**Online Code Editing and Execution**

**Dave-Congwu Tao and Clifford A. Shaffer**

We document a number of ways to implement support for online editing and execution of programs in various programming languages.

1. **JavaScript: Save and recopy from server-side file**

Figure1 shows the interface of the HTML & JavaScript Editor. The left side is a textarea. Anything that a browser can understand can be put in it, because the next step will be simply to feed the text to the browser for display. If the “Run” button is clicked, the outcome will be displayed in the right side. This simply takes whatever is in the left side panel, saves it to a file on the server, then redisplays the contents in the right side panel. Initially, the contents of the page is read from a file on the server.

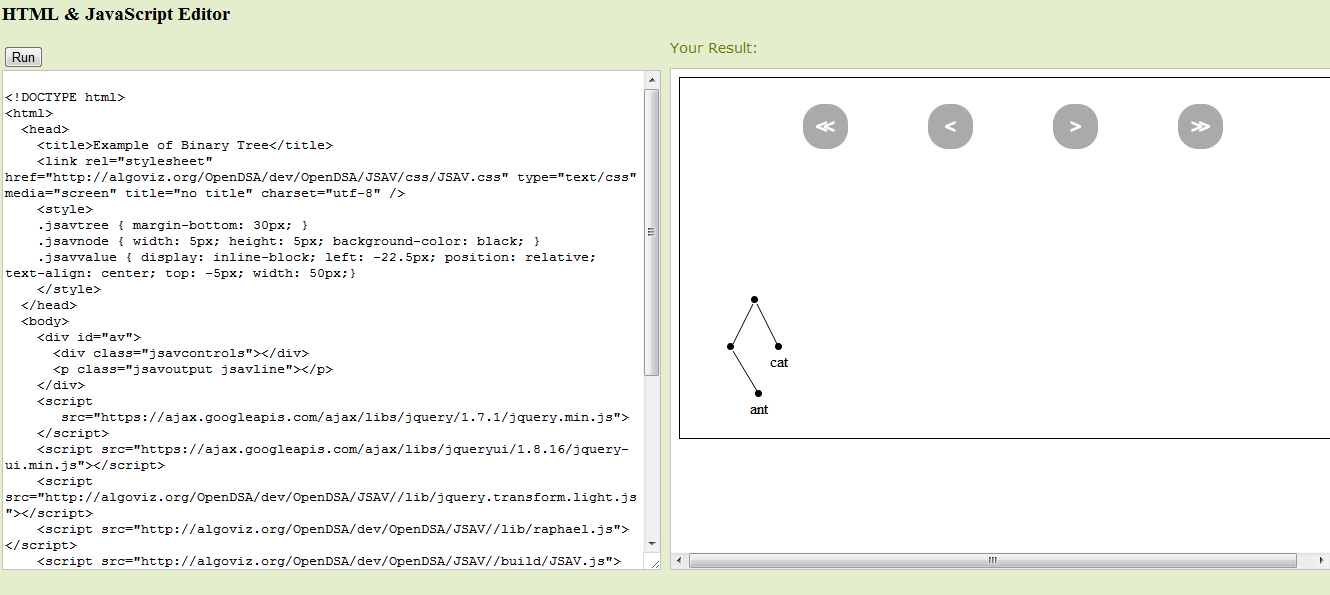


Figure 1: the interface of HTML & JavaScript Editor

The code is composed of the following files:

The “tryit” filefolder includes 3 files:” tryit.css”, “tryit\_view.html” and “convert.php”.

“index.php” includes a form(the textarea) and a iframe(Its source is the “tryit\_view.html”). When the index.php is loading, the code of the “tryit\_view.html” will be displayed in the left side-the “textarea”, and the outcome of running the “tryit\_view.html” will be presented in the right side. If any changes of the code in the left textarea are made and the “Run” button is clicked, the outcome of the running code will be showed in the right side.

**Index.php:**

<?php

header( 'Expires: Mon, 26 Jul 1997 05:00:00 GMT' );

header( 'Last-Modified: ' . gmdate( 'D, d M Y H:i:s' ) . ' GMT' );

header( 'Cache-Control: no-store, no-cache, must-revalidate' );

header( 'Cache-Control: post-check=0, pre-check=0', false );

header( 'Pragma: no-cache' ); //clear the cache

$file\_name = "./tryit/tryit\_view.html";

$file\_pointer = fopen($file\_name, "r"); //read the file

$file\_read = fread($file\_pointer, filesize($file\_name));

fclose($file\_pointer); //close

The most important code of index.php is shown above. First it clears the caches, second it gets the source code from tryit\_view.html on the server.

**Convert.php**

Convert .php attempts to store the modified source code. The important code is as follows:

<?php

$content = str\_replace("\\","",htmlentities($\_POST["content"]) ); //get the content

$content1 = html\_entity\_decode($content);

$file\_name = "tryit\_view.html";

$file\_pointer = fopen($file\_name, "w"); //write the content to a file

fwrite($file\_pointer, $content1);

fclose($file\_pointer); //close

?>

PHP functions “str\_replace()”,”htmlentities()” and“html\_entity\_decode()” process the soure code which comes from the front-end webpage, and store it in “tryit\_view.html”.

This approach has a serious shortcoming. The edited version can only be saved on the server side if the file has world write permission, which makes for a security threat. In our posted instance, we don’t allow the file to have world write permission. So the effect of making any changes will be wiped out when the file is read again (in its original form because the update attempt fails).

1. **Real-time HTML & JavaScript Editor**

The “real-time” HTML & JavaScript Editor htmlJS\_Editor.htm is composed of two textareas. There is no “Run” button because the display panel is updated on-the-fly. Again, the text is simply rendered by the browser.

There is just only one HTML file.The following lists the important Javascript functions in the file.

Function init()

{

window.editbox.document.write(editboxHTML);

window.editbox.document.close();

window.editbox.document.f.ta.value = defaultStuff;

update();

}

function update()

{

var textarea = window.editbox.document.f.ta;

var d = dynamicframe.document;

if (old != textarea.value) {

old = textarea.value;

d.open();

d.write(old);

if (old.replace(/[\r\n]/g,’’) == defaultStuff.replace(/[\r\n]/g,’’))

d.write(extraStuff);

d.close();

}

window.setTimeout(update, 150); // Self-fresh every 150ms

}

In the above code, Window.setTimeout(update, 150) will self-fresh the botton textarea every 150ms.

1. **Processing.js Editor**

**(**Processingjs\_Editor.html**)**

The Figure shows the interface of the Processing.js Editor: it includes a “textarea” and six buttons. The four buttons “Blue Rays”, ”Drawing Board”, ”Star Painted” and “Moving Spotlights” will initiate specific processing demonstration codes.



The code is composed of the following files：



The css: SubEdit/css/js\_style.css

The js: SubEdit/js/l10n.js

SubEdit/js/comment-reply.js

SubEdit/js/processing-1.1.0.min.js

The processing codes are interpreted by the processing.js library.

The tag “canvas” is used in the code like:

<canvas id="canvas1" style="border: 1px solid black;" tabindex="0" width="640" height="400"></canvas>

All processing code should be put in a tag like this:

<script type="application/processing"> </script>

Example : <script type="application/processing" id="drawing\_board">// The Minimal Drawing Board

void setup() {

size(640, 400);

background(0);

}

void draw() {

point(mouseX, mouseY);

mousePressed ? stroke(230,250,120) : stroke(60,180,255);

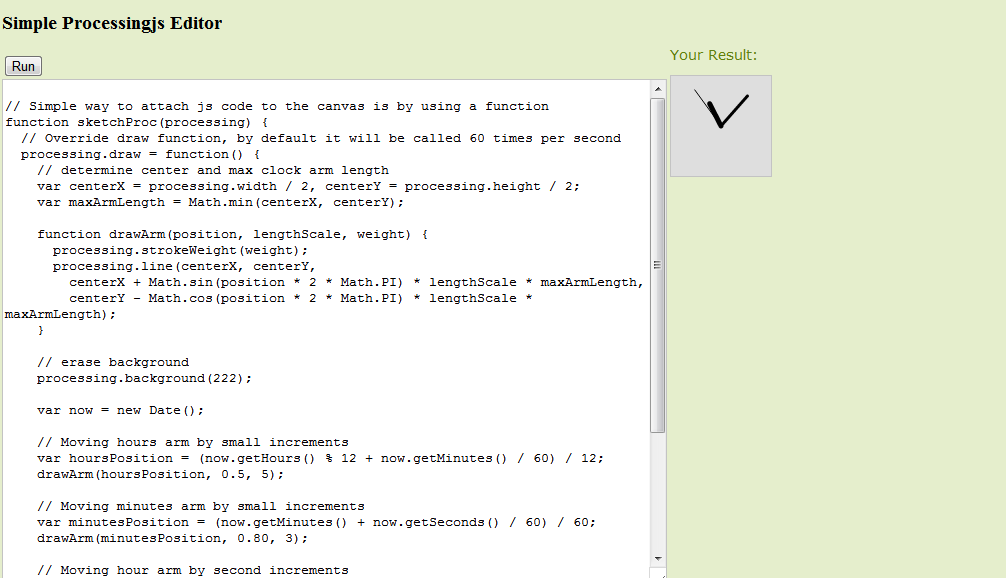
}

</script>

1. **Simple Processing.js Editor**

**(**[SubEdit/SimpleProcessingjs\_Editor.htm](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/SimpleProcessingjs_Editor.htm)**)**

The Figure show the interface of another Processing.js Editor. It includes two parts. The left side has a textarea and a “Run” button, and the right side is a “canvas” area.



The code is composed of the following files：



The css: [tryit/tryit.css](http://algoviz-beta.cc.vt.edu/Processing/tryit/tryit.css)

The most important code of the file “SimpleProcessingjs\_Editor.htm” is using the jquery.js and processing.js. The sample codes are applications of the API of processing.js and jquery.js.

The difference between the Processing.js Editor and the Simple Processing.js Editor is use of the JavaScript function “eval()” to implement the processing.js code in the textarea rather than uses the API of processing.js.

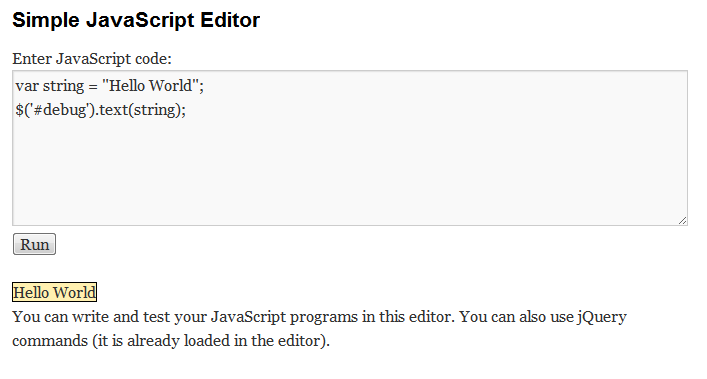
Example:

<input type="button" name="button" id="button" value="Run" onclick="eval(text.value);" /> // use the function eval() to implement the code

# 4. Simple JavaScript Editor

# ([SubEdit/SimpleJS\_Editor.htm](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/SimpleJS_Editor.htm))

The Figure shows the interface for a simple Javascript editor. It is composed of a textarea and a “Run” button. It just can run some simple functions of javascript. The jQuery commands can be used too.



The code is composed of the following files：



The css: SubEdit/css/js\_style.css

The js : [SubEdit/js/comment-reply.js](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/js/comment-reply.js)

SubEdit/js/l10n.js

The sample codes are applications of the API jquery.js and some functions of Javascript. It also uses the function “eval()” to run the code in the textarea.

Example : <script type="text/javascript" language="javascript">

function run\_code() {

var code = document.getElementById('code').value;

try {

eval (code); // use the function eval() to implement the code

} catch (e) {

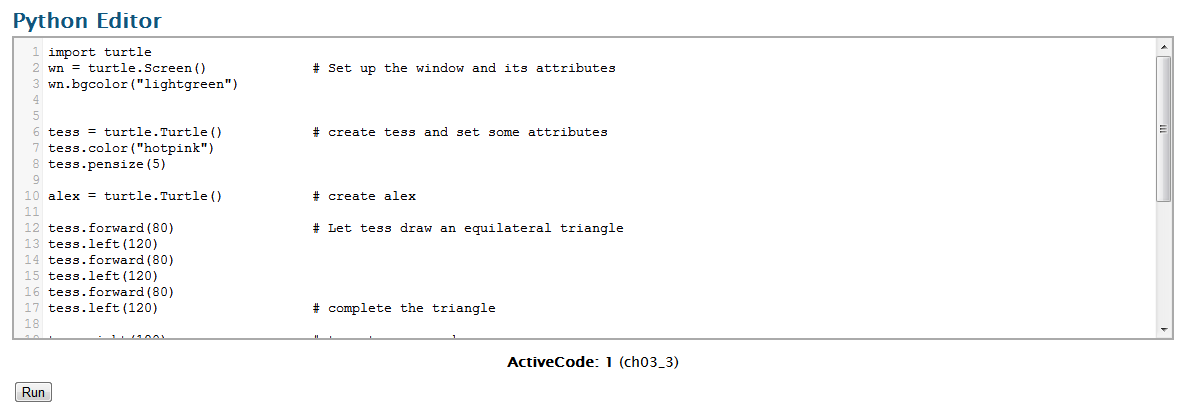
alert("ERROR: "+e);

}

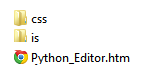
}

</script>

# 5. Python Editor(<http://algoviz-beta.cc.vt.edu/Processing/SubEdit/Python_Editor.htm>)

The Figure is the interface of a Python editor. It is composed of a textarea and a “Run” button. It just can run some simple functions of Python. 

The code is composed of the following files：



The css: [css/python\_style.css](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/css/python_style.css)

css/codemirror.css

The js: [js/bookfuncs.js](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/js/bookfuncs.js)

js/codemirror.js

[js/python.js](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/js/python.js)

[js/skulpt.js](http://algoviz-beta.cc.vt.edu/Processing/SubEdit/js/skulpt.js)

js/builtin.js

The above js libraries and CSS files are part of the How to Think Like a Computer Scientist online python book (<http://thinkcspy.appspot.com>) The most important is sculpt.js, it is used to compile Python code in the front end, which means it can make the Python code run in the browser.

Another important thing is that the tag “canvas” is used in the code like:

<canvas id="ch03\_3\_canvas" height="400" width="400" style="border-style: solid; display: none"></canvas>