Tips, Tricks, and Best Practices for Twine (Harlowe 3.3.4)

What this document is:

- *A collection of my personal top "I wish I knew that sooner!" tidbits*
- *A quick reference for several beginner-friendly bits of code and examples of their usage*

What this document is not:

- *An in-depth how-to guide for using Twine*
- *A complete manual of all of Twine's capabilities and functions*

➔ ***Visit http://twine2.neocities.org for the complete Harlowe user guide!***

While slightly out of date, https://twinery.org/cookbook/ also contains many great coding examples that mostly still work in the current version of Twine.

For examples of Twine stories/games that I created and co-created, please visit https://sd41blogs.ca/martinj/twine/ - feel free to download and import any of my stories/games into Twine so you can see how they work and adapt any bits of code that you like for your own projects (no guarantees that best practices were always followed!)

TABLE OF CONTENTS

IMPORTANT TWINE VOCABULARY AND CONCEPTS

Closing your brackets & quotation marks in the right order, and "nesting" elements	Coding in twine often uses several types of brackets (like these) and quotation marks 'like these' to define the start and end points of certain pieces of information. **For each left-hand side bracket or quotation mark* that is opened, it needs to eventually be closed again by following it with a matching right-hand side counterpart**, like containers with lids that **must** be closed in order to work. [this will work fine] [this will cause problems if I never add a right-hand bracket to close it… You can "nest" multiple elements inside of other elements (like using "these quotes" and [these brackets] inside of these parentheses), but **each inner element needs be closed before you can close the lid of the larger element surrounding it**, like nesting dolls. {I feel confident saying "everything here [including 'this'!] adheres to the nesting (closing tags in order) principle."} "This example will not work (because I'm closing things in the wrong order.") *to clarify, Twine does NOT use stylized right/left-sided quotation mark symbols like "" and ''. Instead, all quotation marks typed into Twine are rendered as universal quotation marks (like " " and ' ') which do need to follow the opening/closing nesting order described and demonstrated above. This is only an issue if pasting in code that was written/edited in a word processor like Microsoft Word of Google Docs, which automatically convert quotation marks into right/left-sided symbols that won't work with Twine's code.
Story	Twine can be used to make stories, games, digital tools, presentations, and all sorts of other things that defy categorization, but the word **"story" is used as a general term for anything you build with twine.**
Reader	While they might more accurately be called a "player" or "user" depending on what you create, **"reader" is the general term for the intended audience of your story**.
Publish to File	Twine's term for **saving/exporting your story as a file**. Twine stories are published as HTML files, which can be opened using any web browser.
"project"/"project files"	Not a term used in Twine, but one that I use occasionally in this document to refer collectively to **a story plus any other separate file assets that it requires in order to work properly** (such as the image files for pictures).

Format	**Formats are different coding languages for creating stories in Twine**. Each story can only use one Twine format, but you can experiment with different formats in different stories. Twine supports several formats that have distinct affordances and levels of complexity. Sugar Cube is a very popular format for advanced users who want to create especially complex stories.
Harlowe	**Twine's default format** (and the only format covered in this document). Harlowe strikes a great balance of being both beginner-friendly but also capable of great complexity. When searching online to learn how to do something in Twine, be sure to check if the answer given is for the same format as what you are using.
HTML, JavaScript, CSS	These are types of **code used for designing websites and web-based content.** Twine stories are essentially just highly specialized webpages that can be created without knowing anything about these languages, but elements of these languages can be used to further augment your Twine story.
Passage	**A passage can be thought of as a "page" of a book (or a website)**. When creating a story in Twine, passages are visually represented by squares that look like sticky notes. Every passage must be given a unique name, and their names are case-sensitive.
Link	**Exactly like a link on any webpage**, links in twine are clickable text (or images!) that can transition the reader to a new passage and/or trigger some other event to happen. In the Twine editor, regular links that lead to other passages are visually represented by arrows showing the direction of "travel" that the reader can take when navigating the story. If a passage has no links leading to it, it will not be discoverable by someone reading the story. Learn more: **LINKS**
Macro	Macros in Twine are **command codes**: special functions that perform specific tasks. Learn more: **Macros**
Variable	A variable is **a named placeholder that can be assigned different values**. When a story is played, variables are replaced by the value assigned to them. Variable names are created by adding a dollar sign to the start of a word, such as $currentscore (or by using an underscore for temporary variables such as _movementthisturn). Learn more: **Variables**
Hook	A hook is **a "container" using square brackets to group text and/or code together** so that [everything inside the brackets] can be treated as a unit. Learn more: **Hooks**
String	A string is a sequence of plain-text letters/numbers/characters/symbols. This whole description would be considered a string. Inside of macros, strings can be denoted by using either 'single' or "double" quotation marks to show where the string begins and ends. This is an important concept to be aware of when **differentiating between the string "10" (which is just a two-character string containing the symbols 1 and 0) and the actual number 10 (note the lack of quotation marks)** which is understood as a numerical value that can be used in operations such as adding points to a reader's score.
Boolean values	**True/false values** for the logic of computer coding, using operators such as **is**, **is not**, **and**, **or**, **<**, **>**, etc. Learn more: **Boolean values and operators**

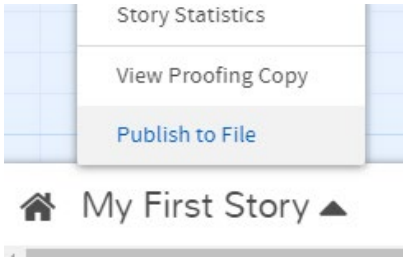
Getting started: Two editors to choose from

There are two ways to use Twine on your computer*. Both versions work the same way and offer all the same features.

BROWSER-BASED (use it online)	DOWNLOAD AND INSTALL ON COMPUTER
No installation required	Requires ability to install software on your device
Always the most up-to-date version	Does not automatically update to latest version
Risk of work being lost if browser data is erased **(manually saving back-ups can prevent this)**	**No unusual risk of work being lost**

**Twine does not work properly on many mobile devices for creating stories. You will likely need to use a computer (Windows, Mac, and Linux are all supported) in order to create a story or game in Twine. However, the story itself will be playable on most mobile devices.*

SAVING AND TRANSFERRING YOUR WORK

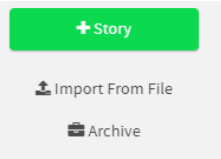


In both the browser-based and downloaded versions, your work is saved locally to the specific device that you started the project on (ie. your work is NOT saved to Twine's website, even when using the browser-based editor). If you want to start a Twine project on one computer and then continue working on another computer, you will need to "Publish to File" in order to export your story/game as an HTML file, and then transfer that file to the other computer using email, USB stick, etc.

Then, you can use **"Import From File"** on the main menu to load your project into Twine on a different device.

If you are using the browser-based version of Twine, using "Publish to File" is important to do periodically to back up a copy of your work to your device. Your internet browser data is not a stable or reliable place for long-term storage of your Twine projects.

If your project uses image files or audio files stored on your computer, you will need to save each story's HTML file into a folder that also contains the other files used in your project, and send the whole folder (compressing the folder to a zip/archive file can be a handy way to do this), as the HTML file by itself will not contain a copy of the other media.



If you want to save, send, or archive all of your stories at the same time, you can archive your whole library from the main menu with the **Archive** button.

COLLABORATIVELY WORKING ON TWINE PROJECTS

Twine does **not** support collaborative simultaneous editing by different users. It also does not easily support taking two separate Twine projects and merging them together.

In order to work on a project collaboratively, group members must either:

- Send a single project file back and forth as each member adds their contributions one at a time; or
- Have each group member make separate self-contained parts of the final project and then copy/paste the names and contents of each of their passages into a table on a service like Google Docs. Then, have a single group member re-copy and paste the other members' contributions into the final project. **Remember that passage names are case-sensitive** ("Library" and "library" are treated as different names)

  Warning!: If you are ever writing/editing Twine content in programs like Microsoft Word, Google Docs, etc., watch out for universal (straight up and down) quotation marks like " " and ' ' being changed into right/left marks like "" and ‘ ’. Text formatting and coding functions in Twine will not work if written with right/left quotation marks like "" or ‘ ’. If this happens by accident, you can use the find/replace functions in either your word editor or in Twine to replace the right/left quote marks with universal ones.

- As a potential 3rd option for more advanced users, it should be possible to use GitHub for Twine collaboration. However, I have not personally tried this.

DIFFERENT WAYS TO SHARE YOUR WORK

- E-mail the HTML file (or a compressed .zip also containing any image and sound files if necessary) to your audience (may not work for mobile devices).
- Publish to your own website.
- Publish to itch.io https://twinery.org/cookbook/starting/twine2/publishing_on_itchio.html (potentially problematic for students' work: not FOIPPA compliant)

Note: some file-sharing services like OneDrive and Teams will not let you upload HTML files. You can sometimes get around this by compressing your HTML file inside a .zip file and uploading that instead.

LINKS

Exactly like a link on any webpage, links in twine are clickable text (or images!) that can transition the reader to a new passage and/or trigger some other event to happen. In the Twine editor, regular links that lead to other passages are visually represented by arrows showing the direction of "travel" that the reader can take when navigating the story (there are also special links that I call "coded links" that do not show up as arrows in the editor even though they will still work for the reader in the story). If a passage has no links leading to it, it will not be discoverable by someone reading the story.

Simple link **[[open the door]]**	Turns the text "open the door" into a clickable link to a passage titled *open the door*. If a passage with that name does not already exist, it will be automatically created.
Named Link **[[open the door->bad ending]]** or **[[open the door|bad ending]]**	Turns the text "open the door" into a clickable link to a passage titled *bad ending*. If a passage with that name does not already exist, it will be automatically created.
Coded Link **(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 also set the $score variable to a value of 0 when the link is clicked.

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

TEXT STYLING MARKUP SHORTCUTS

The Twine editor for Harlowe has an easy interface for adding text styling without needing to type any markup, but sometimes manually typing can be faster and easier for styles that you use frequently.

Styling	Markup code	Result
Italics	`//text//`	*text*
Boldface	`''text''`	**text**
Strikethrough text	`~~text~~`	~~text~~
Emphasis	`*text*`	*text*
Strong emphasis	`**text**`	**text**
Superscript	`meters/second^^2^^`	meters/second^2

Spend some time exploring the options inside the passage editor; Harlowe is capable of all sorts of cool text effects beyond the basic ones on the above table.

If you are using text styling or other macros (such as (text-colour:red)[this macro that turns the text inside this hook red]) to affect a simple link or named link that uses [[double square brackets]], you need to either **add an extra space** in between the open [bracket of the text modification hook and the first pair of [[square brackets for the link, **or** remove the single hook brackets that surround the double-brackets of the link (the double-brackets can count as their own hook).

✓ (text-colour:red)[**[[**passagename**]]**] (with an extra space added between the text-color hook and the link brackets) will successfully turn the color of your link to “passagename” red.

✓ (text-colour:red)**[[**passagename**]]** (with no extra hook brackets) will also turn the link to “passagename” red.

✗ (text-colour:red)**[[**[passagename**]]**] can mis-interpret the intended bracket pairing order and create a new passage called “[passagename” instead of linking to “passagename”.

IMAGES

Harlowe doesn't have its own way of putting images into your story, but we can use HTML code to make them work anyway.

Option 1: local images (pictures on your computer)

To add an image, type or copy this HTML code into a passage:

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

Replace **imagefilename.jpg** with the filename for the image you are using - and remember, it's case-sensitive!

To see how your story looks with its new image(s), use **Publish to File** to save a copy of your story file in the same folder as your image file(s). Then, open this saved file to test your with the images (images on your computer will not be visible when just using the "test from here" or "play" buttons in Twine's editor).

The **width=20%** can be adjusted to any number to change the size of the image relative to the size of the screen it is being displayed on (this method works better across different devices than specifying a pixel size). You can also specify **height= %** instead of (or in addition to) width. Alternatively, you can also specify image size dimensions using **em** (a unit of measurement related to font size), such as width=46em instead of using a %.

Using local images (saved to your computer) is a good way to make sure your story's images will always work.

Images option 2: remotely hosted images (pictures on the internet)

Remotely hosted images are images hosted online (including those that may have uploaded to your own hosting service). You can use that address instead of a filename to display that image inside your Twine story, even when you don't have a copy of that picture saved on your computer.

For example, **<img src="https://static.wixstatic.com/media/2c6bfc_9991defc275247a8b7a1873e42a2dc37~mv2.png" width=50%>** will display an image that my colleague created and uploaded to a free Wix account to be used in a story we made (we used Wix in particular because many other free web file hosting services such as Google Drive, Dropbox, etc. actively prevent "hotlinking" by regularly generating new filenames/addresses for hosted files.). The advantage of this method is that remotely hosted images will work when using "test from here" or "play" in Twine's editor.

It can be tempting to use an image search to find suitable pictures that are already hosted online and to "hotlink" to them in your story by copying their image address into your story's code, but this has several issues:

- Ethical and legal concerns: if you are linking to images you just found online, you are using someone else's image without permission, and syphoning the resources of the web hosting service they pay for.
- The image will stop working in your story if the owner decides to delete or change the name of their logo file
- The image file at the linked address could at any time be replaced with a different image with the same filename.

Alternative 3rd image option: Base64 encoding

I have personally found that this method has more drawbacks than benefits, but if you're interested in learning more about this you can read all about it here:

https://damonwakes.wordpress.com/2019/05/30/twine-for-beginners-adding-images-as-base64/

SOUNDS

Audio is something that Harlowe does not do very well. There are two ways of playing sounds in Harlowe, and neither one works on all devices all the time.

THE ADVANCED WAY:

If audio is super important for your story, you should check out the 3rd-party enhancement called **HAL (the Harlowe Audio Library)** at **https://twinelab.net/harlowe-audio/#/**

The HAL website contains excellent step-by-step instructions for how to add and use HAL for audio in your stories. HAL's audio playback may not work if reading your story on certain mobile devices.

The older, simpler way to have a sound play when viewing a passage is to use this code at the very BOTTOM of a passage:

<audio src="filename.mp3" autoplay>

This method has limitations:

- It only works with .mp3 files (not .wav).
- The audio will stop playing when the reader transitions to a different passage (does not allow for background music that plays continuously as the player navigates the story).
- It often does not work if used in the the very first passage of your story.
- It may not work at all on some devices/browsers.

VIDEOS

If you have a video file in your project folder, you can use the following HTML code:

<video controls width="450"> <source src="moviefile.mp4"></video>

And adjust the width and filename as needed.

Alternatively, you can upload your video to a hosting service like YouTube, and then use the **Share** button and select the **embed** option. This will provide you with a block of code that you can copy and paste into your Twine passage to place the YouTube video directly in your story.

CODING: MACROS, HOOKS, VARIABLES, and BOOLEANS

Macros in Twine are special functions perform specific tasks. These are the engines of Twine's coding capabilities. Harlowe's macros are enclosed by parentheses with a colon following the name, like (this:). The **coded link** example in the above table uses two macros: first the (link:) macro creates clickable text ("open the door") that when clicked will activate the contents of the attached hook. When clicked, the hook contains a (go-to:) macro that and the (go-to:) macro inside the first macro's hook. Common macros include (set:) to define a variable, and (if:), (else-if:), and (else:) to display different content if certain conditions have been met. Macros themselves are not visible to the reader, but their effects can be.

Hooks are chunks of text or code that have been "packaged" together inside single [square brackets] in order to treat everything inside the brackets as a "unit" that can be acted upon by a macro. The brackets that surround a hook are not visible to the reader, but their content will be (unless affected by a macro that hides the hook unless a predetermined condition is met).

Macros can be placed inside of hooks, such as in the coded link example above.

Be sure to keep track of closing your) and] brackets in the right places. In Harlowe, Twine has some useful visual cues to help track this such as shaded highlighting for text contained within brackets, and the ability to click on any text to underline the start and end of that particular container.

Variables are special stand-in values, represented by a keyword of your choice that starts with a dollar sign such as $playername, that can be changed to represent any other value. You can define a variable using the (set:) macro as shown below. A variable's value can be re-defined as many times as you like. If a variable is written in a passage without having been defined using a macro like (set:), Harlowe will assign it a default value of 0.

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

Macros, hooks, and variables all work together. **(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 will be displayed, showing the text "You win, Alex!" (assuming that the reader was earlier given the opportunity to set their $playername to Alex). If the $score variable is not exactly 5, the contents of that hook will not be displayed.

Important considerations for variables:

Numbers: If you are setting a variable's value to a number that you will want to use in math operations (such as adding points to a score, or determining has collected enough money to purchase an item), it is important that you **do not use quotation marks around the number** (see the score example in the table above).

> 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. This is fine if you just want to code **your current speed is $speed** and have it rendered as "**your current speed is 30**", but you will get and error if you try using it in operations like (set: $speed to it +5) or (if: $speed > 50)[You are going too fast!]. Writing (set: $speed to 30) will let you do all 3 of the above examples successfully.

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

Undefined Variables: If you display a $variable in your story that has not yet been defined with (set:), that variable will be given a default value of 0.

Temporary Variables: Another version of variables that can be useful in some circumstances is the _temp variable. The value of a temporary variable, created by using an _underscore instead of a $dollar sign, is only stored for as long as the reader is viewing the current passage in which it was defined and then will be "forgotten" once the reader navigates to a new passage.

Redefining Variables: If you use the (set:) macro to change a variable's value again after it has already been used, the previous uses of that variable will **not** automatically "refresh" or update to reflect the new value. For example, if the $currentlyholding variable is already set to "a flashlight" and the very start of the passage says "You are holding $currentlyholding", the reader will see "You are holding a flashlight". If later in the same passage, the $currentlyholding variable is then re-defined to the new value "an umbrella", the text that the reader sees at the top will still say "You are holding a flashlight". However, any instances of $currentlyholding **after** it is re-defined will give the new value of "an umbrella". If you want to have a variable that automatically updates on the page when it is set to a new value you can either use the (live:) macro on the variable or combine the (set:) variable with a (go-to:) variable targeting the current passage to re-load the page immediately after the new value is set.

Boolean values and operators

"Boolean" refers to true/false logic of computer programming. Like numbers, the values **true** and **false** can be used in Harlowe without quotation marks. Using **operators**, one can use macros like **(if:)** to code parts of a story that only happen if a certain true/false condition is met.

The following table of Boolean operators is from the Harlowe manual: https://twine2.neocities.org/#type_boolean

Operator	Purpose	Example
is	Evaluates to true if both sides are equal, otherwise false.	$bullets is 5
is not	Evaluates to true if both sides are not equal.	$friends is not $enemies
contains	Evaluates to true if the left side contains the right side.	"Fear" contains "ear"
does not contain	Evaluates to true if the left side does not contain the right side.	"Fear" does not contain "eet"
is in	Evaluates to true if the right side contains the left side.	"ugh" is in "Through"
>	Evaluates to true if the left side is greater than the right side.	$money > 3.75
>=	Evaluates to true if the left side is greater than or equal to the right side.	$apples >= $carrots + 5
<	Evaluates to true if the left side is less than the right side.	$shoes < $people * 2
<=	Evaluates to true if the left side is less than or equal to the right side.	65 <= $age
and	Evaluates to true if both sides evaluates to true.	$hasFriends and $hasFamily
or	Evaluates to true if either side is true.	$fruit or $vegetable
not	Flips a true value to a false value, and vice versa.	not $stabbed
matches	Evaluates to true if one side is a pattern or datatype describing the other.	boolean matches true
does not match	Evaluates to true if one side does not describe the other.	boolean does not match "true"
is a, is an	Evaluates to boolean true if the right side is boolean and the left side is a boolean.	$wiretapped is a boolean
is not a, is not an	Evaluates to boolean true if the right side does not describe the left side.	"Boo" is not an empty, 2 is not an odd

USEFUL CODE EXAMPLES

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 the address of 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 decreasing the value of the $hotdog variable by 1 and increasing the value of the $pizza variable by 1 (assuming these are numerical values)
(go-to:)	(go-to:"library")	On its own, the (go-to:) macro will cause the story to immediately advance to the passage called **library** without any interaction from the reader. This can sometimes be useful if you want to have an "invisible" in-between passage for running macros before the reader reaches the next "real" passage.
	(link:"go to the library")[(set:$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 reader to the destination passage when the link is clicked.
(print:)	(print:"$5")	**$5** will be written (as a string)
	(print:"$"+(str:$money))	(assuming $money is set to the value 3) **$3** will be written (first the string "$", then $money's value of 3 that has been converted into the string "3")
	(print:$money+5)	(assuming $money is set to the value 3) **8** will be written (the value of $money, 3, is added to 5, and the resulting value is printed)
(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.

ADDITIONAL USEFUL CODE: Stylesheet

*To add the following features to your story, click on the "**Story**" button at the top of the editor, select "**Stylesheet**", and then copy/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 in Test mode

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

Change background color, font type, font size, font color, and link color for the entire story

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; }
```

FINAL TIPS

Twine does not contain spell-check. You may want to run your prose through a word processor to check for errors!

Google is your friend. Search for "how do I _ in Harlowe" will lead you to exactly the code you need 95% of the time.

TEST your story/game frequently.

The **Debug View magnifying glass icon** is immensely helpful for figuring out why errors are happening in your story (or why it is not functioning the way you expect, even if no errors are generated). When you use **"test from here"**, a magnifying glass will appear next to any errors which, if clicked, will show you a step-by-step process of how the program attempted to execute your code before it encountered something it couldn't handle. The **Debug View button** on **the test from here** control panel will produce a similar magnifying class icon for all pieces of code.

The **backward/forward arrows on the left side of your twine story act as undo/redo buttons** for the reader, not navigation links. 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.

The "undo" button is limited. If you close a passage after making changes, the changes will be saved and cannot be automatically undone when you re-open the passage again. Before making changes to a large/complicated passage, consider copying and pasting a copy of that passage's contents into a new "backup" passage first, in case you accidentally "break" something and want to restore the passage back to the way it was. Make frequent backup files of your whole story (especially before making any big changes).

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

Headers/Footers: To make the same thing appear at the top of bottom of every page (like a status bar showing the reader's progress towards a goal) create a new passage and use the **+ tag** button. Passages with the tag name **header** will appear as a header at the top of every passage in your story/game. The same thing works with the name **footer** to have the a passage's contents appear at the bottom of all passages.

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!