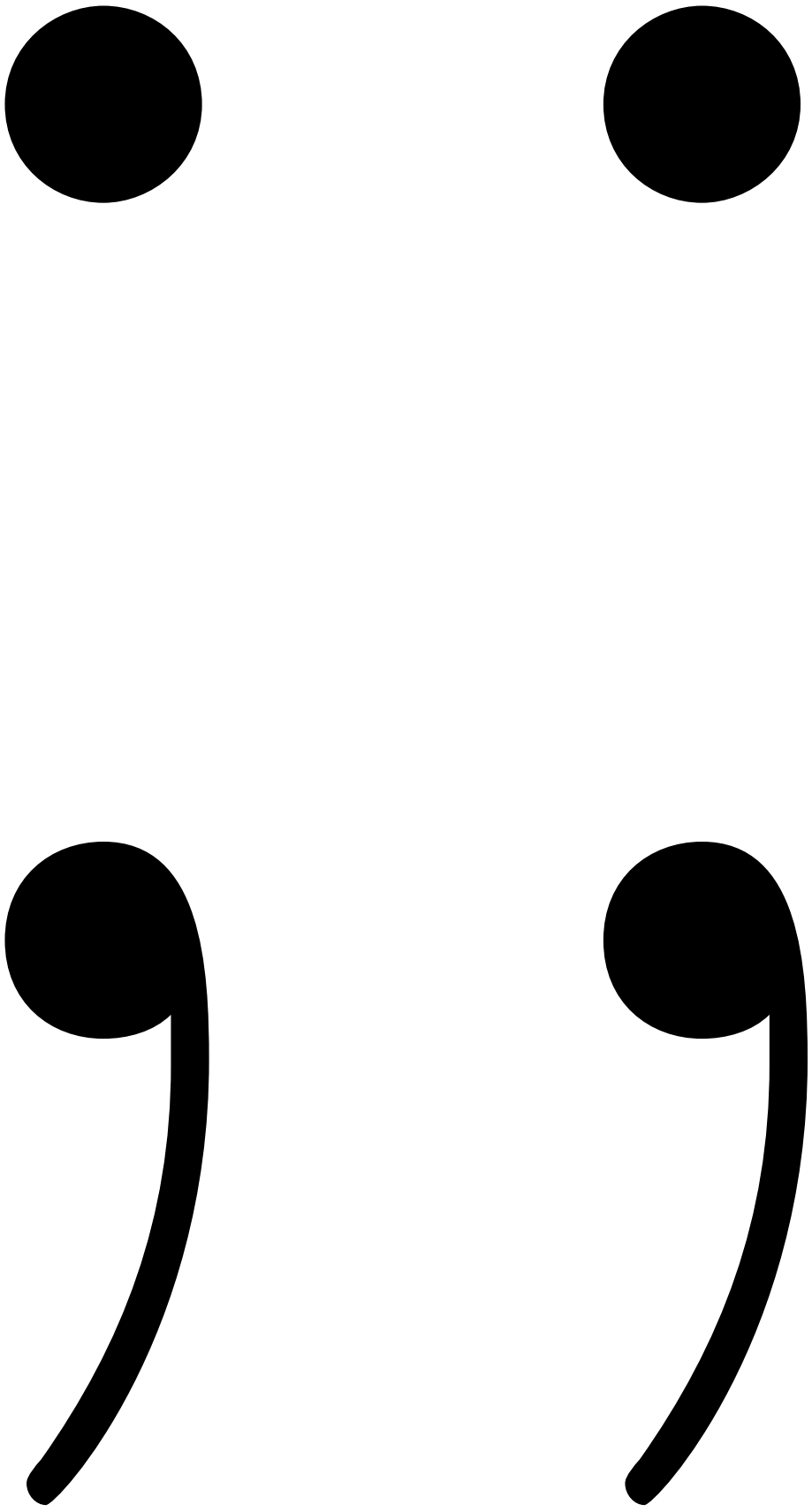


Maximizing Code Readability Using Semicolon Indentation

Tessa Parker
Illinois Institute of Technology
tparker7@hawk.iit.edu

Nelitha Kulasiri
Carnegie Mellon University
nkulasir@andrew.cmu.edu

;;;;1;;Introduction

;;;;;;;;Since the dawn of computer programming, there have been people who wish to impose their preferences on stylistic choices in code on others. This paper draws inspiration from the imposition of certain stylistic choices on others.

;;;;2;;Current Practices

;;;;;;;;Currently, many programmers choose to use one of two types of indentation in their code: tabs or spaces. These two choices are the source of intense debate within computing communities. There is no doubt that these debates have caused rifts in working relationships and potentially lead to divorces and break-ups in romantic relationships[1].
;;
;;;;;;;;Though these practices have been mostly pedantic for many languages, there have been some attempts to use this pedantry in novel languages[5]. These attempts are rather useful to learn from, as we can see how white space can be used.

;;;;3;;A New Approach

;;;;;;;;This paper suggests a new approach to delimiting logic within code: the usage of semicolons for indentation. This may be achieved by simply replacing any tabs or spaces at the start of a line with semicolons. Examples of usage are listed later on in this paper.

[1]This is shown in Season 3 Episode 6 of *Silicon Valley*. No actual cases of this phenomenon occurring have shown up after a brief internet search.

;;;;4;;Benefits

;;;;;;;;4.1;;Reduced Ambiguity

;;;;;;;;;;;While determining a new way to delimit logic in programs, it is important to ensure that the delimitation is not ambiguous. The current practice of using spaces or tabs has the disadvantage that it is not easy to determine, from a glance, which delimiter is being used, as they are both white space characters. [Make this less ambiguous]

;;;;;;;;4.2;;Increased Consistency

;;;;;;;;;;;;One common issue in large codebases, such as an open-source project, is consistency in how code is indented. Everyone has their preferences, and editors may default to one way of formatting, which likely conflicts with other contributors to the project. In fact, this has become such an issue that there are style enforcement tools dedicated to enforcing consistency in code style, including indentation. The industry has arbitrarily decided that

;;;;;;;;4.3;;Better Readability

;;;;;;;;;;;;Many programmers know the pain of having to read code printed on paper. Especially in the academic setting, code is printed on paper, potentially splitting pages. In this situation, indentation context may be lost. Students may be forced to find the ending brackets and hope they properly understand the indentation, especially in languages with significant indentation, such as Python.
;;;;;;;;;;;;The practices introduced in this article solve this issue by using a non-whitespace character to indent code. An example of Python code being used in a manner that would solve this issue is found in section 5.1 Python.

;;;;5;;Usage

;;;;;;;;The principles introduced in this proposal are relatively simple to use in code. This section will detail how this proposal can be implemented in different programming languages. Though not all languages currently support this implementation, workarounds are provided for those that don't.

;;;;;;;;5.1;;Python

;;;;;;;;;;;;Unfortunately, the principles introduced in this proposal are not immediately possible to implement in Python. However, users have come forward with suggestions of how to use the same concepts in their Python code. This method also provides the added bonus of removing any unintentional behavior of the code.

```
def write_research(journal_name):
##print(f"Writing research for {journal_name}")
##for i in range(10000):
####add_random_word();
##add_title_page("A Paper")
##add_broken_link("this link doesn\'t work, this is unintentional")
write_research("Sigbovik 2023")
```

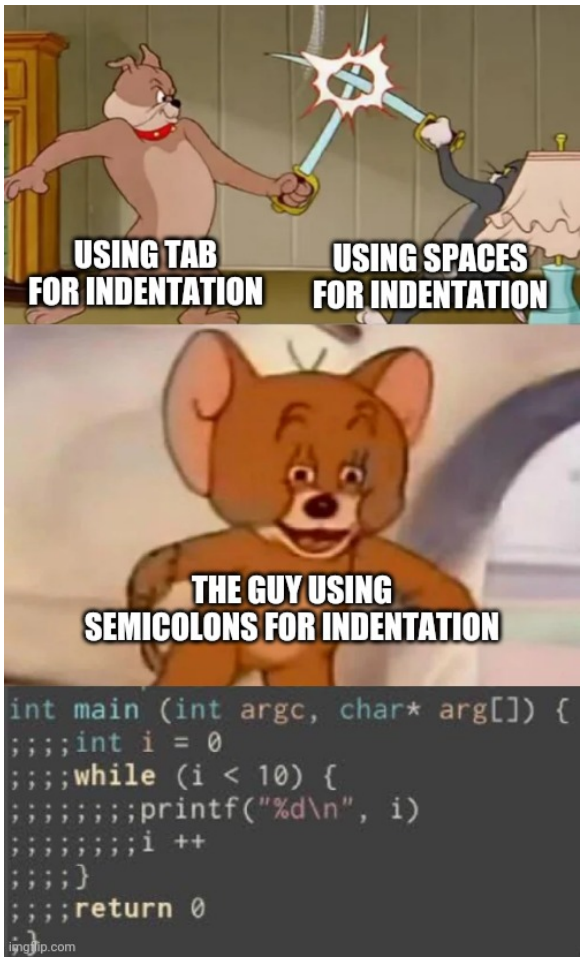
;;;;;;;;**5.2**;;**JavaScript**

;;;;;;;;;;;;With JavaScript, the practice of omitting semicolons from the ends of lines is one that may be divisive when implementing these principles. One of the primary controversies within JavaScript circles is whether to use semicolons or not, as JavaScript has optional semicolons. ;;;;;;;;;;;;This proposal does not specify whether semicolons should be used at the end of lines, so as to not be controversial in this space. With this in mind, we can see this proposal in practice with a snippet of the elevator.js library[3], with and without end-of-line semicolons.

```
 // With end-of-line semicolons
;;;;element.attachEvent("onclick", function() {
;;;;;;updateEndPosition();
;;;;;;document.documentElement.scrollTop = endPosition;
;;;;;;document.body.scrollTop = endPosition;
;;;;;;window.scroll(0, endPosition);
;;;;});
// Without end-of-line semicolons
;;;;element.attachEvent("onclick", function() {
;;;;;;updateEndPosition()
;;;;;;document.documentElement.scrollTop = endPosition
;;;;;;document.body.scrollTop = endPosition
;;;;;;window.scroll(0, endPosition)
;;;;})
```

;;;;;;;;**5.3**;;**C**

;;;;;;;;;;;;This paper was inspired by the elegant C code presented by user ablaniar on Reddit r/programminghumor[6]. Based on this code sample, we can see that this original user has decided to take the side of omitting end-of-line semicolons from their code.



;;;;;;;;;;;;The following is the sample shown above, but including end-of-line semicolons.

```
int main (int argc, char* arg[]) {
;;;;int i = 0;
;;;;while (i < 10) {
;;;;;;;;printf("%d\n", i);
;;;;;;;;i ++;
;;;;}
;;;;return 0;
}
```

;;;;;;;;**5.4**;;**C#**

;;;;;;;;;;;;This is a short example of a sorting algorithm in C#. It's amazing we found this code, thanks to code-maze.com for this one.

```
public int[] SortArray()
{
;;;;var n = NumArray.Length;
;;;;for (int i = 0; i < n - 1; i++)
;;;;;;;;for (int j = 0; j < n - i - 1; j++)
;;;;;;;;;;;;if (NumArray[j] > NumArray[j + 1])
;;;;;;;;;;;;{
;;;;;;;;;;;;;;;;var tempVar = NumArray[j];
;;;;;;;;;;;;;;;;NumArray[j] = NumArray[j + 1];
;;;;;;;;;;;;;;;;NumArray[j + 1] = tempVar;
;;;;;;;;;;;;}
;;;;return NumArray;
}
```

;;;;;;;;**5.5**;;**Java**

;;;;;;;;;;;;Java earns a special place in this paper, purely because of its widescale usage in apps. [Why is Java actually good? Add some praise here]

```
class Main {
;;public static void main(String[] args) {
;;;;System.out.println("Hello world!");
;;}
}
```

;;;;;;;;**5.6**;;**Brainfuck**

```
[squares2.b -- compute square numbers
(c) 2016 Daniel B. Cristofani
http://brainfuck.org/]

>>>>>>>>>>+>++<[
;;;;[[<<+>+>-]++++++[<++++++++>-]<-.[-]<<<]
;;;;++++++++++.[-]>>>>>[>>>>]<<<<[[<<<+>+>>-]<<<-<]>>++[
;;;;;;;;;[
;;;;;;;;;;;;;<<<++++++++++[>>>[->>+<]>[<]<<<<-]
;;;;;;;;;;;;;>>>[>>[-]>>+<<<<[>>+<<-]]>>>>
;;;;;;;;;]<<-[+>>>>]+[<<<<]>
;;;;]>>>[>>>>]<<<<-<<+<<
```

```
]

This program outputs square numbers. It doesn't terminate.
```

;;;;;;;;**5.7**;;**Rust**

;;;;;;;;;;;;As one of the most trendy languages at the time of writing, Rust has become immensely loved by its users. This code below is open source[4].

```
fn main() {
;;let cli = Cli::parse();
;;
;;start_logging_dns(cli.clone());
;;
;;if let Some(command) = cli.command {
;;;;handle_command(command).unwrap();
;;;;return;
;;}
;;
;;#[cfg(not(feature = "with-gui"))]
;;return start_headless();
;;
;;#[cfg(feature = "with-gui")]
;;if cli.headless {
;;;;start_headless();
;;} else {
;;;;gui::run_ui();
;;}
}
```

;;;;;;;;**5.8**;;**COBOL**

;;;;;;;;;;;;COBOL, while unused in most modern applications, is still a very important language to the world we live in. As a part of this, we must ensure that the code that remains is the most readable. This code is from node.cobol[1].

```
;;;;;;;* Exec Node.js code
;;;;;;;IDENTIFICATION DIVISION.
;;;;;;;PROGRAM-ID. EXEC_NODEJS.
;;;;;;;DATA DIVISION.
```

```
;;;;;;;WORKING-STORAGE SECTION.
;;;;;;;01 COMMAND_TO_RUN PIC X(200) value SPACES.

;;;;;;;LINKAGE SECTION.
;;;;;;;01 NODEJS_CODE PIC A(100) value SPACES.

;;;;;;;PROCEDURE DIVISION USING NODEJS_CODE.
;;;;;;;;;;;STRING 'node -e "' DELIMITED BY SIZE
;;;;;;;;;;;;;;;;;;;NODEJS_CODE DELIMITED BY SIZE
;;;;;;;;;;;;;;;;;;;'"'   DELIMITED BY SIZE
;;;;;;;;;;;INTO COMMAND_TO_RUN

;;;;;;;;;;;CALL 'SYSTEM' USING COMMAND_TO_RUN
;;;;;;;;;;;END-CALL
;;;;;;;EXIT PROGRAM.
```

;;;;;;;;**5.9**;;**x86 Assembly**

Turns out, x86 Assembly is perfect for semicolon indents! It's built right into the compiler. It also has the same advantage as Python has, as it removes any bugs in your code.

```
;org  0x100          ; .com files always start 256 ;bytes into the segment

;mov  dx, msg        ; the address of or message in dx
;mov  ah, 9          ; ah=9 - "print string" sub-function
;int  0x21           ; call dos services

;mov  dl, 0x0d       ; put CR into dl
;mov  ah, 2          ; ah=2 - "print character" sub-function
;int  0x21           ; call dos services

;mov  dl, 0x0a       ; put LF into dl
;mov  ah, 2          ; ah=2 - "print character" sub-function
;int  0x21           ; call dos services

;mov  ah, 0x4c       ; "terminate program" sub-function
;int  0x21           ; call dos services

;msg  db 'Hello again, World!$'    ; $-terminated message
```

;;;;6;;Potential Issues with this Approach

;;;;;;;;The approach outlined in this paper is sound in most situations. However, there are certain situations in which these principles may be unable to be implemented properly, or in where the benefits of this may be degraded.

;;;;;;;;6.1;;The Greek Question Mark

;;;;;;;;;;;;The Greek question mark (`;` `U+037E GREEK QUESTION MARK`), is visually identical to the semicolon (`;` `U+003B SEMICOLON`) commonly used in code. This introduces the issue that this C code shows:

```
// Invalid, uses the Greek question mark for indentation
while(i < 10) {
;;printf("%d\n", i);
;;i++
}
// Valid, uses the semicolon for indentation
while(i < 10) {
;;printf("%d\n", i);
;;i++
}
```

;;;;;;;;6.2;;Mid-Line Breaks

;;;;;;;;;;;;Some languages have syntax's that don't permit the usage of semicolon indentation in specific situations, such as when using data structures which use commas to separate items in the structure. Additionally, if a line is broken before the end of the statement, an error may occur when trying to indent this line with semicolons. To prevent these from causing too many issues, the authors of this paper recommend against splitting these lines.
;;;;;;;;;;;;While this paper advocates for readability, this comes at a cost for users who prefer to follow the 80-column rule[2]. An example of some JSON data following this standard of omitting linebreaks is shown below.

```
{"menu": { "id": "file", "value": "File", "popup": {"menuitem": [{"value": "New", "or
```

;;;;7;;Disclaimer

;;;;;;;;This proposal is merely a suggestion, any usage of these principles is at your own risk. Many of the concepts introduced here may cause significant harm if used in a real codebase. You've been warned.

;;References

[1] Ionica Bizau. node.cobol. `https://github.com/IonicaBizau/node.cobol`, 2021.

[2] Vinicius Brasil. The sacred 80-column rule, Nov 2022.

[3] Tim Holman. Elevator.js. `https://github.com/tholman/elevator.js`, 2020.

[4] Devyn Keeney. streamline-control. `https://github.com/devyntk/streamline-control`, 2023.

[5] Stefan Muller. A type-and-affect system for semisignificant whitespace. *Sigbovik 2022*, pages 16–21, Apr 2022.

[6] u/ablaniar. R/programmerhumor - little contribution to the indentation war, Sep 2021.