A hypothesis paper presented at PLATEAU 2015:

Layout & Indentation

How code is laid out on screen can be an extremely personal choice. Layout can suggest relatedness of code and ideas.

Or it can visually separate distinct ideas.

Indentation can suggest:
- tree structures;
- parent –
    - child relationships;
- related ideas;
- or delimit scope.

While some languages enforce this aspect of code style to varying degrees, in others developers are free to experiment. Understanding why developers choose particular techniques can offer insights into their thought processes.

H1 *Limited, consistent indentation assists developer comprehension.*
H2 *Grouping and ordering of program statements can assist developers in identifying patterns and errors.*

Typefaces & Fonts

Typefaces can send different messages but from `Courier` to **`Consolas`**, **`Lucida Console`** to `Letter Gothic`, we code in a monospace monoculture. Exactly what developers choose, if they choose at all, can be a matter for intense debate. Some brave editors are mixing this up, throwing in proportional typefaces for comments. Even when editors support differentiating keywords using multiple styles and fonts why should we limit ourselves to just one typeface?

H3 *Different typefaces aid developers in segmenting code into component parts, such as program instructions and comments.*
H4 *Varying the font, such as using boldface or italic, assists developers in tokenization of the syntax.*

Anti-Aliasing

Jaggies. We all hate them. Or do we? The use of anti-aliasing on text is a proven method to improve the reading speed and comprehension of prose. How does this effect translate to code presentation? Text with anti-aliasing applied can appear "muddy" and "fatiguing." As screen resolutions and dimensions increase does it even matter? Anecdotal commentary seems to suggest that developers combining small point sizes with anti-aliased text are most aggrieved. Maybe point-size can be correlated with developer preferences for text anti-aliasing?

H5 *Anti-aliasing improves onscreen code readability at larger point sizes.*

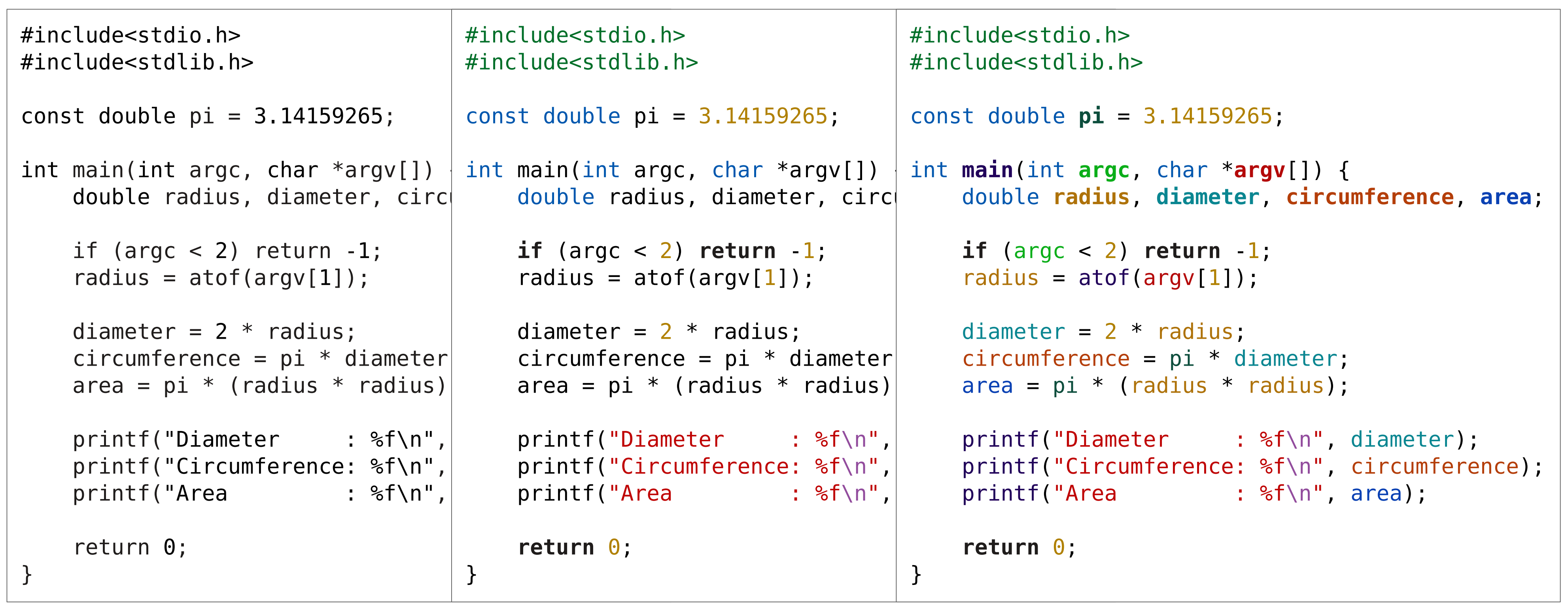
Syntax Highlighting

Syntax highlighting helps. Right? Color seems to aid in tokenizing syntax, but is highlighting all the keywords (or nouns) in blue really all that helpful?

Blue!? The travesty! Purple, surely!

If only we could all agree. Our color schemes are another one of those hard to pin down personal choices. But what if there is more to it? Are some color schemes objectively *better*? With limited research into how syntax highlighting affects developers, moving beyond simplistic schemes is a challenge.

H6 *Color schemes which help tokenize syntax by exploiting physiological properties aid readability of code.*
H7 *High contrast color schemes can induce eye fatigue over long periods of use.*



```
#include<stdio.h>
#include<stdlib.h>

const double pi = 3.14159265;

int main(int argc, char *argv[])
    double radius, diameter, circ

    if (argc < 2) return -1;
    radius = atof(argv[1]);

    diameter = 2 * radius;
    circumference = pi * diameter
    area = pi * (radius * radius)

    printf("Diameter     : %f\n",
    printf("Circumference: %f\n",
    printf("Area         : %f\n",

    return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>

const double pi = 3.14159265;

int main(int argc, char *argv[])
    double radius, diameter, circ

    if (argc < 2) return -1;
    radius = atof(argv[1]);

    diameter = 2 * radius;
    circumference = pi * diameter
    area = pi * (radius * radius)

    printf("Diameter     : %f\n",
    printf("Circumference: %f\n",
    printf("Area         : %f\n",

    return 0;
}
```

```
#include<stdio.h>
#include<stdlib.h>

const double pi = 3.14159265;

int main(int argc, char *argv[]) {
    double radius, diameter, circumference, area;

    if (argc < 2) return -1;
    radius = atof(argv[1]);

    diameter = 2 * radius;
    circumference = pi * diameter;
    area = pi * (radius * radius);

    printf("Diameter     : %f\n", diameter);
    printf("Circumference: %f\n", circumference);
    printf("Area         : %f\n", area);

    return 0;
}
```

Figure 1. KDevelop 4.6 rendering a short code sample in the C language using the *DejaVu Sans Mono* typeface. From left to right: without syntax highlighting; with basic syntax highlighting; and with semantic highlighting.

Semantic Highlighting

Almost all code reads, modifies, and records data. How data flows through a program and how it can be used is key to understanding what is actually going on. Eclipse 3.0, released in 2004, experimented with changing the presentation of code depending on its sematic meaning, such as using italic font face for static fields.

KDevelop (Figure 1.) pushes this further, systematically applying colors to variables to improve recognition. This aids the developer in identifying where the variable appears in the code, and how it impacts (or is impacted by) other variables. Helpful when the author of the code thinks ***color*** and ***color*** are clearly distinct variable names.

You did spot the capital '*i*' right? Which one? Exercise for the reader.

H8 *Highlighting differing scope aids developer identification of the impacts of changes to variables.*
H9 *Semantic highlighting aids developer understanding of data flow through code.*

UNDERSTANDING *the Effects of* CODE PRESENTATION

UNIVERSITY OF CAMBRIDGE

Intelligent Interactive Systems

EPSRC Pioneering research and skills

by **Jason T. Jacques** and **Per Ola Kristensson**
jtj21@cam.ac.uk http://jsonj.co.uk/cam/2015/plateau