IEEE
Computer Chapter
Southern Alberta Section

SCHULICH
School of Engineering
UNIVERSITY OF CALGARY

IEEE Women in Engineering
Southern Alberta Section

SCHULICH IGNITE 2019

Extra Slides

- What are Coding Practices?
- Naming Conventions
- Shrinking Repetitive Code
- Organizing Code

What are coding Practices?

What are coding practices?

Coding practices are common (and often encouraged) ways of writing out code as a programmer.

Some of these are advanced concepts that go above and beyond introductory coding lessons. It’s OK if you don’t understand everything in these slides!

This is **NOT** a complete list!

Why do I need them?

Technically, you don’t, but it makes your life much easier!

1. It’s easier to use with clear practices
   - For analogy, think about why everyone drives on the right lane. It’s a lot easier if everyone agrees to it!

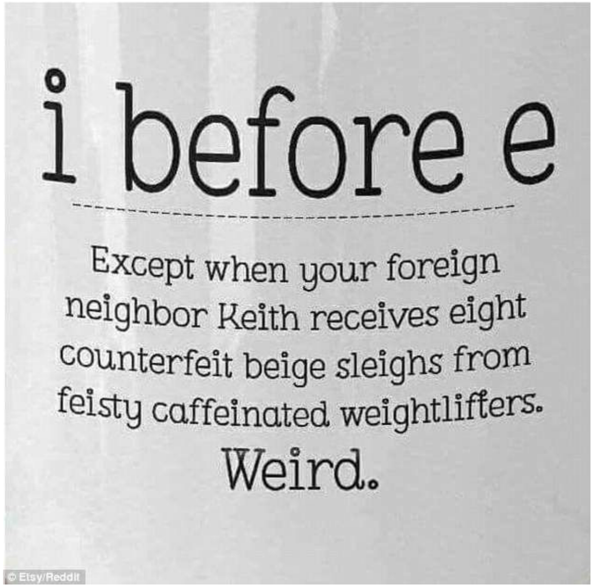
2. It’s easier to edit in the future
   - For big projects, you will edit your code and it’s better when you only need minor edits to make changes you want

Should I always use them?

No. Just like the “i before e, except after c” rule, there are lots of exceptions.

If you don’t know how to use it, or just don’t like it, then don’t use it.

There’s no clear rule when to use these and when not to. Use your best judgement (and you get that by practicing).



i before e

Except when your foreign neighbor Keith receives eight counterfeit beige sleighs from feisty caffeinated weightlifters. Weird.

© Etsy/Reddit

Naming Conventions

Naming Conventions

Every language has naming conventions.

- In English, you write Mr. and Ms. and you capitalize names like Bob.

- Programming languages have conventions too!
    - We’re going to use the Java conventions

Why? Because it is easier to recognize variables, functions and classes by just looking at the naming convention

Variable Naming Conventions

Variables are named using **camelCase**.

- First word is lowercase, and second word onwards, the first letter is uppercase

```
int firstSecondThird = 123; // Right

int FirstSecondThird = 123; // Wrong
int firstsecondthird = 123; // Wrong
```

Processing follows this rule.

- mouseX, mouseY
- keyCode

Function Naming Conventions

Functions also are named using **camelCase**.

- Parameters inside functions are also **camelCase**

```
void myFoo(int numX, int numY) // Right

void MyFoo(int numX, int numY) // Wrong
void myfoo(int numX, int numY) // Wrong
```

Classes and Objects Naming Conventions

Objects again use **camelCase** (since they're just variables!)

Classes use **PascalCase**

- Every word in the name starts with an uppercase letter

```
class BigBall // Right

class bigBall // Wrong

BigBall myBall = new BigBall(); // Right

BigBall myball = new BigBall(); // Wrong
```

Shrinking Repetitive Code

DRY: Don’t Repeat Yourself

We have a principle called DRY: don’t repeat yourself

Whenever you have highly repetitive code, you can (almost) always cut down on the repetition.

Tools to cut down repetition:

- Arrays
- Loops
- Functions
- Classes and objects

Why Use Functions?

- Functions are great for organizing code.
-
- They're much more common in medium and large programs
  - 99% chance you should use functions in your game for organization

- They cut down on lots of repetition

Functions: Condensing Repeating Code

```
if (ball1.y < 500) {
  ball1.speedY += 1;
} else {
  ball1.y = 500;
  ball1.speedY = 0;
}
```

```
if (ball2.y < 500) {
  ball2.speedY += 1;
} else {
  ball2.y = 500;
  ball2.speedY = 0;
}
```

```
if (ball3.y < 500) {
  ball3.speedY += 1;
} else {
  ball3.y = 500;
  ball3.speedY = 0;
}
```

```
void draw() {
  moveBall(ball1);
  moveBall(ball2);
  moveBall(ball3);
}

void moveBall(Ball ball) {
 if (ball.y < 500) {
    ball.speedY += 1;
  } else {
    ball.y = 500;
    ball.speedY = 0;
  }
}
```

Organizing Code

Organizing Your Code

- Programs can get big, fast!
  - Writing just a basic game can probably have hundreds of lines of code

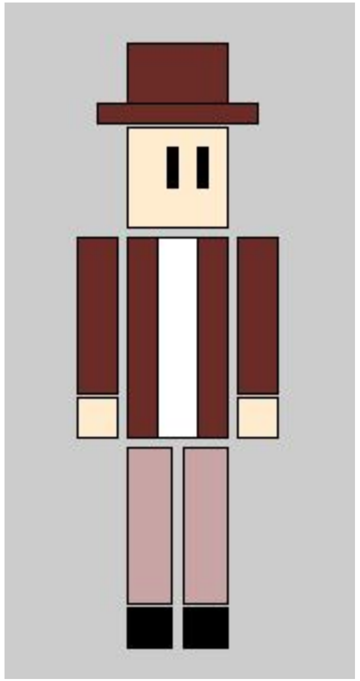
- Programs need to be well organized, or else it's really difficult to work with
  - Finding some code is tough
  - Editing that code is almost impossible!

I'm finally going to put this clutter in the recycling bin!
Once I find the recycling bin...

Organizing Your Code

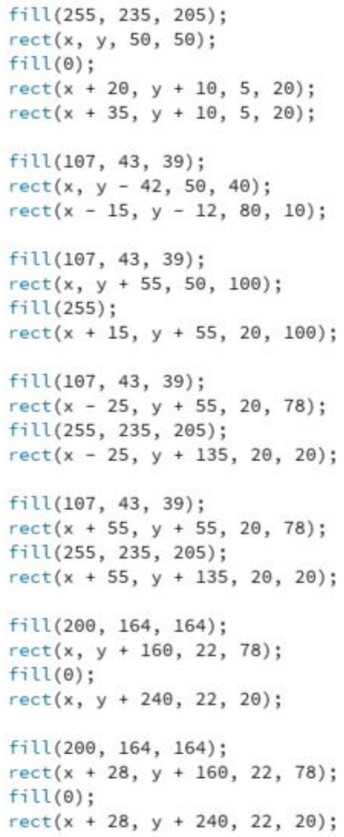
- All the code on the right is just to draw the player inside draw()
- Once we add the ability to make the player walk or jump, then draw() will get huge!
- It's also **really hard to read** what all that code does
- Let's clean this up with functions

```
fill(255, 235, 205);
rect(x, y, 50, 50);
fill(0);
rect(x + 20, y + 10, 5, 20);
rect(x + 35, y + 10, 5, 20);

fill(107, 43, 39);
rect(x, y - 42, 50, 40);
rect(x - 15, y - 12, 80, 10);

fill(107, 43, 39);
rect(x, y + 55, 50, 100);
fill(255);
rect(x + 15, y + 55, 20, 100);

fill(107, 43, 39);
rect(x - 25, y + 55, 20, 78);
fill(255, 235, 205);
rect(x - 25, y + 135, 20, 20);

fill(107, 43, 39);
rect(x + 55, y + 55, 20, 78);
fill(255, 235, 205);
rect(x + 55, y + 135, 20, 20);

fill(200, 164, 164);
rect(x, y + 160, 22, 78);
fill(0);
rect(x, y + 240, 22, 20);

fill(200, 164, 164);
rect(x + 28, y + 160, 22, 78);
fill(0);
rect(x + 28, y + 240, 22, 20);
```

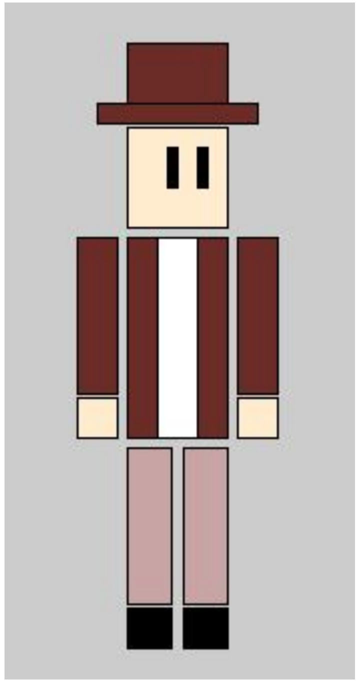


Organizing Your Code

- Just like with repetitive code, look for the “chunks” of code that belong together. Then move it into a function
  - The first 5 lines all are related to drawing the head, let’s make a function called **drawHead()**
  - Let’s also make **drawHat()** and **drawBody()** and **drawArms()** etc.



```
fill(255, 235, 205);
rect(x, y, 50, 50);
fill(0);
rect(x + 20, y + 10, 5, 20);
rect(x + 35, y + 10, 5, 20);

fill(107, 43, 39);
rect(x, y - 42, 50, 40);
rect(x - 15, y - 12, 80, 10);

fill(107, 43, 39);
rect(x, y + 55, 50, 100);
fill(255);
rect(x + 15, y + 55, 20, 100);

fill(107, 43, 39);
rect(x - 25, y + 55, 20, 78);
fill(255, 235, 205);
rect(x - 25, y + 135, 20, 20);

fill(107, 43, 39);
rect(x + 55, y + 55, 20, 78);
fill(255, 235, 205);
rect(x + 55, y + 135, 20, 20);

fill(200, 164, 164);
rect(x, y + 160, 22, 78);
fill(0);
rect(x, y + 240, 22, 20);

fill(200, 164, 164);
rect(x + 28, y + 160, 22, 78);
fill(0);
rect(x + 28, y + 240, 22, 20);
```



Organizing Your Code

```
void draw() {
  drawHat();
  drawHead();
  drawBody();
  drawArms();
  drawLegs();
}
```

```
void drawHat() {
  fill(107, 43, 39);
  rect(x, y - 42, 50, 40);
  rect(x - 15, y - 12, 80, 10);
}

void drawHead() {
  fill(255, 235, 205);
  rect(x, y, 50, 50);
  fill(0);
  rect(x + 20, y + 10, 5, 20);
  rect(x + 35, y + 10, 5, 20);
}

void drawBody() {
  fill(107, 43, 39);
  rect(x, y + 55, 50, 100);
  fill(255);
  rect(x + 15, y + 55, 20, 100);
}
```

```
void drawArms() {
  fill(107, 43, 39);
  rect(x - 25, y + 55, 20, 78);
  fill(255, 235, 205);
  rect(x - 25, y + 135, 20, 20);

  fill(107, 43, 39);
  rect(x + 55, y + 55, 20, 78);
  fill(255, 235, 205);
  rect(x + 55, y + 135, 20, 20);
}

void drawLegs() {
  fill(200, 164, 164);
  rect(x, y + 160, 22, 78);
  fill(0);
  rect(x, y + 240, 22, 20);

  fill(200, 164, 164);
  rect(x + 28, y + 160, 22, 78);
  fill(0);
  rect(x + 28, y + 240, 22, 20);
}
```

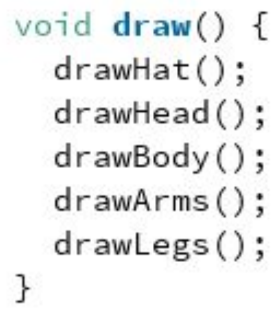
Organizing Your Code

- How is this any better? This is even more lines of code than before!

- Because our main function, draw() is now much simpler

- If you need to know what draw() is doing, you can read it and instantly understand

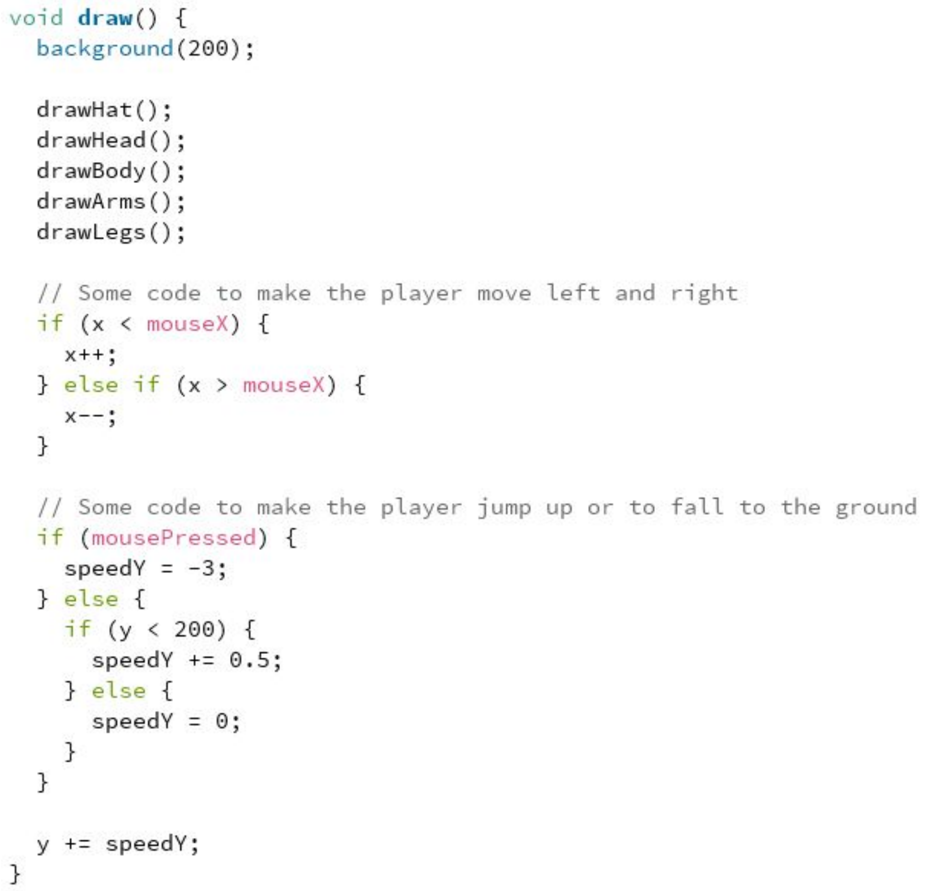
- If you need to know what drawHat() is doing, you can focus on that, and not have to read about the body or arms etc.



```
void draw() {
  drawHat();
  drawHead();
  drawBody();
  drawArms();
  drawLegs();
}
```

Let's Add More Code!

- It's VERY common to need to keep building up bigger and bigger code as you go.
- In this example, the player will follow the mouse and have gravity
- The draw() function is starting to look complicated again, let's use more functions



```
void draw() {
  background(200);

  drawHat();
  drawHead();
  drawBody();
  drawArms();
  drawLegs();

  // Some code to make the player move left and right
  if (x < mouseX) {
    x++;
  } else if (x > mouseX) {
    x--;
  }

  // Some code to make the player jump up or to fall to the ground
  if (mousePressed) {
    speedY = -3;
  } else {
    if (y < 200) {
      speedY += 0.5;
    } else {
      speedY = 0;
    }
  }

  y += speedY;
}
```

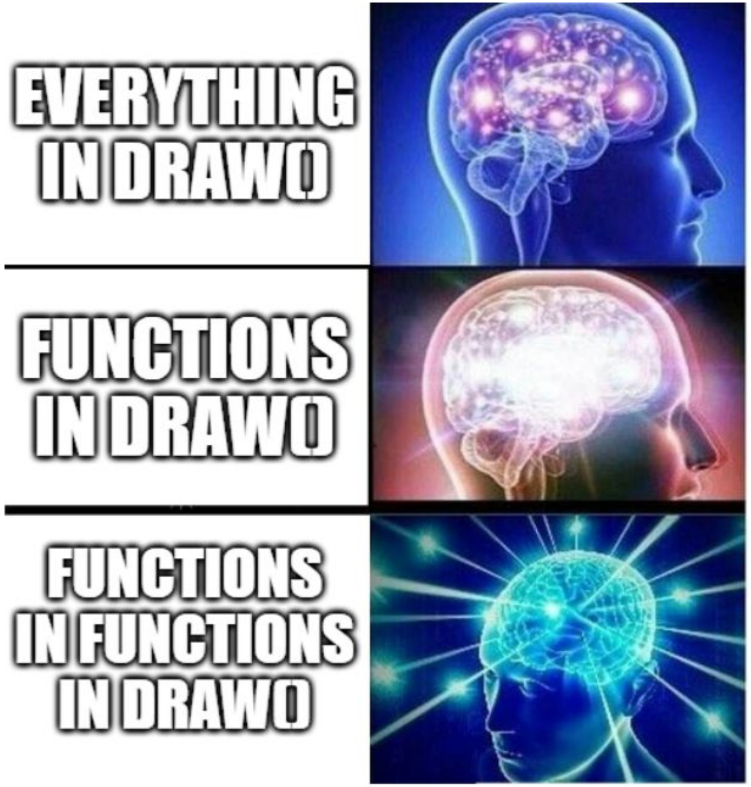
Organizing Our New Code

```
void draw() {
  background(200);

  drawHat();
  drawHead();
  drawBody();
  drawArms();
  drawLegs();

  // Some code to make the player move left and right
  followMouse();

  // Some code to make the player jump up or to fall to the ground
  if (mousePressed) {
    jump();
  } else {
    fall();
  }

  y += speedY;
}
```

```
void followMouse() {
  if (x < mouseX) {
    x++;
  } else if (x > mouseX) {
    x--;
  }
}


void jump() {
   speedY = -3;
}


void fall() {
  if (y < 200) {
     speedY += 0.5;
    } else {
     speedY = 0;
    }
}
```

Organizing Our New Code cont'd

- Our draw() is more readable now, but it can be better still.
- Even with functions, draw() is still getting pretty big
- We can fix this with even more functions!



Organizing Our New Code cont'd

```
void draw() {
  background(200);

  drawPlayer();

  followMouse();
  moveVertical();
}
```

```
void drawPlayer() {
  drawHat();
  drawHead();
  drawBody();
  drawArms();
  drawLegs();
}
```

```
void moveVertical() {
  // Some code to make the player jump up or to fall to the ground
  if (mousePressed) {
    jump();
  } else {
    fall();
  }

  y += speedY;
}
```