than repeating the same questions in the same order, we can use those pieces we just created in a version of "Math Game" that is more randomized and even more dynamic. It also makes your code writing more efficient, so you don't have to write out each question and answer.

Thinking about why it is necessary to use "pick random" to set a variable rather than just put "pick random" in the "join" block can be a great debugging opportunity.

Or try lists, versus variables plus random.

A different example of using operators: 1010671466
More on Variables and Lists



Operators in Conditional Statements

```
when [flag] clicked
forever
    set a to (pick random 1 to 10)
    set b to (pick random 1 to 10)
    ask (join (What is) (join (a) (join (+) (join (b) (?))))) and wait
    if <(answer) = ((a) + (b))> then
        switch costume to (frank - correct)
        start sound (Ya)
        say (Correct!) for (2) seconds
    else
        switch costume to (frank - wrong)
```

```
(item (1) of (number 1))
(item (6) of (number 2))
```

operators in conditional statements with
ng blocks or reporter blocks can give you
under a number of the block categories.
in Conditions
5

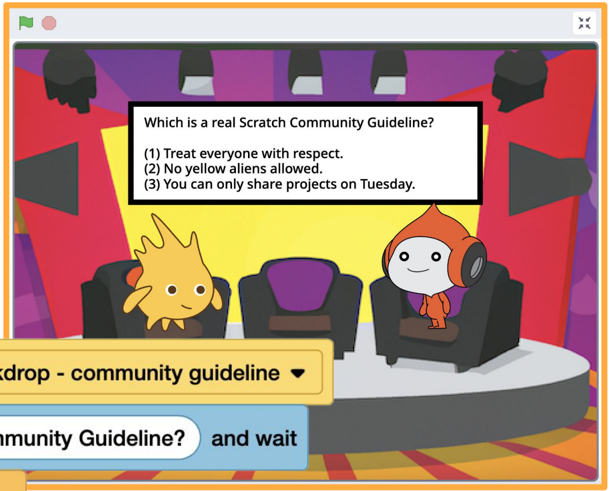
Starter Project

Conditional Storytelling

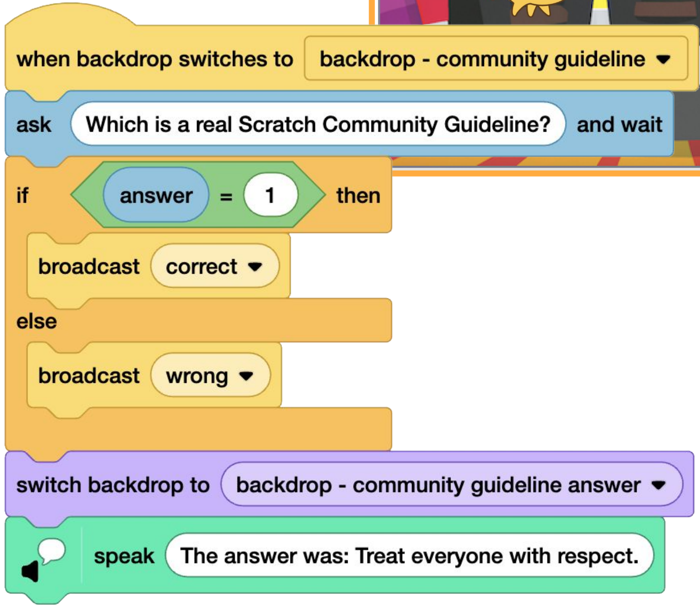
See our starter project "Community Quiz" for an example of using conditional statements in a storytelling or informational project. Explore the code for the backdrop.

How can you use conditional statements to influence the direction of a story?

You could start with our "Story Starter" starter project and continue the story. For example, you could let the viewer choose where the characters should visit next.


Which is a real Scratch Community Guideline?

(1) Treat everyone with respect.
(2) No yellow aliens allowed.
(3) You can only share projects on Tuesday.


```
when backdrop switches to [backdrop - community guideline ▾]
ask [Which is a real Scratch Community Guideline?] and wait
if <(answer) = [1]> then
    broadcast [correct ▾]
else
    broadcast [wrong ▾]
end
switch backdrop to [backdrop - community guideline answer ▾]
speak [The answer was: Treat everyone with respect.]
```

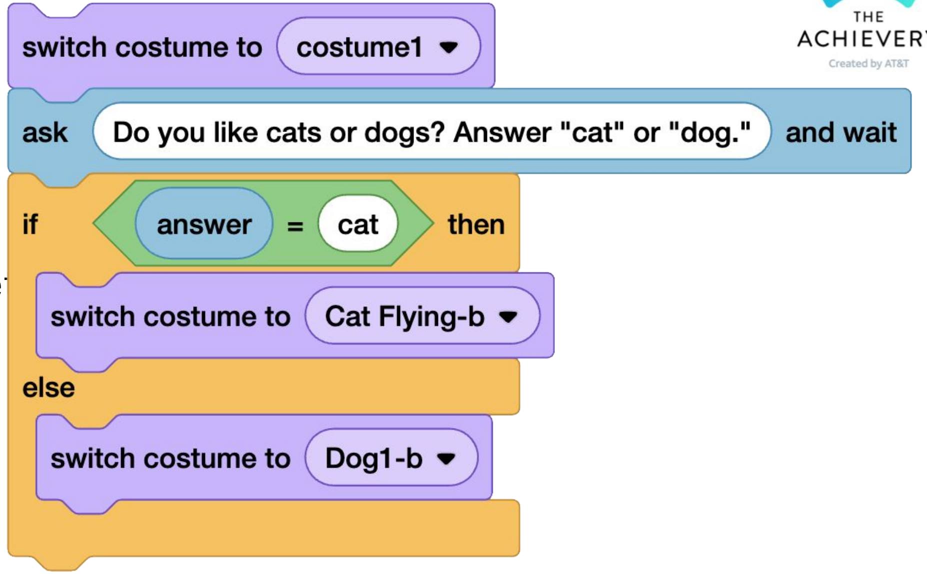
Achievery Unit

The Unexpected

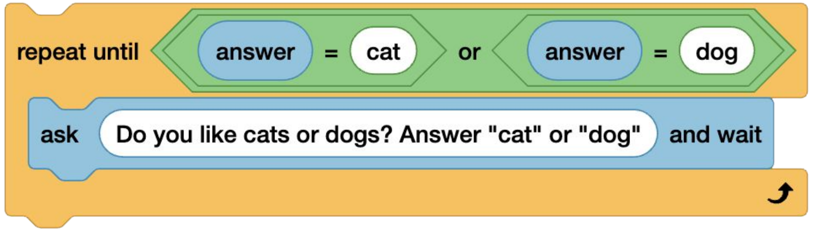
What could happen if you ask a user for information and they give you bad or incomplete data? How can you ensure you get the information you need for a conditional statement to be true or false?

Check out our Achievery Unit "Get the Answers You Want." Using a conditional statement like "repeat until [only specific answers are provided]" ensures that when you move on to the "if then" condition, everything works as expected and you've accounted for unexpected behavior.

An example project: 813299201



```
switch costume to (costume1 ▾)
ask (Do you like cats or dogs? Answer "cat" or "dog.") and wait
if <(answer) = (cat)> then
    switch costume to (Cat Flying-b ▾)
else
    switch costume to (Dog1-b ▾)
```



```
repeat until <<(answer) = (cat)> or <(answer) = (dog)>>
    ask (Do you like cats or dogs? Answer "cat" or "dog") and wait
↺
```

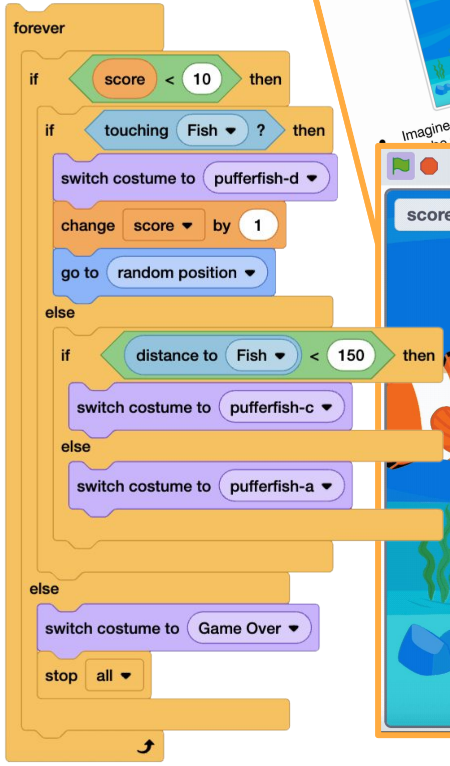
Nesting Statements

Nested conditional statements are where one conditional statement is placed inside another.

The first condition is checked and, depending on if it is true or false, the program may then move forward to check the nested condition inside.

For example, a game could check the total score before continuing to assign points or end the game.

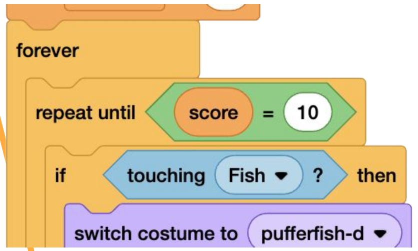
Some example projects: [977165995]() and [1010672973]()



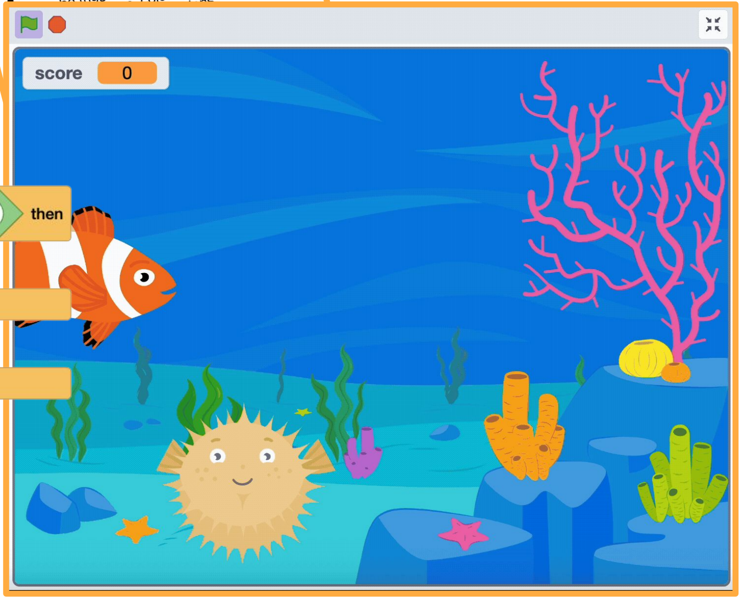
```
forever
  if <(score) < (10)> then
    if <touching (Fish) ?> then
      switch costume to (pufferfish-d)
      change (score) by (1)
      go to (random position)
    else
      if <(distance to (Fish)) < (150)> then
        switch costume to (pufferfish-c)
      else
        switch costume to (pufferfish-a)
  else
    switch costume to (Game Over)
    stop (all)
```

Fish Game: Nested Conditional Statements

Imagine a project where multiple effects
triggered based on the distance
colors sprites are



```
forever
  repeat until <(score) = (10)>
    if <touching (Fish) ?> then
      switch costume to (pufferfish-d)
```



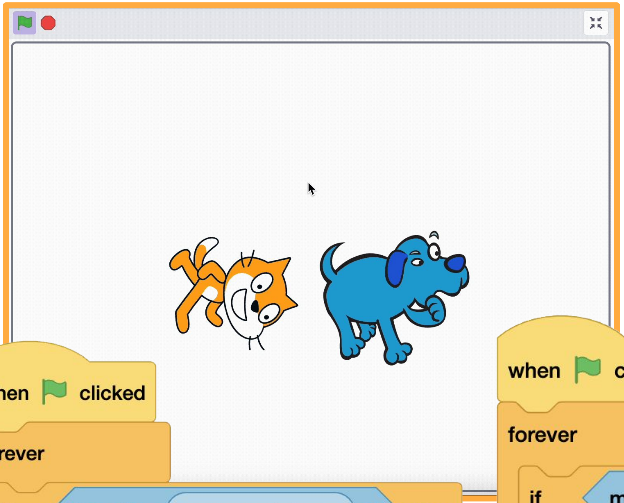
Sequence Matters

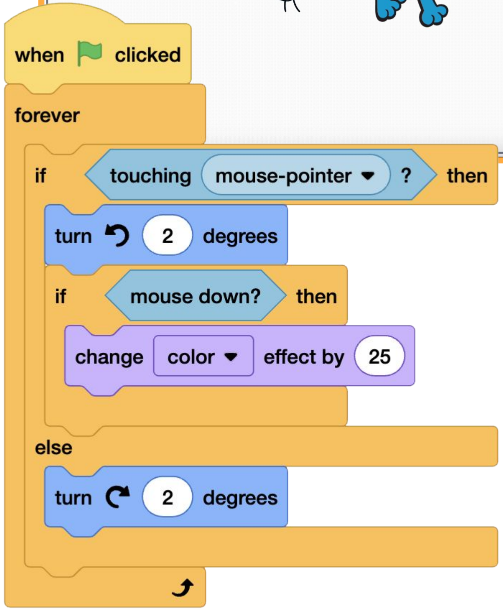
Conditional statement stacks can make for great code reading challenges, like:

- Can you tell which is the cat script and which is the dog?
- What do you think the differences will be if you adjust the order of the nesting?

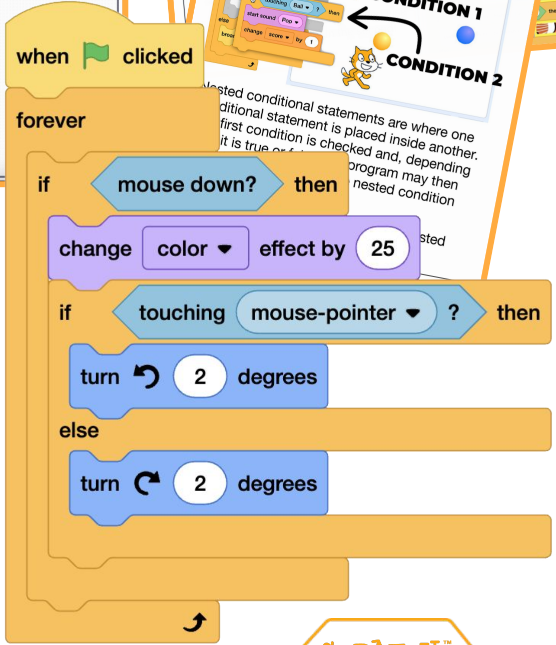
Start hands off: What is your hypothesis of what will happen? Experiment! What did you observe? Why did order make a difference?

An example project: 1010679526

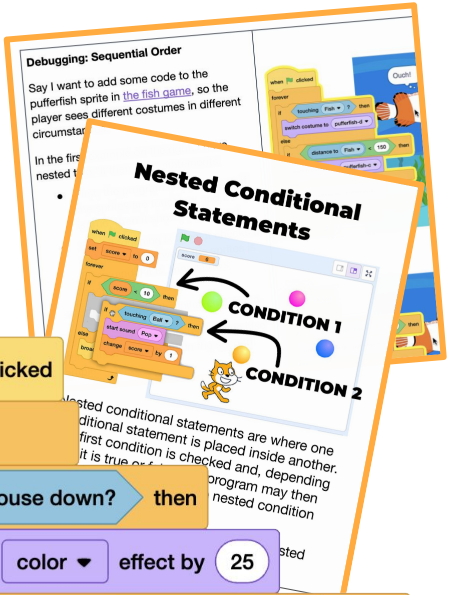




```
when 🏁 clicked
forever
    if touching mouse-pointer ? then
        turn ↺ 2 degrees
        if mouse down? then
            change color effect by 25
    else
        turn ↻ 2 degrees
```



```
when 🏁 clicked
forever
    if mouse down? then
        change color effect by 25
        if touching mouse-pointer ? then
            turn ↺ 2 degrees
        else
            turn ↻ 2 degrees
```



Debugging: Sequential Order

Say I want to add some code to the pufferfish sprite in the fish game, so the player sees different costumes in different circumstan

In the firs nested t

Nested Conditional Statements

Nested conditional statements are where one ditional statement is placed inside another. first condition is checked and, depending it is true or f rogram may then nested condition

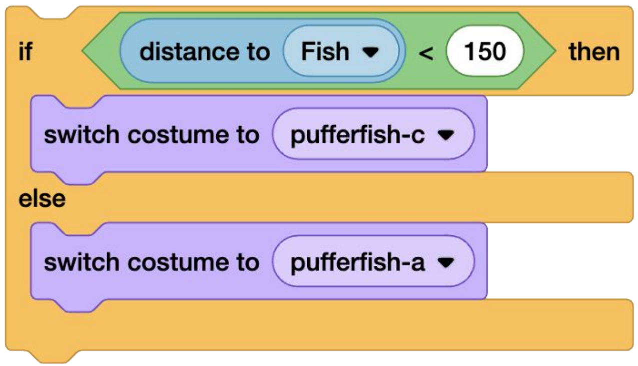
sted

Get Help Setting Your Condition

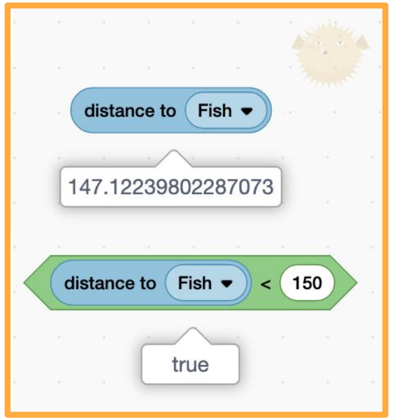
Say you want a condition that is checking the distance between sprites. It can be hard to judge pixel distance by sight.

Click on the reporter block (“distance to”) on the script area to see the value. Click on it while in the operator block to see if it reads true or false.

Temporarily place the reporter inside a “say” block in a forever loop. Now, your sprite will continually report the “distance to” value as it moves.

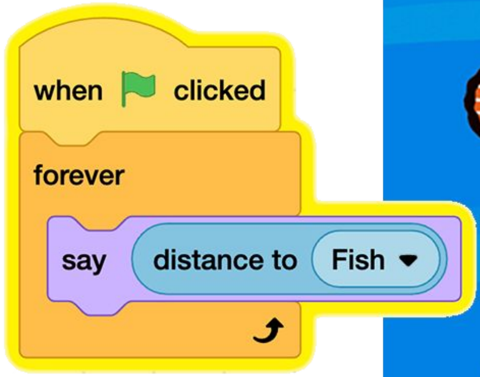


```
if <(distance to (Fish ▾)) < (150)> then
    switch costume to (pufferfish-c ▾)
else
    switch costume to (pufferfish-a ▾)
```

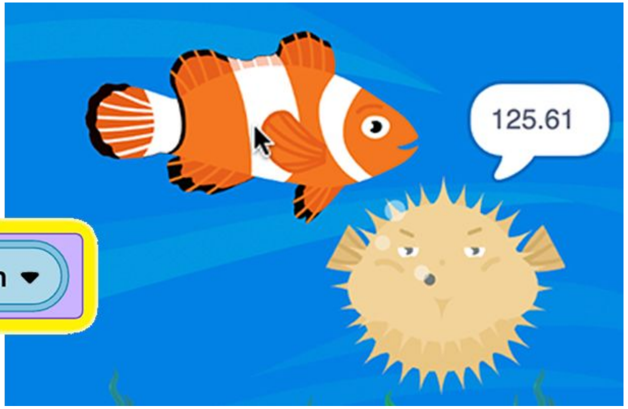


```
(distance to (Fish ▾))
147.12239802287073

<(distance to (Fish ▾)) < (150)>
true
```



```
when 🏁 clicked
forever
    say (distance to (Fish ▾))
```

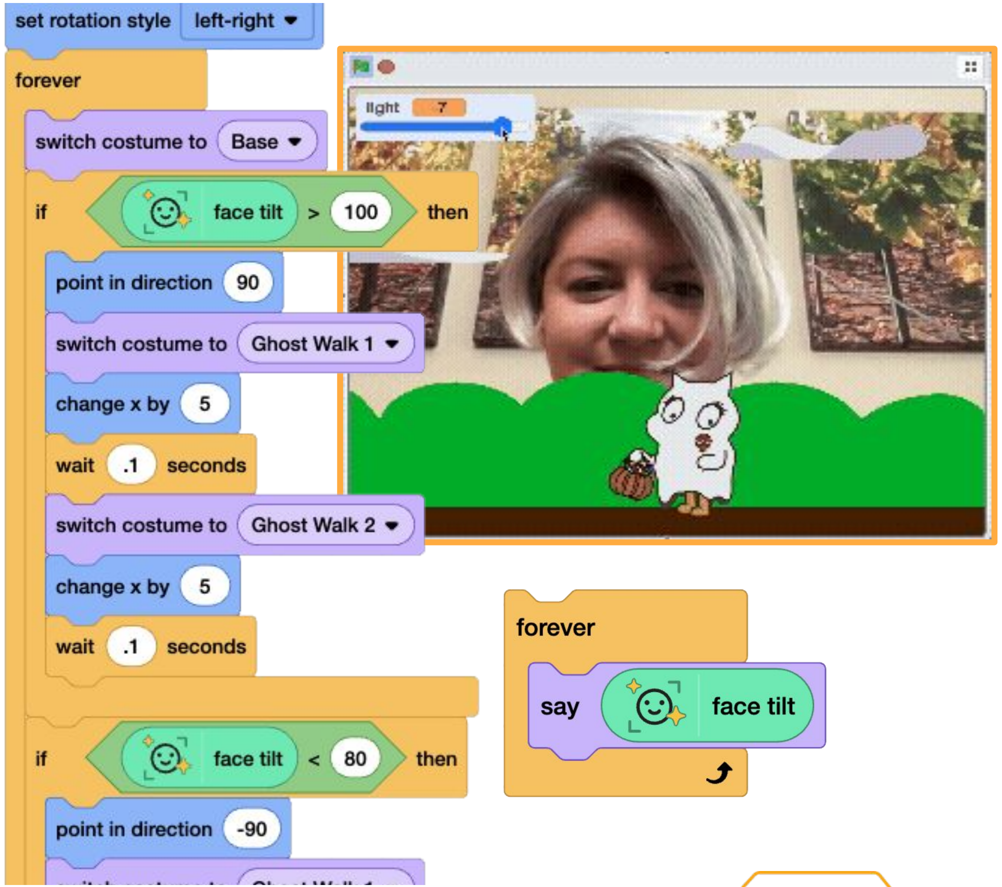


Your Face as the Controller

Try using our Face Sensing blocks and your face to control a project!

These AI-powered blocks use a machine learning model to detect if they see a face and where a nose, eyes, ears, mouth, etc., are. When you use Face Sensing blocks, **only your computer can sense your face. None of your data is stored or sent to Scratch** or any other site, making it a safe, fun, and creative way to explore the possibilities of AI.

More on Face Sensing, including coding cards.
Example project: 1220596710 and studio: 50854499

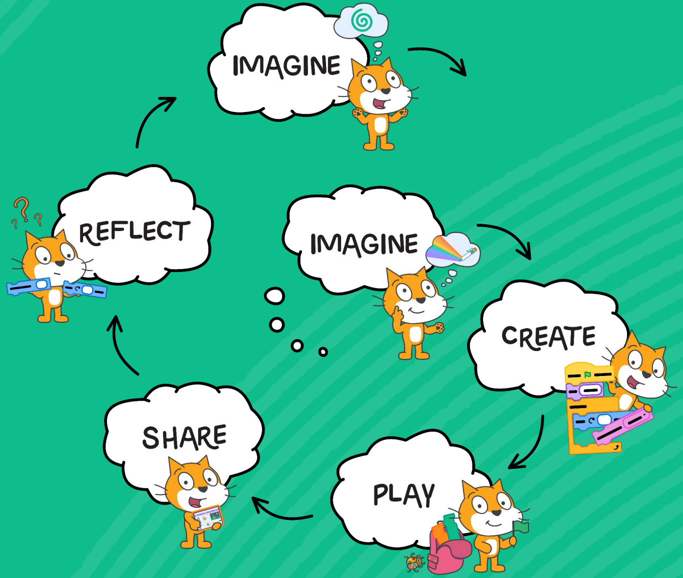


```
set rotation style [left-right ▾]
forever
  switch costume to [Base ▾]
  if <(face tilt) > (100)> then
    point in direction (90)
    switch costume to [Ghost Walk 1 ▾]
    change x by (5)
    wait (.1) seconds
    switch costume to [Ghost Walk 2 ▾]
    change x by (5)
    wait (.1) seconds

  if <(face tilt) < (80)> then
    point in direction (-90)
```

```
forever
  say (face tilt)
```

SCRATCH™ FOUNDATION



Debug, Share, and Reflect

Continue Along the Creative Learning Spiral

Debugging

Debugging strategies to suggest include:

- Read Aloud/Explain the Code Step-By-Step
- Break Long Sequences Apart
- Add Temporary Waits to Slow Action
- Tinker with the Block Order
- Is There a Similar but Different Block Option?
- Check the Values/Inputs

See our Debugging resources for more

Prompts to Try

- "Ask Three Before Me," ask three peers before asking a facilitator.
- I don't know, but let's see if anyone else in the room might know/find out together.
- Which category do you think would be helpful?
- Can you say more about that?
- Let's test it out. What do you observe?
- Walk me through your code. What does it say?

Pair Programming

Have groups of different experience levels? Try pair programming! One person serves as "**driver**" (creating scripts), while the other is a "**navigator**" (reviewing, advising, etc.) and roles are switched frequently.

Together, they can create rich and dynamic projects, in addition to opportunities for them to teach and learn from each other.

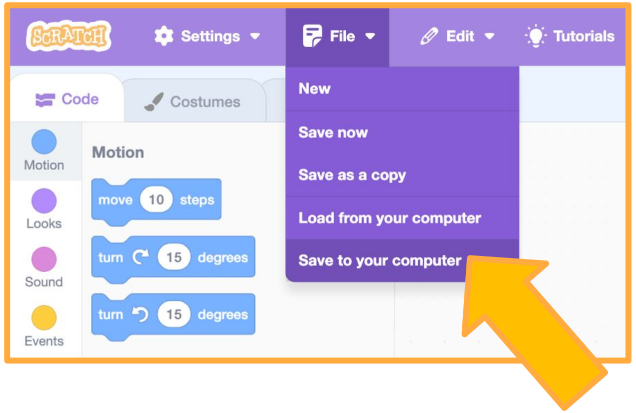
When working with more complex topics like conditional statements, pair programming can be helpful for the debugging process.

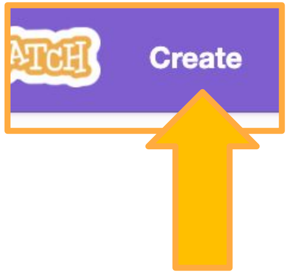
Saving

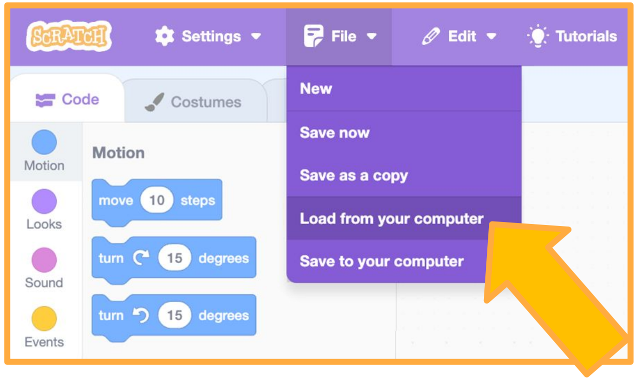
If you have a Scratch account, your project will save automatically.

If you don't have a Scratch account yet, you can save your project to your computer. Click "**File**," then choose "**Save to your computer**."

Next time you want to work on your project, go to scratch.mit.edu and click "Create." Then click "**File**," choose "**Load from your computer**," and upload your project.







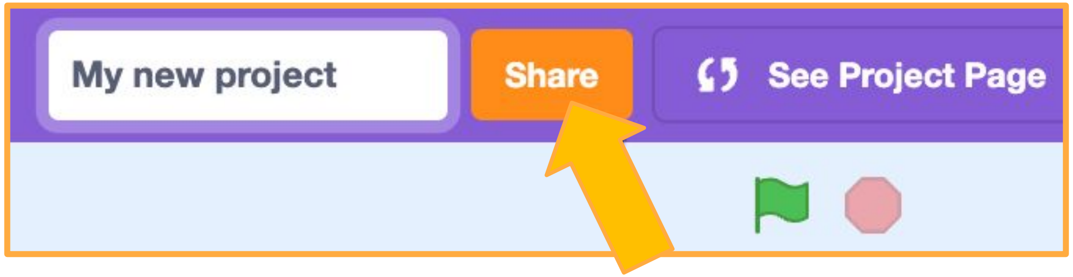
Share Your Project

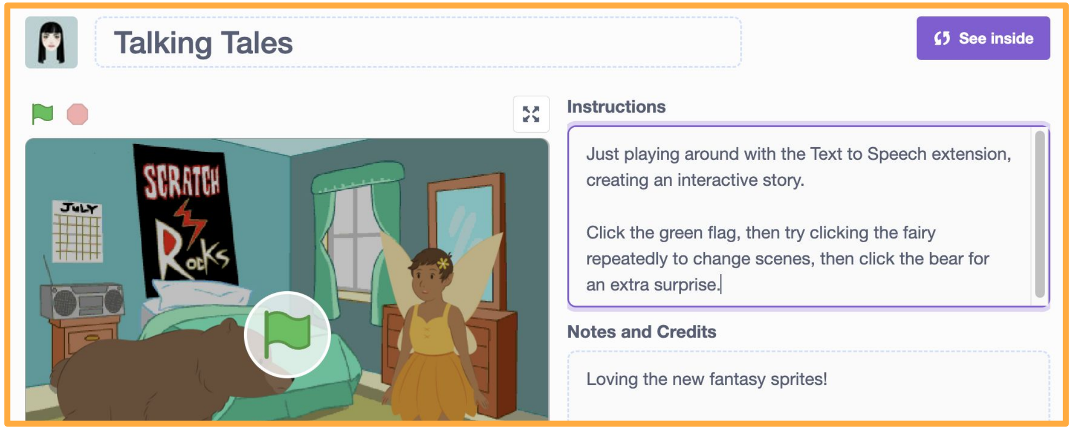
If you have a Scratch account, you can share your project and add it to studios.

Click the orange "**Share**" button at the top of the Scratch editor to share your project with the Scratch community.

Click the "See Project Page" button to go to the project page. This is where you can **add instructions and notes** about your project.

Now other Scratchers can see and interact with your project!





Just playing around with the Text to Speech extension, creating an interactive story.

Click the green flag, then try clicking the fairy repeatedly to change scenes, then click the bear for an extra surprise.

Loving the new fantasy sprites!

SCRATCH FOUNDATION

Reflection

- What was fun about this activity?
- What struggles or frustrations did you have during this activity?
- Many pathways, many solutions: Compare your code with other solutions. Was your solution similar or different? Why did you choose the blocks you did?
- If you had more time what would you add or change?

See our Reflection and Sharing resources for more

Prompts to Try

- I love it! What is it?
- What are your next steps for this project? What do you want to do in the future?

“Turn & Talk” is one technique to reflect and share in a physical environment.

Breakout rooms are an option for small group reflection in virtual spaces.

Record reflections using Scratch’s sound editor. Then, add to a reflection project.

After-Activity Reflection

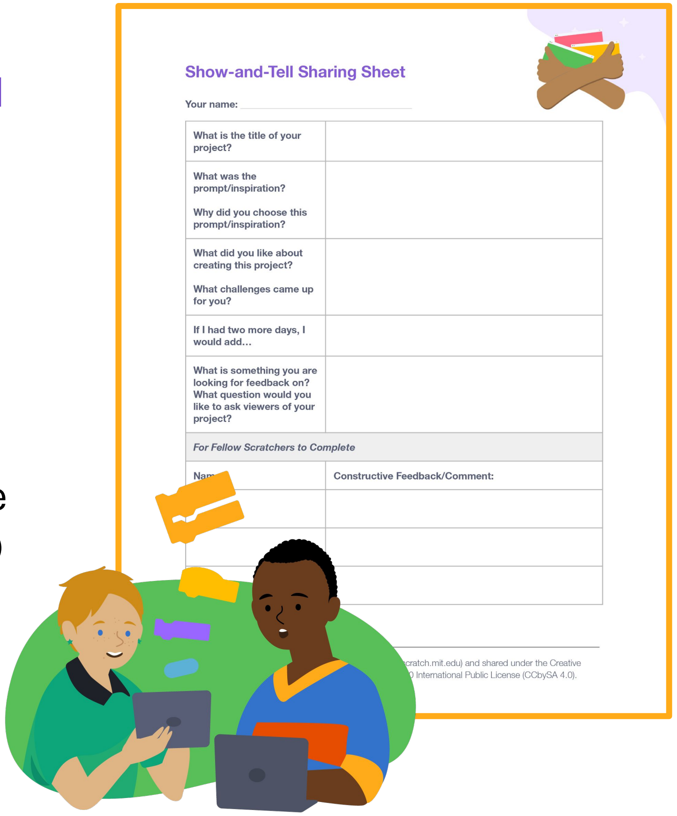
Share Option #1: Create a Class Studio to Gather Shared Projects

Studios are a space on Scratch where users can come together to make, share, and collect projects related to a particular theme, idea, or prompt.

Share Option #2: Gallery Walk

Have your project open on your computer. Walk around the room (or take turns sharing your screen in a virtual space) to experience each other's creations. Take time to look at projects and read/listen/interact with them to learn more about your peers.

More on Teacher Accounts, Studios, and our Reflection and Sharing resources



Show-and-Tell Sharing Sheet

Your name:

What is the title of your project?	
What was the prompt/inspiration? Why did you choose this prompt/inspiration?	
What did you like about creating this project? What challenges came up for you?	
If I had two more days, I would add...	
What is something you are looking for feedback on? What question would you like to ask viewers of your project?	
For Fellow Scratchers to Complete	
Nam	Constructive Feedback/Comment:

cratch.mit.edu) and shared under the Creative
0 International Public License (CCbySA 4.0).

SCRATCH™ FOUNDATION

Wrapping Up

Reflecting on Our Session, Resources, Next Steps

Get a copy of our Creative Learning Materials!

In addition to the resources shared throughout these slides:

- See our Learning Library at scratchfoundation.org/learn/learning-library to find lesson plans, coding cards, tutorial videos, and more! For this session, Conditional Statements would be perfect to explore.
- Getting Started with Scratch
- Scratch Creative Learning Philosophy

Find help, inspiration, and information:

- Visit scratch.mit.edu/ideas and scratch.mit.edu/starter-projects
- Click "Tutorials" to see in-editor guides
- Watch tutorial videos on our channel youtube.com/c/ScratchTeam

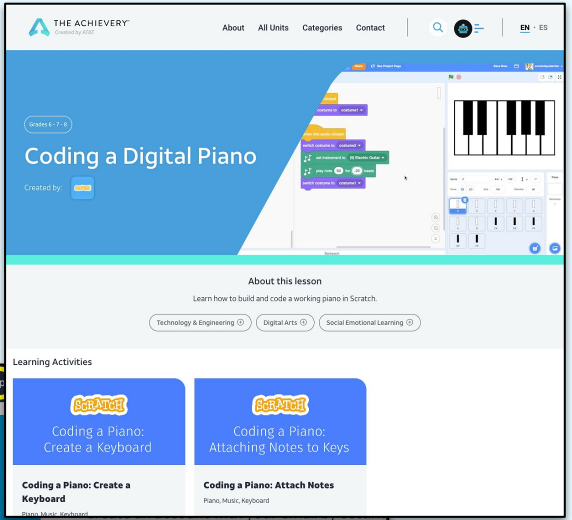
Find Scratch on The Achievery!

The Achievery platform connects K-12 students to a new world of digital learning.

Scratch Foundation has teamed up with The Achievery to provide free beginner and intermediate creative coding lesson plans on a variety of topics for educators, caregivers, and learners.

Sign up (for free!) by using our custom code "SCRATCH" when you register to support our work!

theachievery.com/account/signup



THE ACHIEVERY Created by AT&T

About All Units Categories Contact EN · ES

Grades 6 - 7 - 8

Coding a Digital Piano

Created by:

About this lesson

Learn how to build and code a working piano in Scratch.

Technology & Engineering · Digital Arts · Social Emotional Learning

Learning Activities

Coding a Piano: Create a Keyboard

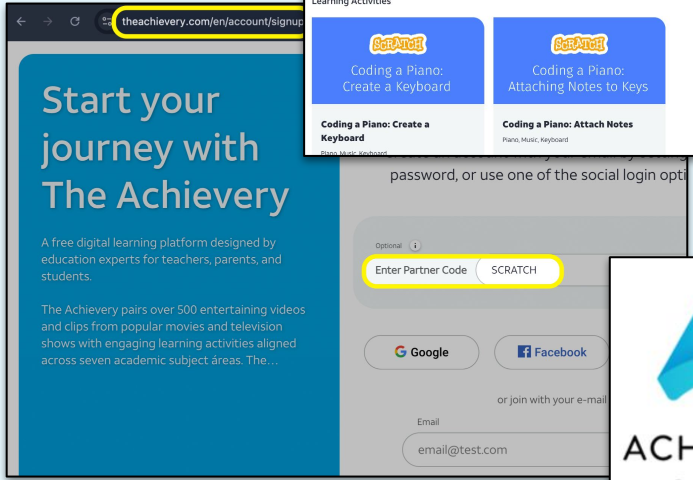
Coding a Piano: Create a Keyboard

Piano, Music, Keyboard

Coding a Piano: Attaching Notes to Keys

Coding a Piano: Attach Notes

Piano, Music, Keyboard



theachievery.com/en/account/signup

Start your journey with The Achievery

A free digital learning platform designed by education experts for teachers, parents, and students.

The Achievery pairs over 500 entertaining videos and clips from popular movies and television shows with engaging learning activities aligned across seven academic subject áreas. The...

password, or use one of the social login opti

Optional

Enter Partner Code: SCRATCH

Google Facebook

or join with your e-mail

Email: email@test.com

Thank you!

Be sure to subscribe to our Scratch Foundation YouTube channel for Educators (@scratchfoundation).

Keep an eye on our Event page for additional opportunities: scratchfoundation.org/get-involved/events

Helpful Links:

- Scratch Application: scratch.mit.edu
- Learning Library: scratchfoundation.org/learn/learning-library
- Email Signup: scratch.mit.edu/connect
- Follow us on Instagram and Facebook @ScratchTeam
- Also see our YouTube channel @scratchteam for tutorials