INKR Tool by Mikhail — Documentation

INKR is designed to allow a user to get smooth, free-flowing and beautiful animated cursive text right in the HTML based on simple text input (pre-configured or dynamic).

The goal of ***INKR*** is to create the aesthetic & feeling of custom written cursive text in your webpage.
The previously available ways to implement anything of this sort, whether with free or premium tools, were not only too rigid, but could not create the final product necessary.

INKR relies on the Filmotype LaSalle font designed by Patrick Griffin and takes advantage of some of the technical benefits it provides, which will be briefly expanded upon below.

1. PRE-REQUISITES & NOMENCLATURE

INKR requires SVG segment files, multiple for each character, and currently supports ***76*** pre-selected UTF-16 characters:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 ! " & ' ( ) , - . / : ; ? [space]

Every SVG segment file was hand-crafted by Mikhail and contains other data necessary to operate correctly, such as positioning and 'viewBox'.

For ease-of-use, compatibility and maintainability, ***INKR*** does NOT use in-line SVG data in the HTML.
The SVG files must be located in a pre-determined path and have the pre-determined nomenclature to be correctly injected in the HTML.
INKR also relies on a single CSS file and a single JS file to function.

The default paths look like this:	root/typing.js	- JS file that performs the logic
	root/css/typing.css	- Folder path of CSS file
	root/typing/a_lower_1.svg	- Folder path of SVG segment files
	root/typing.html	- Can be used as an iframe or substituted

The SVG segment files use the following naming conventions (segment count in examples is random):

a_lower_1.svg	- lower-case letter A	segment 1
m_upper_6.svg	- upper-case letter M	segment 6

The above names are generic and applicable for all letters. The amount of segments is between 1 and 7 depending on the character. The remaining names are more specialized:

zero_1.svg	- number 0	segment 1	(All other numbers also follow this convention)
excl_3.svg	- exclamation mark	segment 3	
quote_2.svg	- quotation marks	segment 2	
amp_1.svg	- ampersand	segment 1	
apos_1.svg	- apostrophe	segment 1	
par_open_1.svg	- parenthesis opening	segment 1	(parenthesis closing is par_close_1.svg)
comma_1.svg	- comma	segment 1	
dash_1.svg	- dash	segment 1	
dot_1.svg	- dot	segment 1	
slasH_1.svg	- slash	segment 1	
colon_2.svg	- colon	segment 2	
semi_2.svg	- semi-colon	segment 2	
quest_1.svg	- question mark	segment 1	
			Note: The 'space' character does not have any SVG files.

There is a total of ***292*** segments by default. Each segment has its own associated animation data and nomenclature inside the CSS file:

SVG FILE	CSS CLASS	ANIMATION
a_upper_1.svg	a_upper_1_draw	a_upper_1_anim
four_3.svg	four_3_draw	four_3_anim

2. OPERATION PIPELINE

The input is stored in a string constant '***userInput***'. This can be pre-determined or set through an HTML form submission, for example.
The CSS class '***.charParent***' is hydrated as the constant '***charParent***'. It is initially responsible for an empty HTML Div that will be dynamically populated with the appropriate features.
The constant '***charHeight***' is a height-reference for every segment/character and is set to a default 50 (pixels); it tweaks the size of all characters and can be edited.
The constant '***emHeight***' is similar to the above but is instead measured in 'font units' and is by default 1505. IT IS RECOMMENDED TO NEVER CHANGE THIS.

5 new JS Map structs are initialized - '***charOf***', '***nameOf***', '***charWidth***', '***advWidth***', '***lbWidth***'.
'***charOf***' keys are strings of segment names, values are UTF-16 characters. ***charOf.get ("a_upper") = "A"***
'***nameOf***' is an inverse map of '***charOf***'. The reason it is initialized alongside '***charOf***' is performance based. ***nameOf.get ("A") = "a_upper"***

The remaining 3 maps follow the convention: ***key*** = UTF-16 character ***value*** = (***charHeight/emHeight)* * [pre-determined 'font units' measurement]***

'***charWidth***' stores the width of a character. '***advWidth***' stores the advance-width of a character. '***lbWidth***' stores the left-side-bearing width of a character.

If you are not familiar with these technical terms of font glyphs, know that they are integral towards determining the correct size, placement, spacing / kerning of all characters.
The values are replicating some of the basic functionality of fonts & glyphs - IT IS RECOMMENDED TO NEVER CHANGE THEM.

The '***charParent***' HTML div's '***innerHTML***' property is programmatically edited via string manipulation: ***charParent.innerHTML = populateHTML(userInput);***
The function '***populateHTML***' takes the string input, and feeds each character to the function '***partSpawn***': ***output+=partSpawn(character);...;return output;***
The function '***partSpawn***' takes each character and ***1.*** internally determines how many segments it must have
(eg: upper-case A has 5, lower-case S has 3, comma has 1);
2. uses the '***nameOf***' map to find correct the nomenclature;
For example, the character "a" has 6 segments and partSpawn("a") returns a string of HTML:

```
"<div class="char a_lower">
        <img class="a_lower_1"  src="/typing/a_lower_1.svg">
        <img class="a_lower_2" src="/typing/a_lower_2.svg">
        <img class="a_lower_3" src="/typing/a_lower_3.svg">
        <img class="a_lower_4" src="/typing/a_lower_4.svg">
        <img class="a_lower_5" src="/typing/a_lower_5.svg">
        <img class="a_lower_6" src="/typing/a_lower_6.svg">
</div>"
```

Every div that has the class '***.char***' (all character divs) is set to a height of (***charHeight * 0.6***)px.
This does NOT determine the height of the characters, merely the divs, and is responsible for vertical line spacing.

Every character div's width is determined dynamically through ***'setWidthAll(node,char)'*** (The node is an HTML Node Element). The char's width is found through ***charWidth.get(char)***.
In this way, each possible character is hydrated and set. Example: ***const y_lower=document.querySelectorAll(".y_lower"); setWidth(y_lower,"y");***

At this time, each character has been spawned in the HTML and given the appropriate height & width based on '***charHeight***'. Now they must be spaced/kerned appropriately through some logic that uses ***maxW=document.body.clientWidth****0.98 and the 3 width Maps. We apply a visual offset ('***right***' CSS style property) to a specific div to maneuver it in space.
This possesses the limitation of text performing a newline far before it runs out of visible space on-screen. The function responsible for this behavior is ***setTail(charParent.children)***.

Animation is performed by modifying the clip-path of an SVG segment with a CSS '***animation***' property pre-configured with unique values for each segment in the CSS file.
For simplicity, the clip-path is based on a circle SVG with a specified center-point & an expanding radius. It looks like this:

```
.g_lower_4      { position: absolute;               clip-path: circle(    0% at 39% 68%);}
.g_lower_4_draw{ animation: 0.1s g_lower_4_anim; clip-path: circle(65.9% at 39% 68%);}
@keyframes g_lower_4_anim{
          0%{ clip-path: circle(    0% at 39% 68%);}
        100%{ clip-path: circle(65.9% at 39% 68%);}}
```

All segments start completely hidden, the are revealed through animation, and then remain visible. This is synchronized through the asynchronous function ***animateHTML(charParent)***.

charParent.children is turned into an array. For every ***element*** of this aray, its ***element.children[i].classList[0]*** is a CSS class name, such as '***a_lower_1***'.
By adhering to nomenclature, we ADD another CSS class to it: '***a_lower_1_draw***'. The animation will then be hooked to the segment.
We then find the CSS style property value '***animation-duration***' for every animation and use it with '***await animDelay()***'
to sync every segment & animation together, one after the other.

All of the logic can hypothetically be transfered to a back-end, but for simplicity's sake & performance, it is ran client-side.