What’s Up with Processing?

Testing and Repetition

Lawrence Snyder
University of Washington, Seattle

Announcements

- How’s it going?

- Techniques to write CORRECT programs
  - Start with a program that “works”; then check often
  - Always think of the computation from the computer’s point of view
  - If your are confused about the values of variables, get a sheet of paper and write out the changes to the values as the program proceeds

Review from Last Time

- Mod … x % ? produces 0, 1, 2, 3, 4, 5, 6, 7

Plan For Today

- To present the basic features of Processing so they are all in one place so you can check it if you forget how something goes
- Check Chapter 4 in the Reas & Fry book …
- You can also get really good help from the Processing Reference Page:

  http://www.processing.org/reference/

  Everything I mention today is there

Plan On Returning To This Lecture When You Forget

List of Items

- Variables and Declarations*
- Assignments*
- Expressions
- Repetition (looping) or For-statements
- Tests or If-statements)
- Writing programs

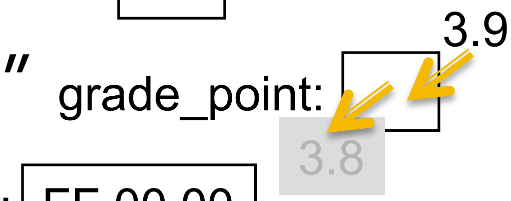
* You have already learned about these

Variables …

- Facts about variables are …
  - Variables "contain" their values, and they can be changed using assignment
  - Variables have a data type such as `int`, `float`, `color`, etc. which is the kind of data they contain
- Rules about variables are …
  - Variables can be any string of letters, numbers or underscores (_) starting with a letter; case-sensitive
  - Variables must be declared; declarations at the top of the program or at the start of a function
  - Variables can be initialized in a declaration

Variables, the Picture

- Facts
  - "Contain their value": grade_point: 3.8
  - "Assign to change: grade_point = 3.9;"



  - "Variables have data type": red: FF 00 00
- Rules
  - "Any string": Pick, MEANINGFUL, varz, theyRuseful_4_U_despite_their_length
  - "Declare vars": int score; float gpa; color purple;
  - "Initializing is OK": int score=0; float gpa=4.0; color purple=color(128, 0, 128);

Assignments

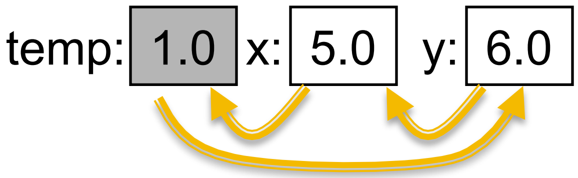
- Facts about assignment:
  - Its form is always <*variable*> = <*expression*>
  - Information moves from right to left
  - The <*expression*> is computed first, then the variable is changed, so x=x+1 is sensible
  - To exchange values in two variables takes 3 stmts
- Rules about assignment:
  - All assignment statements end with a semicolon

Assignments, The Picture

- Facts
  - "Form": grade_point=3.9; yellow=color(255,255,0);

    3.9 = grade_point is ILLEGAL
  - "Info moves right to left": x = 4.0;
  - "Compute <*expression*> first: x = x + 1; 5.0 ←→ 4.0 + 1
  - "Exchanging values of x, y takes 3 statements":

```
temp = x;
x = y;
y = temp;
```

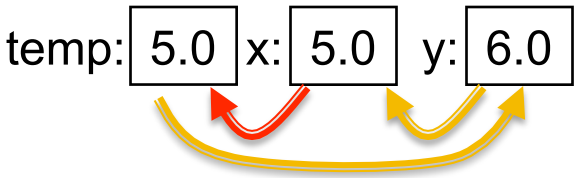

temp: 1.0 x: 5.0 y: 6.0

Assignments, The Picture

- Facts
  - "Form": grade_point=3.9; yellow=color(255,255,0);

    3.9 = grade_point is ILLEGAL
  - "Info moves right to left": x = 4.0;
  - "Compute <*expression*> first: x = x + 1; 5.0 ←→ 4.0 + 1
  - "Exchanging values of x, y takes 3 statements":

```
temp = x;
x = y;
y = temp;
```



Assignments, The Picture

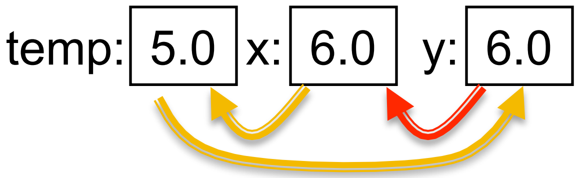
- Facts
  - "Form": grade_point=3.9; yellow=color(255,255,0);

    3.9 = grade_point is ILLEGAL
  - "Info moves right to left": x = 4.0;
  - "Compute <*expression*> first: x = x + 1; 5.0 ←→ 4.0 + 1
  - "Exchanging values of x, y takes 3 statements":

    temp = x;

    x = y;

    y = temp;



temp: 5.0 x: 6.0 y: 6.0

Assignments, The Picture

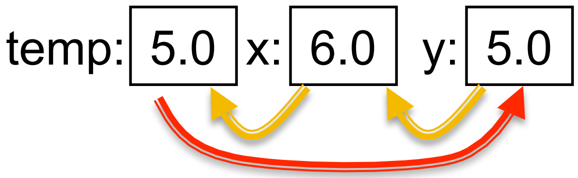
- Facts
  - "Form": grade_point=3.9; yellow=color(255,255,0);

    3.9 = grade_point is ILLEGAL
  - "Info moves right to left": x ← = 4.0;
  - "Compute <*expression*> first": x = x + 1;   5.0 ←→ 4.0 + 1
  - "Exchanging values of x, y takes 3 statements":

    temp = x;

    x = y;

    y = temp;



Expressions

- Facts about expressions
  - Expressions are formulas using: + - * / % || ! && == < <= >= > !=
  - Operators can only be used with certain data types and their result is a certain data type
  - Putting in parentheses is OK, and it's smart
- Rules about expressions
  - Expressions can usually go where variables can go

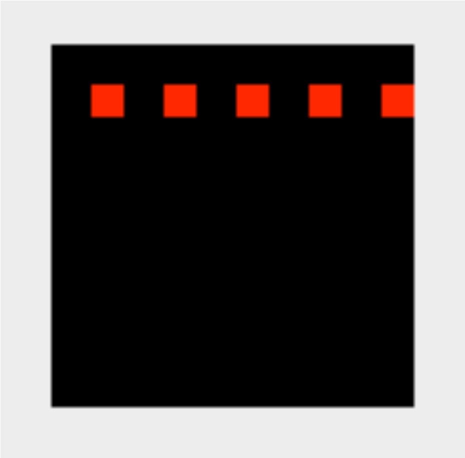
Expressions, the Picture

- Facts
  - Expressions are formulas: a+b points*wgt
    (year%4 == 0) 7 != 4 (age>12) && (age<20)
  - "Need & give data types" + - * / % < <= => > want numbers; && ! || want logical (Boolean) values == and != want arguments to be the same type
  - "Parentheses are good": (a * b) + c is the same as a*b+c, but easier to read
- Rules
  - "Expressions replace vars": rect(x, y, x+4, y+4);

Repetition (or looping)

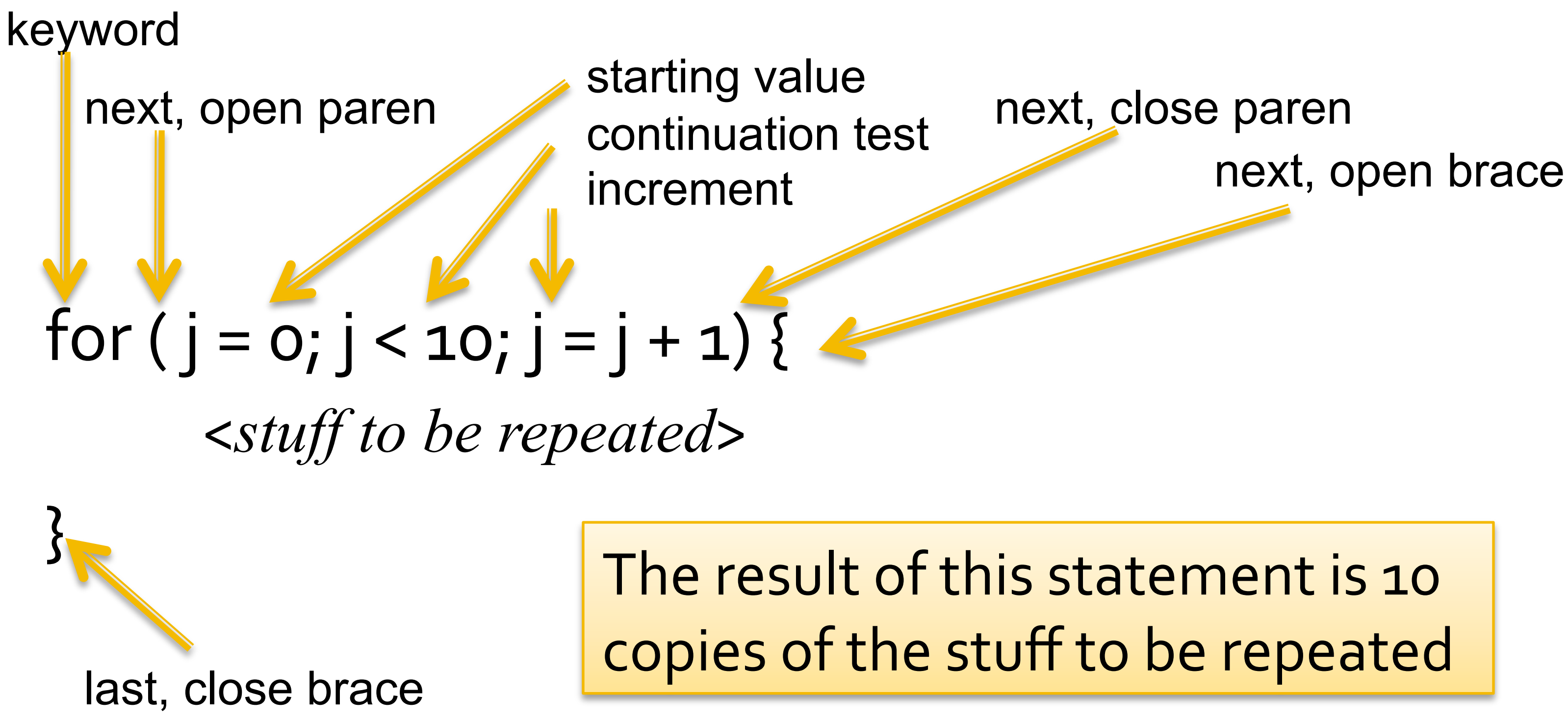
- Repeating commands is a powerful way to use a computer … we could repeat them, but all programming systems have a way to loop:
  - Lightbot 2.0 used recursion, a function calling itself
  - Symbolic Lightbot prefixed a number, 2:Step
- Processing (and other modern languages) use a **for** loop:

```
for (i = 0; i < 5; i = i + 1) {
   rect(10+20*i,10,10, 10);
}
```



Repetiton, the Picture

- A for loop has several parts, all required …



keyword

next, open paren

starting value
continuation test
increment

next, close paren

next, open brace

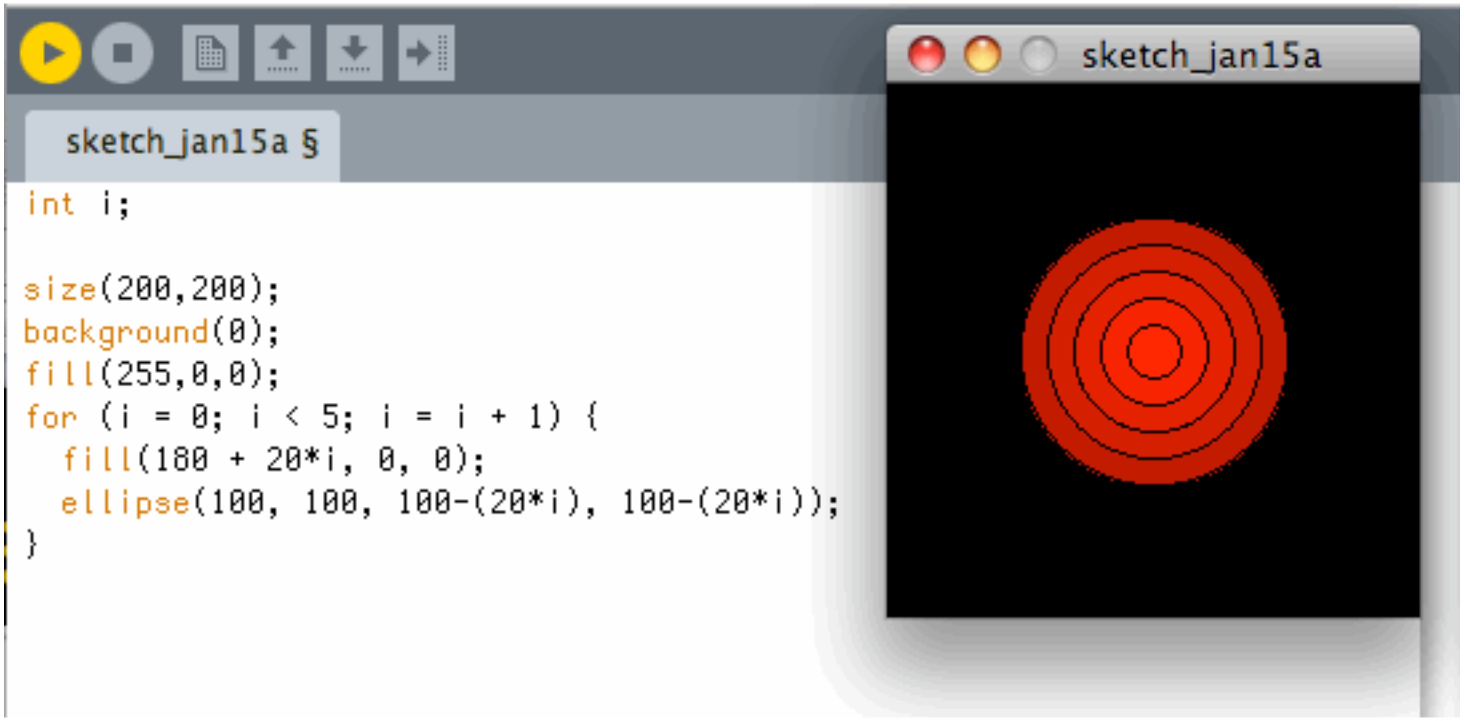
```
for ( j = 0; j < 10; j = j + 1) {
        <stuff to be repeated>
}
```

last, close brace

The result of this statement is 10 copies of the stuff to be repeated

Repetition, Another Picture

- As a further example, consider a bullseye



```
int i;

size(200,200);
background(0);
fill(255,0,0);
for (i = 0; i < 5; i = i + 1) {
  fill(180 + 20*i, 0, 0);
  ellipse(100, 100, 100-(20*i), 100-(20*i));
}
```

i: 0
1
2
3
4

- Note the *loop variable* must be declared … could write: for (int i = 0; …

Tests, A/K/A If statements

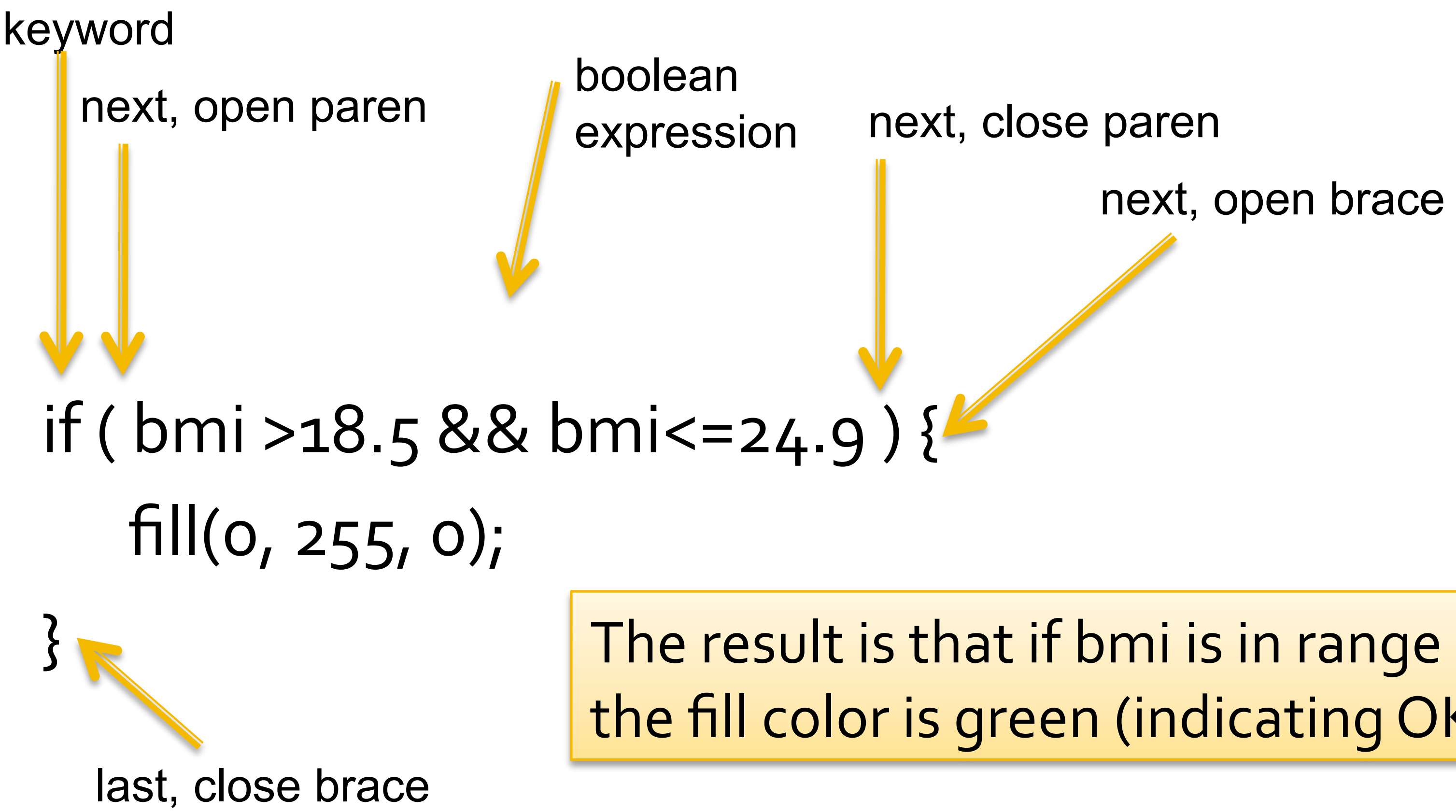
- The instructions of a program are executed sequentially, one after another … sometimes we want to skip some: Say “Hello” to the **If**
- **If** also has a required form

```
if (year%4 == 0) {
        <stuff to do if condition true>;
}

if (chosen_tint != red) {
        fill(chosen_tint);
}
```

Tests, the Picture

- An **If**-statement has a standard form



keyword

next, open paren

boolean expression

next, close paren

next, open brace

```
if ( bmi >18.5 && bmi<=24.9 ) {
    fill(0, 255, 0);
}
```

last, close brace

The result is that if bmi is in range the fill color is green (indicating OK)

Else Statement

- What happens if we want to do something else if the condition is false? What else? **else**!
- The **else** statement must follow an **if** …

```
if (year%4 == 0) {
        <stuff to do if condition true>; //Then Clause
} else {
        <stuff to do if condition false>; //Else Clause
}
```

Else, the Picture

- The standard form my now be obvious

```
if (year%4 == 0) {
        feb_days = 29;
} else {
        feb_days = 28;
}
```

Else must follow if because it does the test

open brace, immediately after “else”

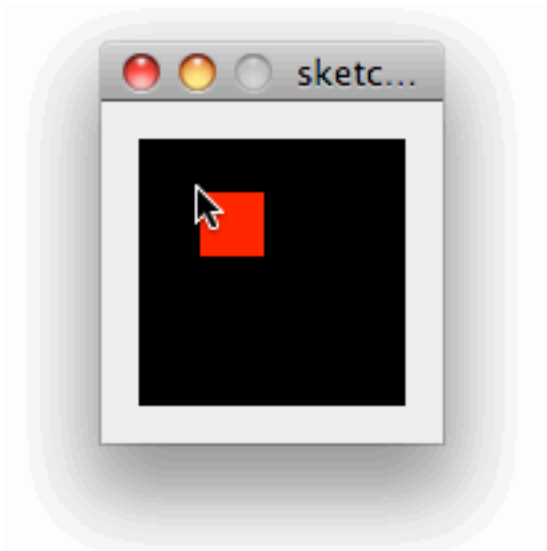
keyword

finally, close brace

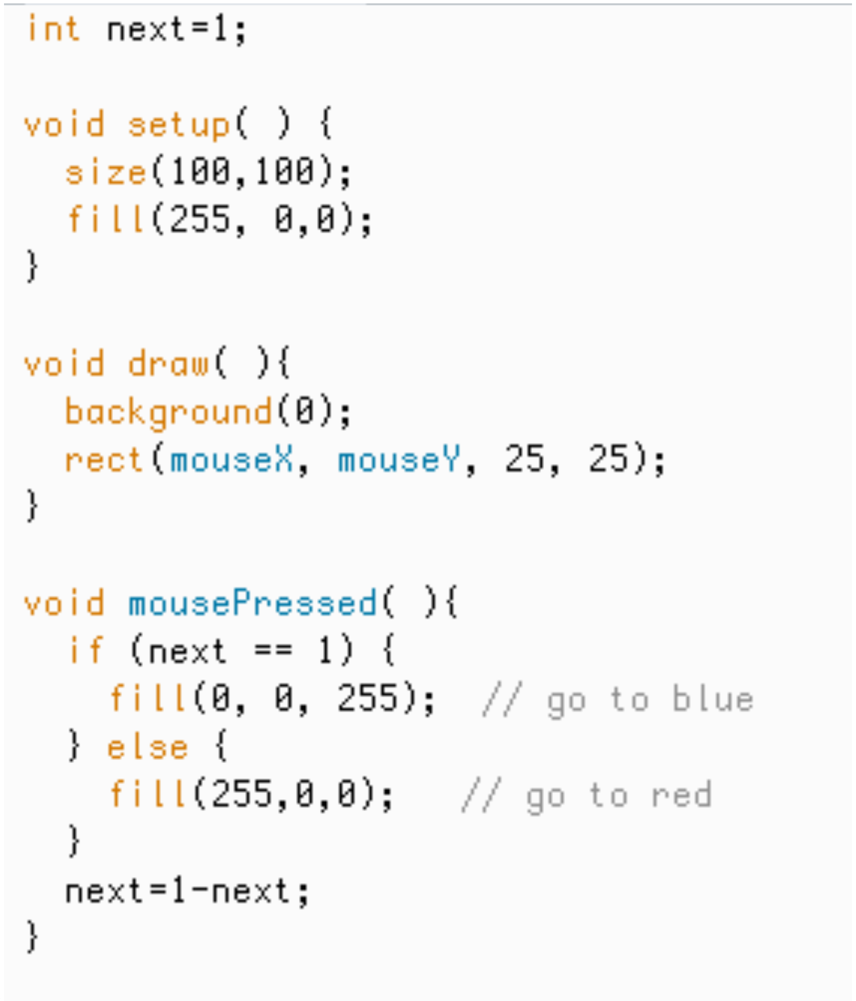
The result is sets the number of days in February based on leap year

If/Else, The Demo

- Let's go to processing and recall an early class example



Just Do It



```
int next=1;

void setup( ) {
  size(100,100);
  fill(255, 0,0);
}

void draw( ){
  background(0);
  rect(mouseX, mouseY, 25, 25);
}

void mousePressed( ){
  if (next == 1) {
    fill(0, 0, 255);   // go to blue
  } else {
    fill(255,0,0);    // go to red
  }
  next=1-next;
}
```

Writing Programs

- Naturally, programs are given sequentially, the declarations at the top
- Braces { } are statement groupers … they make a sequence of statements into one thing, like the "true clause of an If-statement"
- All statements must end with a semicolon EXCEPT the grouping braces … they don't end with a semicolon (OK, it's a rare inconsistency about computer languages!)
- Generally white space doesn't matter; be neat!