Organization

- Using good programming style is important
  - Especially as program size grows
- In general (not just in code) things are easier to use and understand when they are in-order
- Good style is crucial to maintaining a code base
void draw() {
  int red = 50;
  for (int i = 20; i < 220; i += 60) {
    fill(red, 20, 20);
    red = red + 70;
    for (int j = 5; j < 500; j += 50) {
      rect(j, i + 20, 40, 40);
    }
  }
}

void draw() {
  int red = 50;
  for (int i = 20; i < 220; i += 60) {
    fill(red, 20, 20);
    red = red + 70;
    for (int j = 5; j < 500; j += 50) {
      rect(j, i + 20, 40, 40);
    }
  }
}
void draw()
{
  fill(255, 100, 25);
  rect(0, 0, 300, 100);

  fill(255, 255, 0);
  rect(0, 100, 300, 100);

  fill(0, 255, 255);
  rect(0, 200, 300, 100);

  if (mousePressed) {
    if (mouseY < 100) {
      link("http://www.amazon.com");
    } else if (mouseY < 200) {
      link("http://www.google.com");
    } else {
      link("http://www.espn.com");
    }
  }
}
// variable to control fence animation
int position = 0;

void draw() {
  background(200, 230, 255);
  strokeWeight(0);
  fill(100, 255, 100);
  rect(0, 140, 300, 100);
  strokeWeight(4);
  // repeatedly draw the fence posts
  for(int i = 0; i < 2001; i += 25) {
    line(position + i, 50, position + i, 150);
  }
  line(0, 75, 300, 75);
  line(0, 125, 300, 125);
  // increment to control fence movement speed
  position = position - 1;
}

int P1z = 0;
void draw() {
  background(200, 230, 255);
  strokeWeight(0);
  fill(100, 255, 100);
  rect(0, 140, 300, 100);
  strokeWeight(4);
  for(int i = 0; i < 2001; i += 25) {
    line(P1z + i, 50, P1z + i, 150);
  }
  line(0, 75, 300, 75);
  line(0, 125, 300, 125);
  // What is going on here?
  P1z = P1z - 1;
}
Name at least two things that you identified as bad style.
What makes good style

- Indentation
- Commenting/Documentation
- Naming
- Spacing
Indentation

- We’ve seen several types of code-constructs that use curly-braces
  - setup and draw
  - if-statements
  - for-loops
- Whenever a new “level” of curly-braces is reached, best-practice is to indent the code accordingly
- In processing, indentation is 2-spaces
  - Varies between language!
void draw () {
    background(100, 200, 250);
    strokeWeight(5);
    fill(200, 100, 50);
    rect(50, 50, 100, 100);
}

Indentation

- One pairing of open/close curly-braces indicates a new “chunk” of code

```java
void draw () {
  background(100, 200, 250);
  strokeWeight(5);
  fill(200, 100, 50);
  rect(50, 50, 100, 100);
}
```
• One pairing of open/close curly-braces indicates a new “chunk” of code
• Because of this, indent all code in-between 1 level (which is 2-spaces)

```plaintext
void draw () {
    background(100, 200, 250);
    strokeWeight(5);
    fill(200, 100, 50);
    rect(50, 50, 100, 100);
}
```
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  strokeWeight(5);
  if (mousePressed) {
    fill(255, 0, 0);
    strokeWeight(10);
  }
  rect(50, 50, 100, 100);
}
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  strokeWeight(5);
  if (mousePressed) {
    fill(255, 0, 0);
    strokeWeight(10);
  }
  rect(50, 50, 100, 100);
}
void draw () {
    background(100, 200, 250);
    fill(0, 0, 255);
    strokeWeight(5);
    if (mousePressed) {
        fill(255, 0, 0);
        strokeWeight(10);
    }
    rect(50, 50, 100, 100);
}
void draw () {
    background(100, 200, 250);
    fill(0, 0, 255);
    strokeWeight(5);
    if (mousePressed) {
        fill(255, 0, 0);
        strokeWeight(10);
    }
    rect(50, 50, 100, 100);
}
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  strokeWeight(5);
  if (mousePressed) {
    fill(255, 0, 0);
    strokeWeight(10);
  }
  rect(50, 50, 100, 100);
}
void draw () {
    background(100, 200, 250);
    fill(0, 0, 255);
    if (mousePressed) {
        for (int i = 0; i < 10 ; i += 1) {
            rect(i*20, 20, 15, 15);
        }
        fill(255, 0, 0);
    }
    rect(50, 50, 100, 100);
}
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  if (mousePressed) {
    for (int i = 0; i < 10 ; i += 1) {
      rect(i*20, 20, 15, 15);
    }
    fill(255, 0, 0);
  }
  rect(50, 50, 100, 100);
}
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  if (mousePressed) {
    for (int i = 0; i < 10 ; i += 1) {
      rect(i*20, 20, 15, 15);
    }
    fill(255, 0, 0);
  }
  rect(50, 50, 100, 100);
}
void draw () {
  background(100, 200, 250);
  fill(0, 0, 255);
  if (mousePressed) {
    for (int i = 0; i < 10 ; i += 1) {
      rect(i*20, 20, 15, 15);
    }
    fill(255, 0, 0);
  }
  rect(50, 50, 100, 100);
}
How should this be changed to have proper indentation?

Get out paper and rewrite it!

```cpp
void setup () { size(200, 200) ; }
void draw () {
  background( 100, 200, 250);
  fill(0, 0, 255);
  if (mousePressed) {
    for (int i = 0; i < 10 ; i += 1) {
      rect(i*20, 20, 15, 15);
      if (mouseX > 100) {
        fill(100, 200, 255); }
    }
  }
  if ( mouseButton == RIGHT) {
    background(0 , 0 , 0 );
    strokeWeight(7);
  }
  fill(255, 0, 0);
}
rect(50, 50, 100, 100 );
```
void setup () {
    size(200, 200);
}

void draw () {
    background(100, 200, 250);
    fill(0, 0, 255);
    if (mousePressed) {
        for (int i = 0; i < 10; i += 1) {
            rect(i*20, 20, 15, 15);
            if (mouseX > 100) {
                fill(100, 200, 255);
            }
        }
    }
}

//Continued here
if (mouseButton == RIGHT) {
    background(0, 0, 0);
    strokeWeight(7);
}
fill(255, 0, 0);
rect(50, 50, 100, 100);
Processing-Enforced naming rules

- There are some processing-enforced rules for variables names
  - Can start with any letter (A-Z, a-z), underscore (_), dollar sign ($)
  - Can continue with any of the above, and can also continue with digits (0-9)
  - Can have unlimited length
  - Cannot be any Processing keyword (void, if, else, for, etc)
  - Cannot be `false, true, null`
Which of these variables are valid?

(1) `int 123abc = 17;`
(2) `int LARGE NUMBER = 10000;`
(3) `float AnotherNumber34 = 123.456;`
(4) `char not?sure_ = ‘r’;`
(5) `int one_TWO_three = 123;`
(6) `char 3verything3 = 7;`
(7) `int true = 1`
Naming Style Guidelines

- In addition to the enforced rules, programmers have come up with best-practices for variable names
- Some programming languages have differing naming guidelines
- In processing, we use camelCase
Two types of camelCase naming

- **camelCase** - compounding words together where the first word is not capitalized, and the rest are, and no spaces
  - iPhone  eBay
- **CamelCase** - compounding words together where all words are Capitalized, and no spaces
  - FedEx  DreamWorks
- In both, numbers can be used too
- The latter is what processing uses, and what we will use for the rest of class
  - Other languages do things differently!
camelCase Naming

- What would the following convert to in camelCase?
  - “a very large number”
  - “the 3rd best item”
  - “times 10”
  - “red, green, blue”
  - “6 afraid of 7”
Variable naming

For each variable, determine if it is either (A) an error, (B) valid, but not good style, or (C) valid and good style.

(1) int 8timesTheForce = 16;
(2) int tallestPerson = 82;
(3) float inchesToFeet = 12.0;
(4) char FIRSTCharacter = ‘A’;
(5) bool SkyIsBlue = true;
(6) int 7eight9 = 789;
Commenting and Documenting

- As programs grow in size, it can become harder to keep track of what the program is doing and how it works
- Programmers use **comments** to make code easier to understand
- Comments are no excuse for bad code!
- There are actually two types of comments in processing
// Comments

- One type of comments is the double-slash
- Everything to the right of the // on the same line is considered a comment, not code
  - Thus, ignored by processing

```cpp
// everything here is a code-comment
int offset = 10; // everything over here is a comment
// here, I'm explaining what the if-statement is for
if (offset > 100) {
    fill(0);
}
```
Another type of comment is called a **block-comment**

- A block comment begins with /* and ends with */
  - Everything in-between these two symbols is a comment
  - Can span multiple lines!

```javascript
/*
  everything here is a code-comment */
int offset = 10; /* everything over here is a comment */
/* here, I’m explaining what the if-statement is for */
if (offset > 100) {
  fill(0);
}
```
/* Comments */

/*
  This is a comment
  And this is also a comment
  And so is this */

int redValue = 55; /* Use this for the red fill */

/* Can also do one-line comments! */
if (redValue > 200) {
    rect(100, 200, 300, 300);
}

/*
 * Name: J Student
 * Description: A very complex program :)
 */
Commenting and Documenting

- Comments should be used in multiple ways
  - High-level descriptions of a whole program, or function
    - Typically at top of program or function
  - Fine-grained descriptions of what a specific line of code does
    - Typically above or to the right of a line of code
void setup() {
    // Don't want to use the default of 60 - use 30 instead
    frameRate(30);
}

/**
 * This function repeats until the program exits
 */
void draw() {
    fill(100, 200, 100);
    // draws one row of squares and another row of circles below it
    for (int i = 0; i < 300; i += 50) {
        rect(i, 50, 40, 40);
        ellipse(i - 25, 100, 40, 40);
    }
}
Spacing

- There should be a space between the various “chunks” of code in your program
  - Between `setup` and `draw`
    - And other functions . . .
  - Between chunks of code that do logically different things
  - Between global variable declarations and functions
What does this code do?

```cpp
int P1z = 0;

void setup () { size(300, 200); }
void draw() { background(200, 230, 255);
    strokeWeight(0);
    fill(100, 255, 100);
    rect(0, 140, 300, 100); strokeWeight(4);
    for(int i = 0;
        i < 2001; i += 25)
    {
        line(P1z + i, 50, P1z + i, 150);
    }
    line(0, 75, 300, 75);
    line(0 , 125, 300, 125); P1z = P1z - 1;
    // What is going on here?
}
```
void draw() {
  background(200, 230, 255);
  strokeWeight(0);
  fill(100, 255, 100);
  rect(0, 140, 300, 100);
  strokeWeight(4);
  // repeatedly draw the fence posts
  for(int i = 0; i < 2001; i += 25) {
    line(position + i, 50, position + i, 150);
  }
  line(0, 75, 300, 75);
  line(0, 125, 300, 125);
  // increment to control fence movement speed
  position = position - 1;
}

How about this?

// variable to control // the fence animation
int position = 0;

/**
 * Set the size of the
 * Fence animation
 */
void setup () {
  size(300, 200);
}
Which has better style?

// variable to control fence animation
int position = 0;

void setup () {
  size(300, 200);
}

void draw () {
  background(200, 230, 255);
  strokeWeight(0);
  fill(100, 255, 100);
  rect(0, 140, 300, 100);
  strokeWeight(4);

  // repeatedly draw the fence posts
  for(int i = 0; i < 2001; i += 25) {
    line(position + i, 50, position + i, 150);
  }

  line(0, 75, 300, 75);
  line(0, 125, 300, 125);
  // increment to control fence movement speed
  position = position - 1;
}

int P1z = 0;

void setup () { size(300, 200); }  
void draw () { background(200, 230, 255);  
  strokeWeight(0);
  fill(100, 255, 100);
  rect(0, 140, 300, 100); strokeWeight(4);
  for(int i = 0;
    i < 2001; i += 25)
  {
    line(P1z + i, 50, P1z + i, 150);
  }
  line(0, 75, 300, 75);
  line(0, 125, 300, 125);
  // What is going on here?
  P1z = P1z - 1;  
}
Materials

● Required Materials
  ○ https://github.com/processing/processing/wiki/Style-Guidelines
  ○ GSWP Appending A/Coding Tips
How to write a program that behaves like this?