Inputts

Inputts is a *"kinda game engine"* I made to make html text adventure games. It has things that I think a text adventure game would be nice with. It is currently in a single html file but I highly recommend creating an external js file for the actual game thing (more on that later).

This is just for windows and screens with a width of 616px or more for now because it still kinda messes up.

Components

Interface

This the default page you're going to see when opening the html file. It has two main divs:

1. Output
2. Input

The output box is the one on top. It is where things get displayed when using *outpt()*. It has overflow-y on scroll and x on hidden. The height of this box depends on the height of the input box so I made it that when a key is pressed, the height adjusts.

The input box is the one below. It does not use a *"form thingy"* but instead uses an EventListener for input. It detects most keyboard inputs and shift also works but caps lock is entirely ignored. Ctrl + Backspace deletes all of the input box. The spaces are * *'s

Text are the text. Arrows are the things at the left of them.

The entire html that you see at first:

```
<body id="body" onload="loaded()">
    <div id="outptbx"></div>
    <div id="textbox">
        <div class="arrow">></div>
        <div class="txt"><span id="inpt"></span><span id="cursor"> </span></div>
    </div>
```

Javascript

The javascript is a bit messy and unconventional (i'm kinda new here) so if you're struggling to understand it or you don't want to go through all of that code, you can read this. This already has jquery and google fonts plugged in.

> Recommended Additional Resources
> W3Schools Javascript
> Strings
> HTML DOM

Variables

```
//Variables
const inptbx = document.getElementById("inpt");
const otptbx = document.getElementById("outptbx");
const bdy = document.getElementById("body");
const titl = document.getElementById("title");
const ficon = document.getElementById("icon");
const pntr = document.getElementById("cursor");
var contnt = [];
var lineno = {num: 0};
var cssvar = document.querySelector(':root');
var keysPressed = {};
var textAnim = {array: [null], anims: 0};
var updwntxt = {no: lineno.num + 1, store: "", fi: true};
var crsrstate = true;
var loadstate = 0;
var loadedst = false;
var customcursor = {on:" ",off:" "};
```

inptbx - The span where the input is showed.
otptbx - The whole output box.
bdy - The whole body.
titl - The Page Title Element
ficon - Favicon
pntr - Input box cursor
contnt - Contains the text displayed inside the output box. However, since the content of each line can be changed, it can not be trusted sometimes. Each line is stored in one array cell. The first cell (*contnt[0]*) should always be undefined/empty.
lineno - Tells how many lines are there in the output box.
cssvar - Helps in the changing of css variables.

keysPressed - Contains the list of keys pressed, used mainly for the rejection of input when the control and/or alt key is being pressed.
textAnim - Used in carrying out text animation functions
updwntxt - Used in carrying out *arrowtxt(up)*
crsrstate - Is the cursor on or off
loadstate and *loadedst* - For cursor function
customcursor - custom cursors for when *gmst.cursor* is *"c"*

> When reset, *contnt*, *lineno*, *textAnim*, and *updwntxt* become their initial values again.

gmst

gmst is Game State. It determines the behaviour of the functions and/or the game.

```
//States
var gmst = {
        input: true,
        eyesopen: false,
        enterblank: false,
        loadlock: true,
        lights: true,
        enterdef: true,
        pOutpt: 1,
        cursor: 1,
        cursorRate: 500,
        bodybold: false,
        bodyital: false,
        bodyundr: false,
        inptbold: false,
        inptital: false,
        inptundr: false,
        otptbold: false,
        otptital: false,
        otptundr: false,
}
```

input - Keypress detection
eyesopen - Input box text detection
enterblank - Allowing player to input empty lines
loadlock - The blocking of the keypress detection when the cursor is at a loading state
lights - Whether the body is visible or not
enterdef - Whether or not *pEnter()* does the default code or not
pOutpt - What type of output your player has is *pEnter()* is on default

State	Output
1	> Text
2	< Text
3	$ Text
4	- Text

State	Output
5	= Text
6	Text
7	Text
8	1 Text
9	[Custom] Text

cursor - Cursor state/appearance

State	Cursor
-2	Loading Animated (|/-\)
-1	Loading Animated (...)
0	Blank/Hidden
1	|
2	_
3	▉
"c"	*customcursor*

cursorRate - How fast the cursor blinks in ms
bodybold | *bodyital* | *bodyundr* - Body font styles
inptbold | *inptital* | *inptundr* - Input box font styles
otptbold | *otptital* | *otptundr* - Output box font styles

IDs

> x = Line No.

Each line in the output box has different elements with different ids.

```
<div class="container" id="line-1"> == $0
  <div class="arrow" id="arwln-1">> </div>
  <div class="txt" id="txtln-1">roewe</div>
</div>
```

line-x - the whole line
arwln-x - the arrow
txtln-x - the text
Media in each line has ids too
imgln-x - image
vidln-x - video
audln-x - audio

> When they are created with the same lineno, it becomes *imgln-x-y*. y is 1 if it is the second img.

Functions

First of all, delays. Unfortunately there is no sleep function so just use:

```
setTimeout(() => { "[code]" }, "[delay in ms]");
```

Body On Load

loaded() - Function that occurs in *body onload*

Enter

pEnter() - On default, it deletes the text in the input and outputs. You could change what happens though

Output Box Scroll

scrldwn() & *scrlup()* - Scrolls output box

Output

outpt(x, y, z) - x = text, y = output type (see *pOutpt* table, default is 6), z = custom number for output type 5 and custom 2 character string for output type 6
outpttxt(x, y, z) - Basically *outpt(x, y, z)* but it returns the text that goes into the line (Line No. is 1 + current lineno)
setupanim (x, y) - Creates an animation that makes it so that the text displays one character at a time, x = string, y = delay per character in sec
playanim(x, y, z1, z2) - Plays an animation that has already been setup in the output box, x = animation number, y = Line No. (if there is no such line in the output box, it'll create a new line), z1 = string that comes before the animated text (is not animated), z2 = string that comes after the animated text (is not animated)

Files

> Line No. for these is 1 + current lineno

outary(ary, spc) - Turns an array of strings into a string you can display, ary = array of string, spc = the thing you put in between each item (leaving it blank will make it "")
outpti(link) - outputs an image
outptv(link, text, loop, btn) - outputs a video, *btn* (boolean) play button, text = additional text after the video (and button), *loop* (boolean) whether or not it loops
outpta(link, text, loop, btn) - outputs a video, same input fields as video

Controls

> x = Line No.

Image

correctimg(x) - Corrects the img size
changesrci(x, y) - y = new img src
imgbordeR(x,y) - Change image border radius, x = Line No., y = border-radius
bgimg(x) - Change background image, x = background-image property (bgimg uses url())
changeIcon(x) - Change the favicon of the page, x = new favicon link

Audio

playaud(x) - Plays the audio file
pauseaud(x) - Pauses the audio file
resetaud(x) - Restarts the audio file
settimeaud(x, y) - Sets audio time, y in secs
lengthaud(x) - Gives audio length
changesrca(x, y) - y = new audio src

Video

playvid(x) - Plays the video file
pausevid(x) - Pauses the video file
resetvid(x) - Restarts the video file
settimevid(x, y) - Sets video time, y in secs
lengthvid(x) - Gives video length
correctvid(x) - Corrects the video size
changesrcv(x, y) - y = new video src

Reset

resetBox() - Deletes all output box text, resets contnt and lineno
resetAllFormat() - Resets the text format of body, output, and input boxes (not including font family and font size)
turnlights() - Flips the state of *gmst.lights* and turns of visibility of body

Text

getInnerText(x) - Get the innerHTML of a line of text in the output box, x = Line No.

Changing Text

cinpt(x) - Changes input box text, x = new text, if x = undefined, text will be just deleted
ctxt(x, y) - Changes text in output box, x = new text, y = Line No.
carw(x, y) - Changes arrow in output box, x = new arrow, y = Line No.
inptarw(x) - Changes arrow in input box, x = new arrow
rplaceall(x, y, z) - Replaces all instances, x = string, y = search, z = replace
changeTitle(x) - Change the title of the page, x = new page title

Color

> x = color

maincolor(x) - Changes color of body (default = black)
accentcolor(x) - Changes color of body (default = white)

> x = Line No., y = color

linecolor(x, y) - Changes color of whole line in output box
textcolor(x, y) - Changes color of just the text in line in output box
arrowcolor(x, y) - Changes color of just the arrow in line in output box
iarwcolor(y) - Changes color of the arrow of input box
itxtcolor(y) - Changes color of the text of input box
cursorolor(y) - Changes color of the cursor of input box

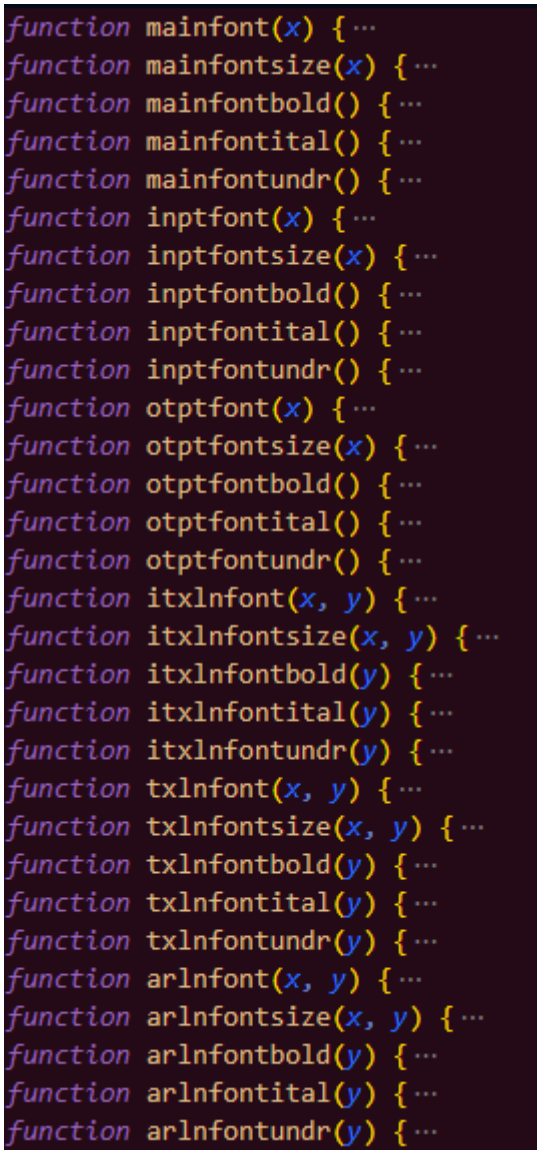
> x = text, y = color

colortext(x, y) - Outputs colored text

> All resets with *resetBox()*
> the color can be anything just make sure to format it correctly (i.e. having the # for hexcodes)

Fonts

> x = the thing needed, y = Line No.



> *itxln* (whole line) vs *txln* (just the text)

font - font family
fontsize - font size
bold - bold
ital - italicize
undr - underline

> x = text

boldtext(x) - returns bold span
italtext(x) - returns italicized span
undrtext(x) - returns underlined span

and Spaces

rmvspace(x) - Removes all spaces
rmvespace(x) - Turns multiple spaces into one
rmvnbsp(x) - Removes all * *s
rmvenbsp(x) - Turns multiple * *s into one

nbspspc(x) - Turns * *s into spaces
spcnbsp(x) - Turns spaces into * *s

> x = string

Key Press

This is the event listener. Wanna make changes? Look here!

```
532 >       document.addEventListener('keydown', (event) => {···
535 >       document.addEventListener('keyup', (event) => {···
538 >       $(window).blur(function(){···
541 >       document.addEventListener('keydown', (event) => {···
```

1. Adds to *keysPressed* the keys pressed
2. Removes keys in *keysPressed* when they're not being pressed
3. Resets *keysPressed* when page is out of focus
4. This detects keyboard inputs and adds them to the input box

arrowtxt(up) - Lets you change input box text with output box text

Blinking Cursor

cursor() - This basically controls the input box cursor

Input Detection

> x is a string provided by the system

ears(x) - Gets entered input (happens before *pEnter()*)
eyes(x) - Gets text in input box before it is entered

Game

dice(x) - Returns a number from one to *x*.

localStorage & sessionStorage

There is no function for this but this is how to do it:

- *localStorage* does not expire when you exit the browser but *sessionStorage* does.

1) *.setItem()*

```
localStorage.setItem("mytime", Date.now());
```

This creates and stores an item. The first value is the *keyname* and the second is the *keyname*'s *value*. Try not to have spaces in the *keyname*.

2) .getItem()

```
var x = localStorage.getItem("mytime");
```

X is now the *value* of the inputted *keyname*.

3) .removeItem()

```
localStorage.removeItem("mytime");
```

Removes the *keyname* and its *value* from storage.

4) key()

```
var x = localStorage.key(0);
```

This makes storage act like an array. x is now the *keyname* of the first local storage item.

5) length

```
var x = localStorage.length;
```

x is now the number of locally stored items.

6) clear()

```
localStorage.clear();
```

Removes all local storage items.

7) JSON.stringify()

```
var obj = { "name":"John", "age":30, "city":"New York"};
var myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
```

```
myJSON = “{"name":"John","age":30,"city":"New York"}”
```

Syntax :

```
JSON.stringify(obj, replacer, space)
```

Parameter	Description
obj	Required. The value to convert to a string
replacer	Optional. Either a function or an array used to transform the result. The replacer is called for each item.
space	Optional. Either a String or a Number. A string to be used as white space (max 10 characters), or a Number, from 0 to 10, to indicate how many space characters to use as white space.

> This also works with arrays.

8) *JSON.parse()*

```
var obj = { "name":"John", "age":30, "city":"New York"};
var myJSON = JSON.stringify(obj);
newMyJSON = JSON.parse(myJSON);
```

```
newMyJSON = { "name":"John", "age":30, "city":"New York"}
```

> This also works with arrays.

Resources

> More Resources
> W3Schools Storage
> W3Schools Javascript JSON

Game Making

The game making part really just have two functions built in. However, I suggest having a saved responses and stages variable. I also suggest doing this in a separate js file and linking it below the script tag.

Future Plans

- [ ] Planning on making the cursor movable
- [ ] Better function naming
- [ ] Developer version