

COMP 1010- Summer 2015 (A01)

Jim (James) Young

young@cs.umanitoba.ca

jimyoung.ca

Back to our badguy example

```
final int MAX_MOVE = 20;
final int BG_COLOR = 0;
int badGuy1Size = 20;
int badGuy1Color = 255;
int badGuy1X = 0;
int badGuy1Y = 0;
```

```
int badGuy2Size = 40;
int badGuy2Color = 100;
int badGuy2X = 0;
int badGuy2Y = 0;
```

```
int badGuy3Size = 5;
int badGuy3Color = 180;
int badGuy3X = 0;
int badGuy3Y = 0;
```

```
void setup()
{
  size(500,500);
}
```

```
void drawBadGuy(int x, int y, int size, int col)
{
  fill(col);
  stroke(col);
  rect(x, y, size, size);
}
```

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

```
  // bad guy 1
  int move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy1X += move;
  badGuy1X = min(badGuy1X, width-1);
  badGuy1X = max(badGuy1X, 0);
```

```
  move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy1Y += move;
  badGuy1Y = min(badGuy1Y, height-1);
  badGuy1Y = max(badGuy1Y, 0);
```

```
  drawBadGuy(badGuy1X, badGuy1Y, badGuy1Size, badGuy1Color);
```

```
  // bad guy 2
  move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy2X += move;
  badGuy2X = min(badGuy2X, width-1);
  badGuy2X = max(badGuy2X, 0);
```

```
  move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy2Y += move;
  badGuy2Y = min(badGuy2Y, height-1);
  badGuy2Y = max(badGuy2Y, 0);
```

```
  drawBadGuy(badGuy2X, badGuy2Y, badGuy2Size, badGuy2Color);
```

```
  // bad guy 3
  move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy3X += move;
  badGuy3X = min(badGuy3X, width-1);
  badGuy3X = max(badGuy3X, 0);
```

```
  move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
  badGuy3Y += move;
  badGuy3Y = min(badGuy3Y, height-1);
  badGuy3Y = max(badGuy3Y, 0);
```

```
  drawBadGuy(badGuy3X, badGuy3Y, badGuy3Size, badGuy3Color);
}
```

How can we simplify this?

```
// bad guy 1
int move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
badGuy1X += move;
badGuy1X = min(badGuy1X, width-1);
badGuy1X = max(badGuy1X, 0);

move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
badGuy1Y += move;
badGuy1Y = min(badGuy1Y, height-1);
badGuy1Y = max(badGuy1Y, 0);
```

See similarities? What if we call a function once for X and once for Y... something like

```
badGuy1X = doMove(badGuy1X, 0, width-1);
badGuy1Y = doMove(badGuy1Y, 0, height-1);
```

Make the function

```
int move = (int)(random(MAX_MOVE*2)-MAX_MOVE);
badGuy1X += move;
badGuy1X = min(badGuy1X, width-1);
badGuy1X = max(badGuy1X, 0);

...
badGuy1X = doMove(badGuy1X, 0, width-1);
badGuy1Y = doMove(badGuy1Y, 0, height-1);
```

Header:

```
int doMove(int position, int minimum, int maximum)
```

Body:

copy the above but use the local variables

Rewrite the main code

Careful!

using non-final globals in a function!!

- hard to keep track of
- doesn't scale well to larger programs

A function should accept changing, specific data as parameters

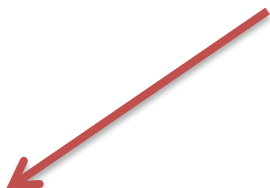
A function should return data through the return mechanism.

Be very very careful if your function is going to touch any global variables, as other code will need to expect those to change based on your function.

get data back from a function call by “return”ing it!

```
double addTax(double price) {  
    price = price * TAX_RATE;  
    return price;  
}
```

Specific data
comes in!



Preferred way for data to come back!



function parameters and variables...

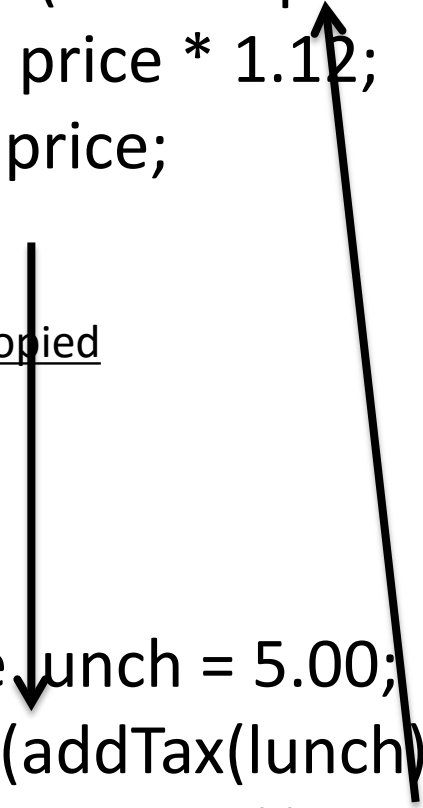
data is always copied

```
double addTax(double price) {  
    price = price * 1.12;  
    return price;  
}
```

the information is copied
here.

```
void draw()  
{  
    double lunch = 5.00;  
    println(addTax(lunch));  
    println(lunch); // what is output?  
}
```

the information is copied
here. inside the method you
only have a copy. changes here
do not reflect back!



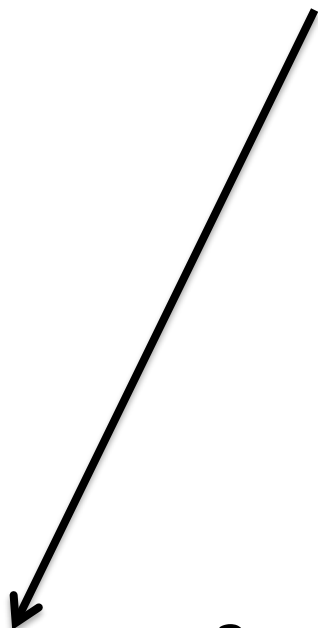
function parameters and variables...

data is always copied

```
double addTax(double price) {  
    price = price * 1.12;  
    return price;  
}
```

```
void draw()  
{  
    double lunch = 5.00;  
    println(addTax(lunch));  
    println(lunch); // what is output?  
}
```

the output is 5.0. Although the function modifies the price variable, since only a copy was passed in, the original was unchanged!



keep tunnel vision.....

forget the rest of your program and solve only the simpler problem in front of you.....

```
double addTax(double price) {  
    //....  
    return?  
}
```

think: I have “price”, and need to return a double
The rest of the program is irrelevant!

functions can call each other!

you can call user-defined methods from anywhere – even inside other methods!

make a program that tests if a string is a palindrome!

palindrome: a word that is spelt the same forwards as backwards

algorithm: reverse a word and compare it to the original

program structure

