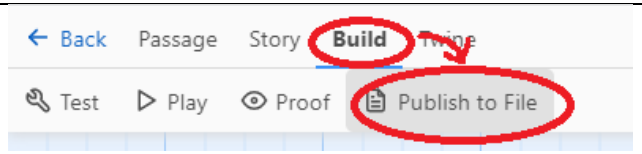
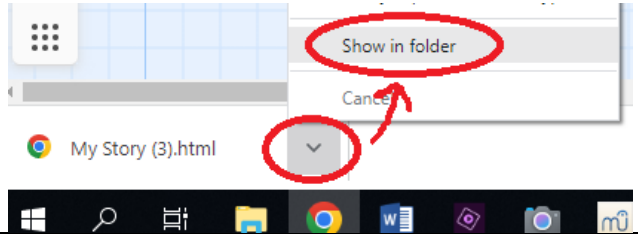
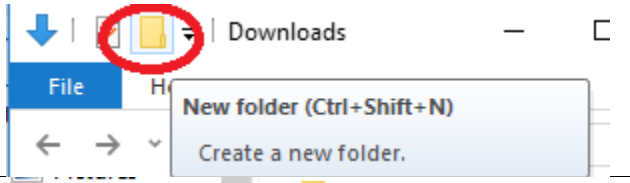
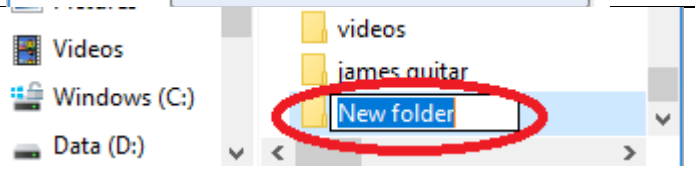
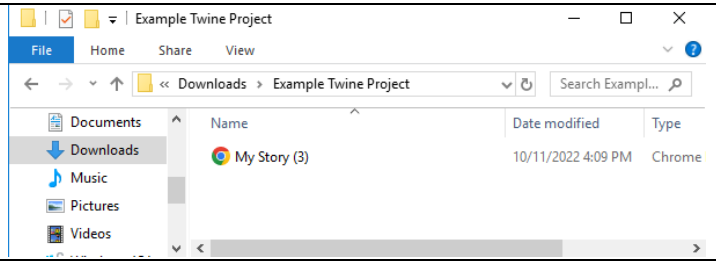
TOP TWINE CODE TIPS for STUDENTS

SAVING YOUR GAME TO A PROJECT FOLDER

You should save each Twine game in its own project folder, along with any other files it uses (like pictures). Here's how:

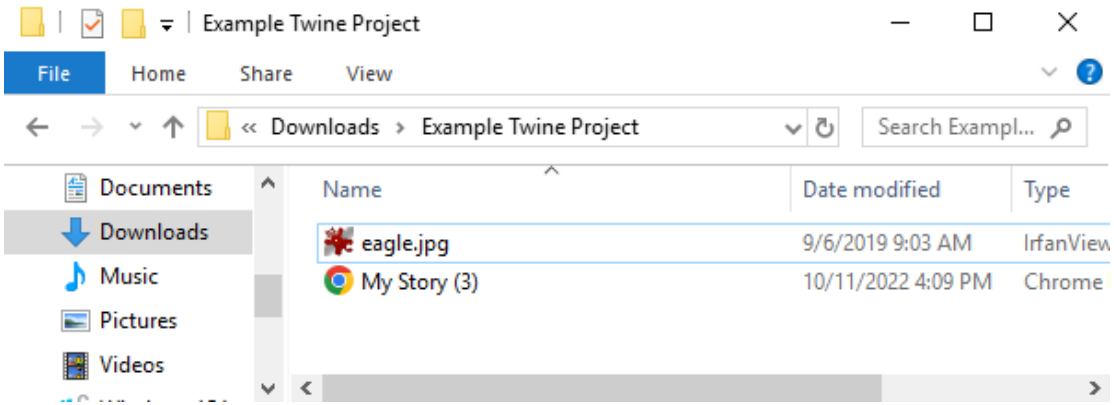
1	Click **Build**, then **Publish to File** to save your Twine game as a file. This is also called "exporting" your file. Your file will be saved in the **Downloads** folder on your computer.	
2	Your game file will appear at the bottom of your screen. Click on the **down-arrow** next to it and then click **Show in folder.**	
3	Make a new folder by clicking on the New Folder icon (or pressing Ctrl, Shift and N keys at the same time).	
4	Name your new folder (the name of your game is a good choice for this). This folder is now our **project folder** for this game.	
5	**Click on your game file and drag it into your project folder.** When you go inside the project folder (double-click it), you should now see your game file there.	

You only need to make the Project Folder once, but you will need to repeat steps 1, 2, & 5 each time you want to save a new version of your game in your folder.

PICTURES

Pictures only work when playing the **published** version of your game located **inside your project folder**.

The **picture file** you want to use needs to also be **saved in the same project folder** as your game.

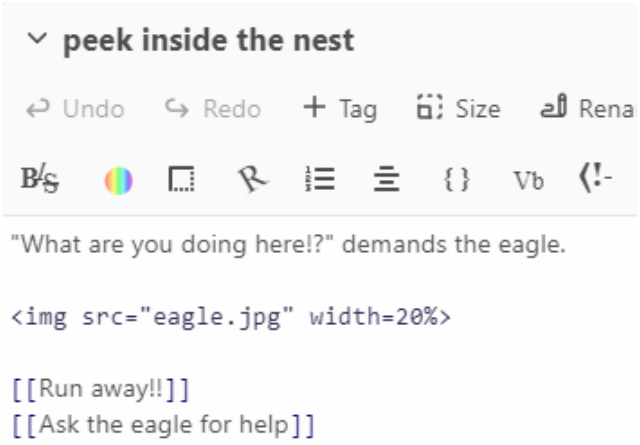


To add an image:

1. Save your picture file inside your **project folder**. (You can click and drag it if it is already saved somewhere else on your computer)
   If your picture file has a complicated name, **right-click** the file and then click **Rename** to change the name to something easy to type).

2. In your Twine game, type or copy this code into the passage where you want the picture to be:

   <img src="eagle.jpg" width=20%>



*For your game, **replace eagle.jpg** with the actual file name of the picture you have saved in your project file. This name must **exactly match the name of your file**, including if there are any capitalized letters.*

The picture will be 20% as wide as the screen. You can make this number bigger or smaller to change the size of the picture.

3. Publish a new copy of your game to your project file to save your new changes there.

4. Open your game file (double-clicking the file inside your project folder) to test your game with the new picture.

5. When you test or play your game from inside the Twine editor, you will only see a “broken image” icon where your pictures are supposed to be. This is normal. Remember, the pictures will only work using the **published** version of your game located **inside your project folder** (which is what you will share with others when you are ready for others to try playing your game).

LINKS

Simple link **[[open the door]]**	Creates a link that says “open the door” and goes to the passage named **open the door**.
Named Link **[[go back outside->start]]** or **[[go back outside|start]]**	Creates a link that says “go back outside”, but goes to the passage named **start**.
Coded Link (advanced) **(link:"open the door")[(go-to:"bad ending")]**	Turns the text “open the door” into a clickable link to a passage titled *bad ending*. This method **does not** create a new passage with the destination name if it does not already exist, and **does not create arrows in the editor** to visually show this type of connection between passages. This method is useful because allows for other coding functions to be added to the link when it is clicked. For example, (link:"open the door")[**(set: $score to 0)**(go-to:"bad ending")] contains extra code that will activate and set the variable called score to a value of 0 only when the link is clicked.

Passage names are case sensitive. “LIBRARY”, “Library”, and “library” are all different names in Twine. Extra spaces or differences in punctuation will also be treated as different passage names.

(MACROS:), [HOOKS], and $VARIABLES

Macros, hooks, and variables are the parts of Twine's code that let us do more interesting things with coding. **(if: $score is 5)[You win, $playername!]** uses the **(if:)** macro to check whether or not the **$score** variable is 5. If this is true, the attached **hook** [inside square brackets] will be displayed, showing the text "You win, Alex!" (assuming that the player was earlier given the opportunity to set their $playername to Alex). If the $score variable is anything other than exactly 5, the contents of that hook will not be displayed.

Passage contains:	Reader sees:
(set: $score to 1)	
(set: $playername to "Alex")	
Score: $score	Score: 1
My name is $playername	My name is Alex

If you try using a $variable that you have not **defined** (by using the (set:) macro), the $variable's value will be **0**.

Numbers and true/false values for variables (such as **(set: $speed to 30) or (set: $key to true)**) do NOT need quotation marks around them. Any other words DO need quotation marks, "like this" (for example, (set: $pet to "dog")). If you write (set: $speed to "30"), the value of the variable will be treated as a "string" (plain text) of the characters 3 and 0 instead of the numerical value thirty, and will not be usable in operations like (set: $speed to it +5) or for evaluating numerical conditions such as (if: $speed > 50)[You are going too fast!].

Variable names are **case-sensitive** ($SPEED, $Speed, and $speed are all treated as separate variables).

OTHER USEFUL TIPS

TEST your story/game after every edit.

The **backward/forward arrows on the left side of your twine story act as undo/redo buttons for the reader/player**, not navigation links for "moving" between locations/scenes in your game/story. This means that if the reader uses the back/undo arrow to leave a passage and does not return, the (history:) macro will not count that passage as having been visited, and any other effects of visiting that passage (such as setting variables) will be undone. If you wish to remove the undo/redo arrows from your story/game, see the **USEFUL CODE** section of this document.

If your story game is very large and complex, **create a "Quickstart" passage** that you can use to set your variables to whatever you like and jump right to the middle of the story/game instead of needing to playtest from the very beginning every single time. This is also a great way to find out if certain features like image size and audio are working properly on other devices.

Use **{Curly brackets}** around your code so that line spacing used to keep the code readable does not display large empty spaces when your passages are read.

Tag a passage as **header** (below the passage's title) to have it appear as a header at the top of every passage in your story/game. The same can be done for a **footer** at the bottom.

If you want to **omit certain pages from displaying a header**, tag those passages as **no-header** and then include this code in the body of the header passage:

(unless: (passage:)'s tags contains "no-header")[CONTENT OF THE HEADER]

When you think your story or game is complete, ask friends to **play-test** it for you before publishing. It's amazing how many bugs and errors can slip by you that a new set of eyes can catch right away!

USEFUL MACROS

Macro name	examples	result
(set:)	(set: $dollars to 0) (set: $points to it + 1) (set: $name to "James") (set: $headgear to $color+" hat")	$dollars becomes **0** $points becomes **1 more than its previous value** $name becomes **James** $headgear becomes **blue hat** *(assuming $color had been set to blue)*
(if:)	(if: $pet is "dog")[Your pet is a loyal companion.] (if: $shoes > 2)[You have more shoes than you can wear at once!] (if: $money is >= 3.75)[You have enough money to take the bus.]	If the $pet variable has been set to the text string "dog", displays: **Your pet is a loyal companion.** Otherwise, nothing is displayed. If the $shoes variable is a number greater than 2, displays: **You have more shoes than you can wear at once!** Otherwise, nothing is displayed. If the $money variable is a number equal to or greater than 3.75, displays: **You have enough money to take the bus.** Otherwise, nothing is displayed.

(else-if:)	(if: $pet is "dog")[Your pet is a loyal companion.] (else-if: $pet is "cat")[Your pet has nine lives.]	If the $pet variable has been set to “dog”, displays: **Your pet is a loyal companion.** If the condition of the initial (if:) macro is not met, any following (else-if:) macros will then be checked in order. If $pet is “cat”, this code will display **Your pet has nine lives.** If $pet is neither “dog” nor “cat”, nothing will display.

(else:)	(if: $pet is "dog")[Your pet is a loyal companion.]
	(else-if: $pet is "cat")[Your pet has nine lives.]
	(else-if: $pet is "dragon")[Your pet is mythical.]
	(else-if: $pet is 0)[You do not have a pet.]
	(else:)[I’m not sure what to say about your pet.]
		If none of the (if:) or (else-if:) conditions are met, the final (else:) macro will be triggered, revealing **I’m not sure what to say about your pet.**
		
		{
		The example on the left is a perfect situation to surround the code with curly brackets like this, to prevent empty line breaks from being displayed in your story
		}

(link-reveal:)	(link-reveal:"You open the box... ")[and find nothing inside.]	Turns the text **You open the box...** into a one-time clickable link. When clicked, the initial text remains (but is no longer clickable) and the contents of the hook are revealed, resulting in **You open the box... and find nothing inside.**
(link-repeat:)	(link-repeat:"say it again")["it again"]	Turns the text **say it again** into a link that can be clicked repeatedly. The contents of the hook repeat every time the link is clicked, inserting the text **“it again”** to the story as many times as the link is clicked.
	(link-repeat:'<img src="trade.gif" width=20%>')[(set: $hotdog to it - 1)(set: $pizza to it + 1)]	Inserts the image **trade.gif** (*assuming this image is in the same folder as where the Twine file is saved and being opened from – this can be replaced with a hyperlink to an online image too*) and turns it into a link that can be clicked repeatedly. The contents of the hook repeat every time the link is clicked, in this case lowering the value of the $hotdog variable and increasing the value of the $pizza variable.
(go-to:)	(go-to:"library")	On its own, the (go-to:) macro will cause the story to immediately advance to the passage called **library** without any interaction from the player. This can sometimes be useful if you want to have an “invisible” in-between passage for running macros that the player technically visits but never actually sees.
	(link:"go to the library")[(set:$location to "library")(go-to:"library")]	This example creates a clickable link (**go to the library**) that sets the value for a variable at the same time as sending the player to the destination passage when the link is clicked.

(print:)	(print:"$5")	**$5** will be written without turning it into a variable. Useful for anything you want to write as plain text that is unintentionally being turned into code by Twine.
(display:)	(display: "Dream Sequence")	The full contents (including text and code) of the passage named **Dream Sequence** will be displayed at the current point in this passage. This can be useful if you have a large block of text or code that you want to use frequently in many passages, without needing to copy it out multiple times (or needing make the same update to all those places if you later decide to edit that text/code).
(prompt:)	(set: $playername to (prompt: "My name is:", ""))	Creates a pop-up box that reads **My name is:** that will set the $playername variable to whatever the reader types in.

OTHER USEFUL CODE (Stylesheet)

*To add the following features to your story, click on your story's title in the bottom-left corner, select "**Edit Story Stylesheet**", and then paste in the code you want to add.*

Remove the sidebar with the "back" and "forward" navigation arrows (make sure not to leave any dead-end passages with no links unless you intend for them to be endings – and even then, sometimes a "start over" feature can be nice.)

```
tw-sidebar
{
	display: none;
}

tw-passage img {
	display: block;
	margin: 0 auto;
}
```

Remove/Disable the debug tool

```
tw-debugger {
	display: none;
}
```

Change background color, font type, font size, font color, and link color

Any attributed that you do not wish to alter can be removed from this code (for example, you can delete “font-size: 28px;” from the code if you just want the default font size to be used).

Use the Google Color Picker to select the 6-caracter HEX color codes you want: https://g.co/kgs/FNmKJS)

Find names for web-safe font types at www.cssfontstack.com

```
body, tw-story
{
background-color: #D1FFF8;
color: #D1FFF8;
font-family: Century Gothic,CenturyGothic,AppleGothic,sans-serif;
font-size: 28px;
color: #000000;
}

tw-link
{
color: #06907B;
}
```

Set different background images for different passages based on their tags

Passages with the tag “forest” will use the image ***trees.jpg*** as their background; passages with the “night” tag will use ***stars.jpg*** as their background. This also works with URLs for web-hosted images (the current example assumes a local image file in the same folder as the Twine HTML file).

```
tw-story[tags~="forest"] { background-image: url("trees.jpg"); background-repeat: no-repeat;
background-size: cover; } tw-story[tags~="night"] { background-image: url("stars.jpg");
background-repeat: no-repeat; background-size: cover; }
```

Set different font colours for different passages based on their tags

Passages with the tag “forest” will have grey text; passages with the “night” tag will have yellow text.

```
tw-story[tags~="forest"] { color: #7F7F7F; } tw-story[tags~="night"] { color: #F5F50C; }
```