***Degenerative*: a web page that degenerates with each visit**
**Eugenio Tisselli**

*"Every time someone visits this page, one of the characters that make up the HTML code gets either deleted or replaced. This causes a step-by-step degeneration, not only of the page's content, but also of its structure."*

These words could originally be read at the top of the *degenerative* web page, a piece I programmed and launched in 2005[1]. At that time, generative art had become quite popular, so coding a *de-generative language art* piece was my way of going against the grain, as I usually do. I was inspired by the work of artists who sought aesthetic pleasure in destruction, such as Gustav Metzger. In 1959, Metzger published his *Auto-Destructive Art Manifesto*, encouraging artists to use perishable or unstable materials that would degrade naturally or through human intervention. Through the progressive or immediate degradation of the artwork, his purpose was to parody what he saw as industrial societies' "obsession with destruction." Auto-destructive art was considered by Metzger as a "form of public art for industrial societies" that aesthetically demonstrated the power of those societies to "accelerate disintegrative processes" (Metzger 1959). Pete Townshend, co-founder and guitarist of the British band The Who, was one of Metzger's most famous students at the Ealing Art College in London. As you might recall, he mirrored his teacher's lessons by enthusiastically bashing his guitar at the end of the shows.

The post-industrial, digital *codeworld* compressed into my *degenerative* piece echoes and updates Metzger's views and Townshend's onstage praxis. However, beyond its raw nihilism, *degenerative* has provoked a number of interesting thoughts. Justin Berner, for instance, found that the piece confronts readers with technical obsolescence, and thus critiques the unopposed instrumentality prevalent in fields such as digital humanities, as well as in now widespread views such as *technosolutionism*. Because *degenerative* reveals its underlying code through a subtractive process triggered by its readers, Berner argues that the irreversible descent into textual formlessness foregrounds the material effects of collective action (whether conscious or not) on a medium that conceals its materiality through seemingly immaterial interfaces and proprietary black boxes (Berner 2020). Considering the precariousness and volatility of natural language that become apparent in *degenerative*, Diego Zorita Arroyo proposed that stretching fragility to a breaking point ultimately resulted in the obstruction of the formal task of language. According to Zorita Arroyo, natural language, upon losing its semantic function, reveals itself purely as a "material surface of inscription" devoid of meaning, not unlike unmodeled clay, or, better yet, like a sculpture that disfigures itself into nothingness (Arroyo 2022). Indeed, the original text of the *degenerative* web page, that is, the text that could be read before its collectively triggered defacement, also featured the following passage:

*"One day, all this text will be unreadable. This will happen slowly or quickly, depending on how this page is visited. It will not be a natural decomposition: it will be the effect of code that lies within. Like an unstoppable disease, this code will eventually render the page sick and crippled."*

In this recipe, I will show you how *degenerative* does its destructive thing. It's actually quite simple, but if you want to set your own version of the piece in motion, you'll have to make sure you have the proper technical elements and settings in place. Here's what you will need:

- A plain text editor.

---

[1] The piece is available at http://motorhueso.net/degenerative (accessed on 09/19/2025).

- Access to a remote or local web server with PHP support.

You will need the editor to write the code of *degenerative*, and the server with PHP support to execute it. Before we start, let's consider servers and the PHP programming language for a moment. As you surely know, servers are computers that fulfill users' internet requests. In the ancient World Wide Web, servers delivered mostly static hypertext pages encoded in a standard markup language called HTML. However, as the demands for personalized and context-sensitive content increased, the web became populated by dynamic web pages that contained data that was tailored according to specific requirements before being delivered to the user. PHP, or *Hypertext Pre-Processor,* is a language that carries out such kind of server-side data processing. And yes, it's true that a considerable number of programmers prefer to avoid PHP. It does feel a little patchy, like it was designed by a gang of crazy monkeys, each with its own idea of how the language should look like. But hey, PHP has been around for 30 years, and I think it's just perfect for *degenerative*!

Let's now dive into the fun part. The *degenerative* code is divided into two parts, each one in a separate file:

- The *original* code and text of the degenerative web page, written in HTML (file **A**)
- The PHP code that executes the degeneration (file **B**)

When readers open the *degenerative* web page, **B** reads **A**, destroys a fragment of **A** and saves the altered content. All of this happens on the web server; what the reader sees on his or her screen is the result of this process. Let's start by writing the contents of **A**, a text file with hypertext markup. Open a simple text editor (such as Leafpad[2] on Linux) and type this example:

```
<!DOCTYPE html>
<html>
<head>
<title>degenerative</title>
</head>
<body bgcolor="#000000" text="#999999">
<p align="center"><strong><font size="5" face="Arial, Helvetica, sans-
serif">Your text here</font></strong></p>
</body>
</html>
```

Save the file as **A.txt**. Of course, file **A** can include whatever you want, as long as its contents are composed exclusively of HTML tags and text.

Now let's turn to the PHP code[3] in file **B**:

```
<?php
$modify = "A.txt";
$fd = fopen($modify,"r"); // open file A.txt
$pgdata = fread($fd, filesize($modify)); // store file contents in $pgdata
fclose($fd); // close file
```

---

[2] Leafpad: http://tarot.freeshell.org/leafpad/ (accessed on 09/19/2025).

[3] If you want to go deeper into PHP, I recommend looking at the documentation available at https://php.net (accessed on 09/19/2025).

The four lines below the PHP opening tag read the contents of file **A** (which must be located in the same folder as file **B**) and store them in a variable named `$pgdata`.

(I am aware that my variable naming choices are a bit on the lousy side, but please bear with me)

```
$j = strlen($pgdata)-1; // contains the length of file A
$k = strpos($pgdata,"<body");
$k = strpos($pgdata,">",$k)+2; // position after the <body> tag

$charlist = array("*","_","-","","."); // characters used for degeneration
$c = $charlist[rand(0,sizeof($charlist))]; // choose one at random

$destrukt = rand($k,$j); // randomly choose a character to degenerate

$pgdata = substr($pgdata,0,$destrukt-1).$c.substr($pgdata,$destrukt,($j-
$destrukt+1)); // reconstruct the original text of file A minus with the
degenerated character
```

These lines do the following:

- The first three lines assign values to variables `$j` and `$k`. Let's recall that `$pgdata` contains the entire text of **A**, so we can tell that variable `$j` will point to its last character. Variable `$k` will point to a position in the text just after the "`>`" character of the `<body>` tag. Shortly, we will see how `$j` and `$k` are used.

- In the fourth line, a list with possible characters that can be used to *degenerate* the original text is created. Please note that this list includes the empty ("") character. And, in the fifth line, one character from the list is randomly chosen and stored in variable `$c`.

- The ominously named variable in the sixth line (`$destrukt`) is assigned a random numerical value between the values of `$j` and `$k`. What we're doing here is choosing a random position within the text string contained in `$pgdata`. As we now know, this position can be anywhere between the "`>`" character of the `<body>` tag (in order to respect the HTML code before it) and the end of the text.

- The dirty deed is done in the seventh line. Here, the character contained in `$c` is inserted at the position stored in variable `$destrukt`. This is achieved by splitting the `$pgdata` text string into two parts: before and after the desired position. The character in `$c` is used to join the parts, so we get a new value for `$pgdata` in which the text has been degenerated by substituting (or destroying) the character that was originally located at the position defined by `$destrukt`.

By the way, please don't type all these explanatory comments into the editor, or you will get some weird error messages when you try to execute the code ;-)

```
unlink($modify);
```

This line deletes the previous version of file **A**.

```
$fc = fopen($modify,"w+");
fwrite($fc,$pgdata);
fclose($fc);
```

These lines write the new version of **A**, which will now contain the degenerated text string.

```
echo($pgdata);
?>
```

And, finally, we use the `echo` command to output the resulting text for the user to read.

That's it. That's the PHP code! Once you've typed it, save it as **B.php**. As mentioned above, this new file should be on the same folder as **A.txt**. If you have access to a remote server, upload both files and get your own version of degenerative going by accessing **B.php** at the corresponding URL (for example `https://myserver.net/b.php`). If you are using a local server[4] instead, save the files to the appropriate folder on your computer and open **B.php** from your web browser.

Figure 1 shows the initial state of the piece you just typed. Two of its successive stages of degeneration are illustrated in Figures 2 and 3.

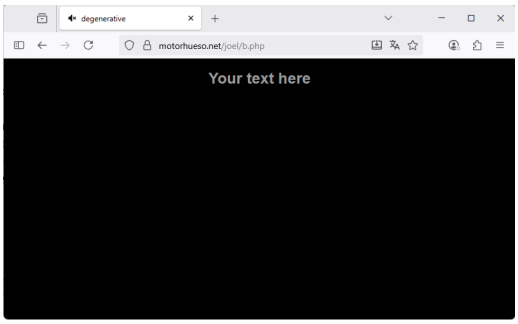


*Figure 1*

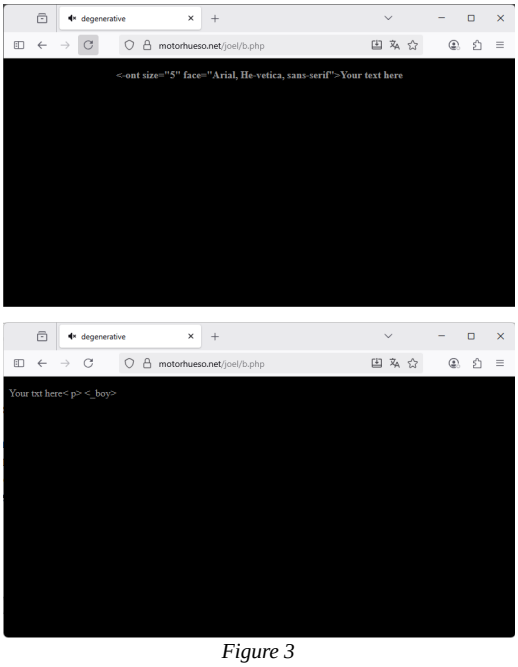


*Figure 3*

[4] XAMPP is a well-known and reliable local server specifically designed to develop PHP code. XAMPP is available at https://www.apachefriends.org/ (accessed on 09/19/2025)

Please let me add a final note. As I write this recipe in 2025, twenty years after originally coding *degenerative*, I can't shake away the feeling that the digital environment is rapidly falling apart. I look around and it seems like an "unaccommodating, destructive inertia" (that's how Justin Berner described my work!) has taken hold of the online spaces in which both trivial and important aspects and events of our lives play out. It's not a pretty sight, and I just don't know how to make things better anymore. Perhaps it's too late for that, and we'd be better off just by letting go of the digital. I really just don't know. For now, I can only say that we've taken our passion for disruption several apps too far, and that these turbulent times ask of us to maybe cool it down so we can imagine ways of rebuilding things and caring for them, rather than breaking them while we move fast. Gentle is the new punk. In that spirit, I hope *degenerative* will remind us of the fragility of our collective, online spaces.

**References.**

Arroyo, Diego Zorita (2022). *Dilemas ontológicos en la demarcación del hecho literario a raíz de Degenerativa (2005) y Regenerativa (2005) de Eugenio Tisselli*. In *Formas precarias en las literaturas hispánicas del siglo XXI* (pp. 123-135). Peter Lang.

Berner, Justin (2020). *Unhelpful Tools: Reexamining the Digital Humanities through Eugenio Tisselli's degenerative and regenerative.* Electronic Book Review. https://doi.org/10.7273/kbfc-4145

Metzger, Gustav (1959). *Auto-Destructive Art*. http://radicalart.info/destruction/metzger.html (accessed on 09/19/2025).