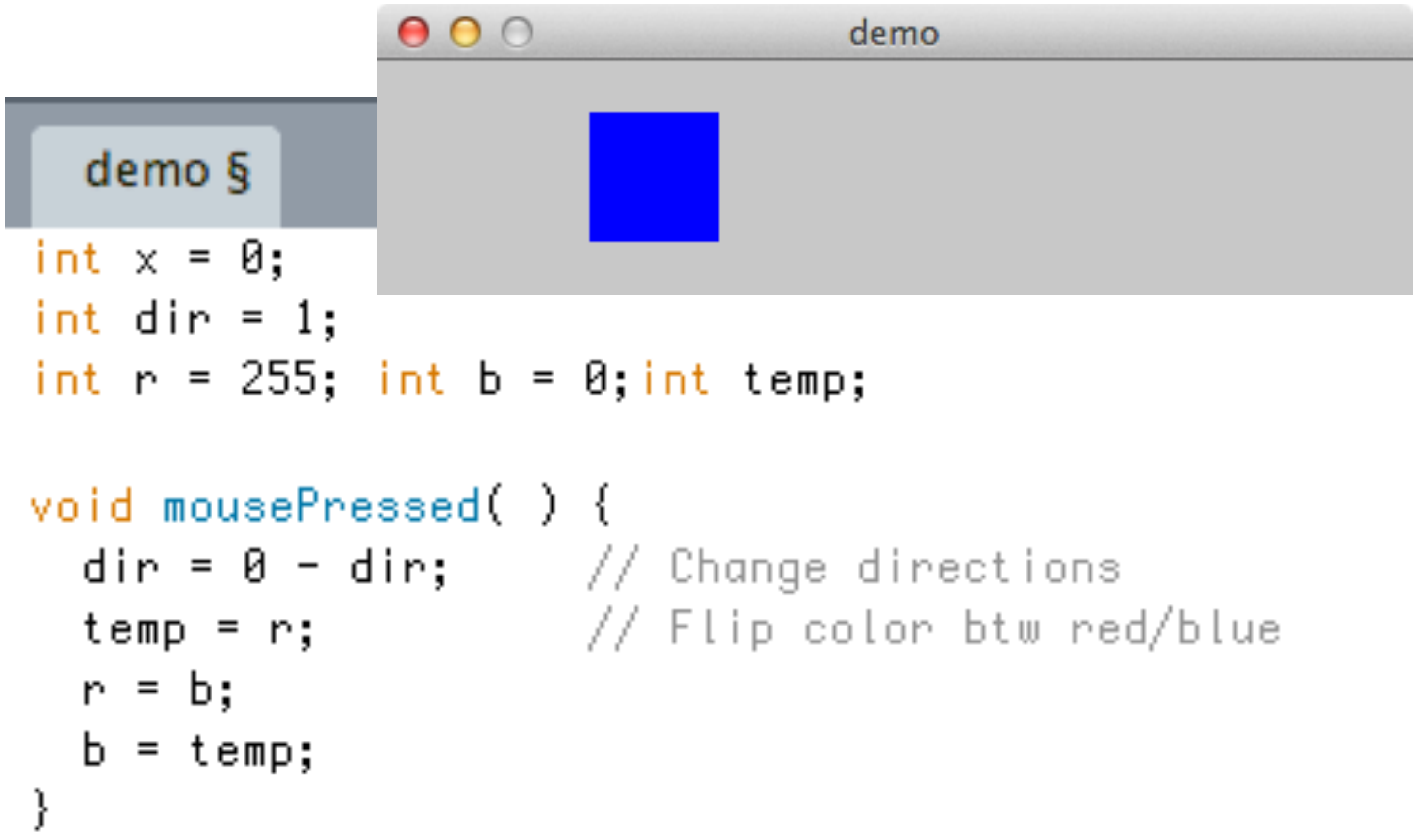
Changing Control

Testing and Repetition

Lawrence Snyder
University of Washington, Seattle

Let's Begin W/ Idea From Last Time

- We saw how to change the color of the square and its direction with a mouse click
- Recall



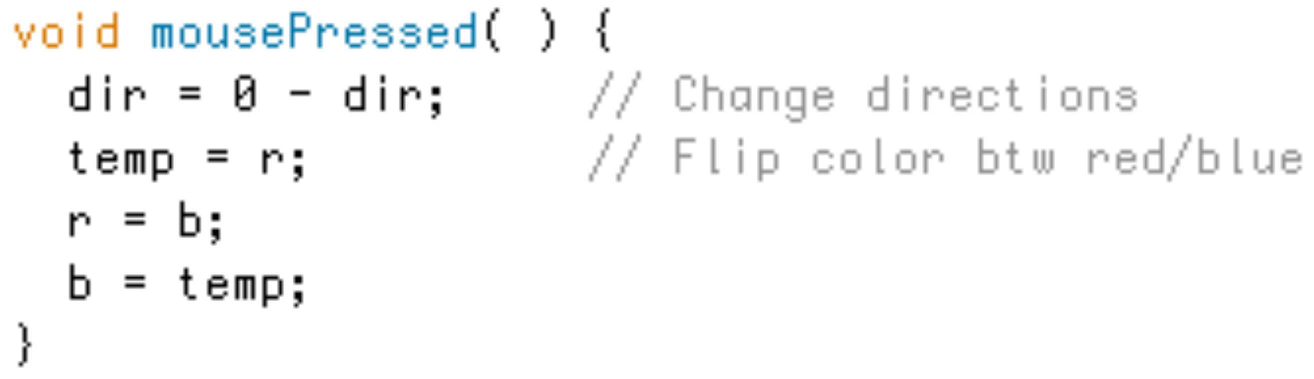
...

```
int x = 0;
int dir = 1;
int r = 255; int b = 0;int temp;

void mousePressed( ) {
  dir = 0 - dir;      // Change directions
  temp = r;           // Flip color btw red/blue
  r = b;
  b = temp;
}
```

This Shows Assignment At Work

- Rule: Assignment always moves information from right to left, as in

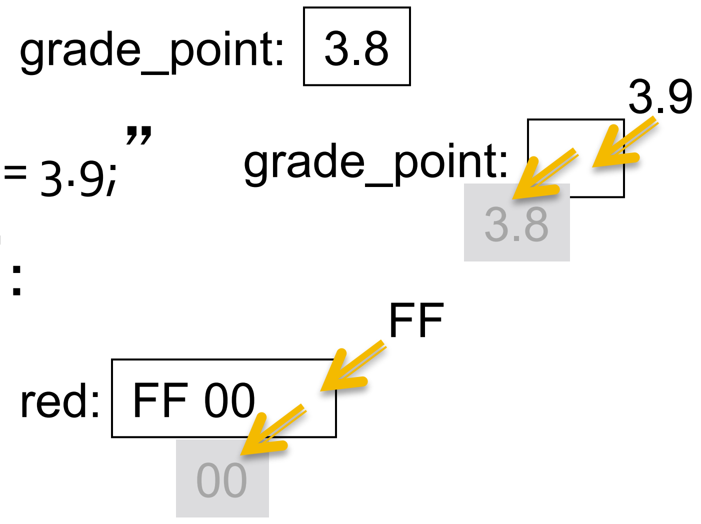


```
void mousePressed( ) {
  dir = 0 - dir;      // Change directions
  temp = r;           // Flip color btw red/blue
  r = b;
  b = temp;
}
```

- dir = 0 – dir;
- Rule: Always evaluate (compute) the right side, then assign the result to the name on the left side … so, 0-dir = dir; IS SO WRONG

Variables …

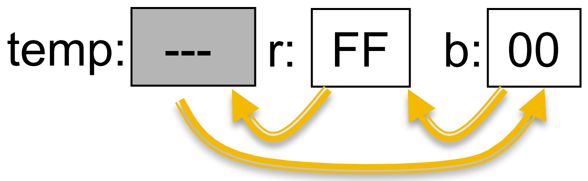
- Variables "contain" their values like a mailbox contains a letter, and so when we change them using assignment, we "push the old value out" and replace it with a new value
  - "Contain their value":
  - "Assign to change: grade_point = 3.9;"
  - "Variables have a data type":



A 3 Statement Exchange

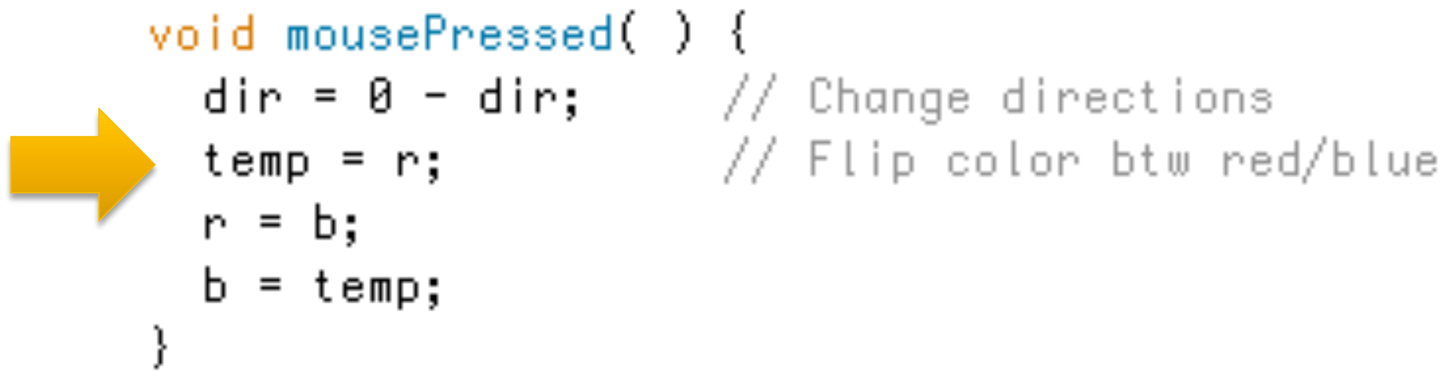
- How does the 3-statement exchange work?

```
void mousePressed( ) {
  dir = 0 - dir;      // Change directions
  temp = r;           // Flip color btw red/blue
  r = b;
  b = temp;
}
```

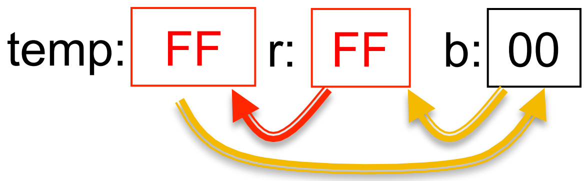


A 3 Statement Exchange

- How does the 3-statement exchange work?

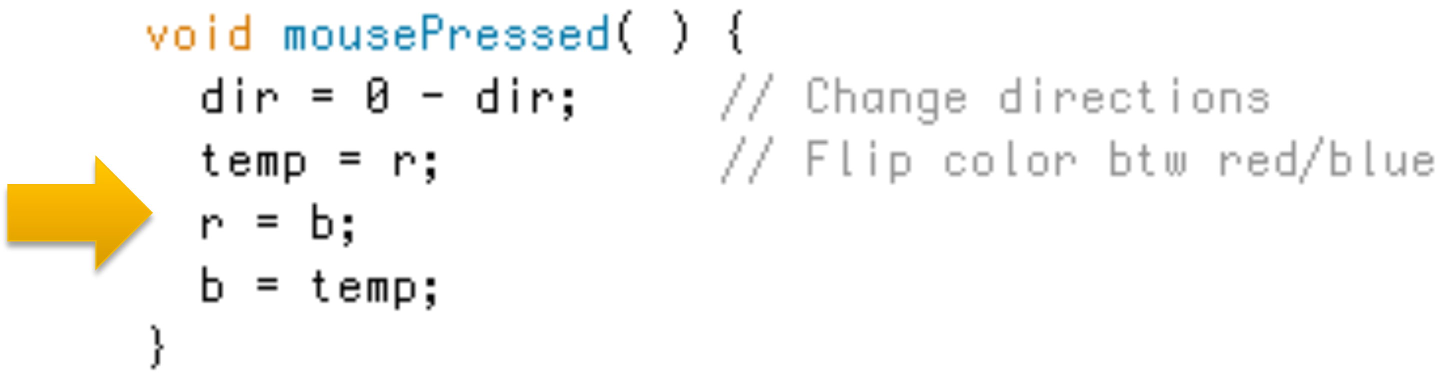


```
void mousePressed( ) {
  dir = 0 - dir;       // Change directions
  temp = r;            // Flip color btw red/blue
  r = b;
  b = temp;
}
```

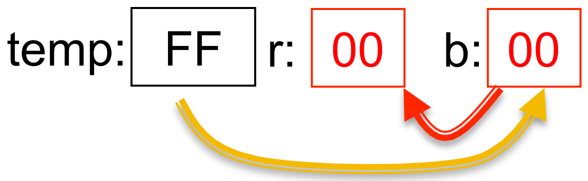


A 3 Statement Exchange

- How does the 3-statement exchange work?



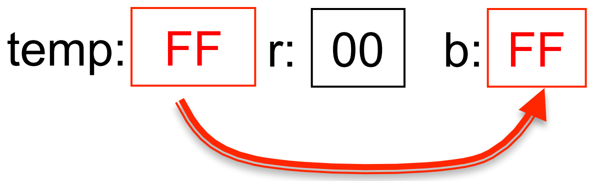
```
void mousePressed( ) {
  dir = 0 - dir;      // Change directions
  temp = r;           // Flip color btw red/blue
  r = b;
  b = temp;
}
```



A 3 Statement Exchange

- How does the 3-statement exchange work?

```
void mousePressed( ) {
  dir = 0 - dir;       // Change directions
  temp = r;            // Flip color btw red/blue
  r = b;
  b = 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

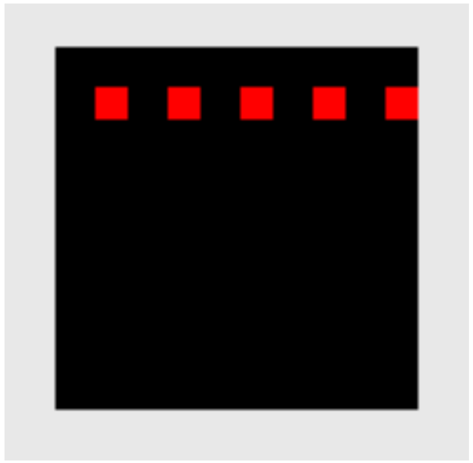
Expressions, the Picture

- 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)

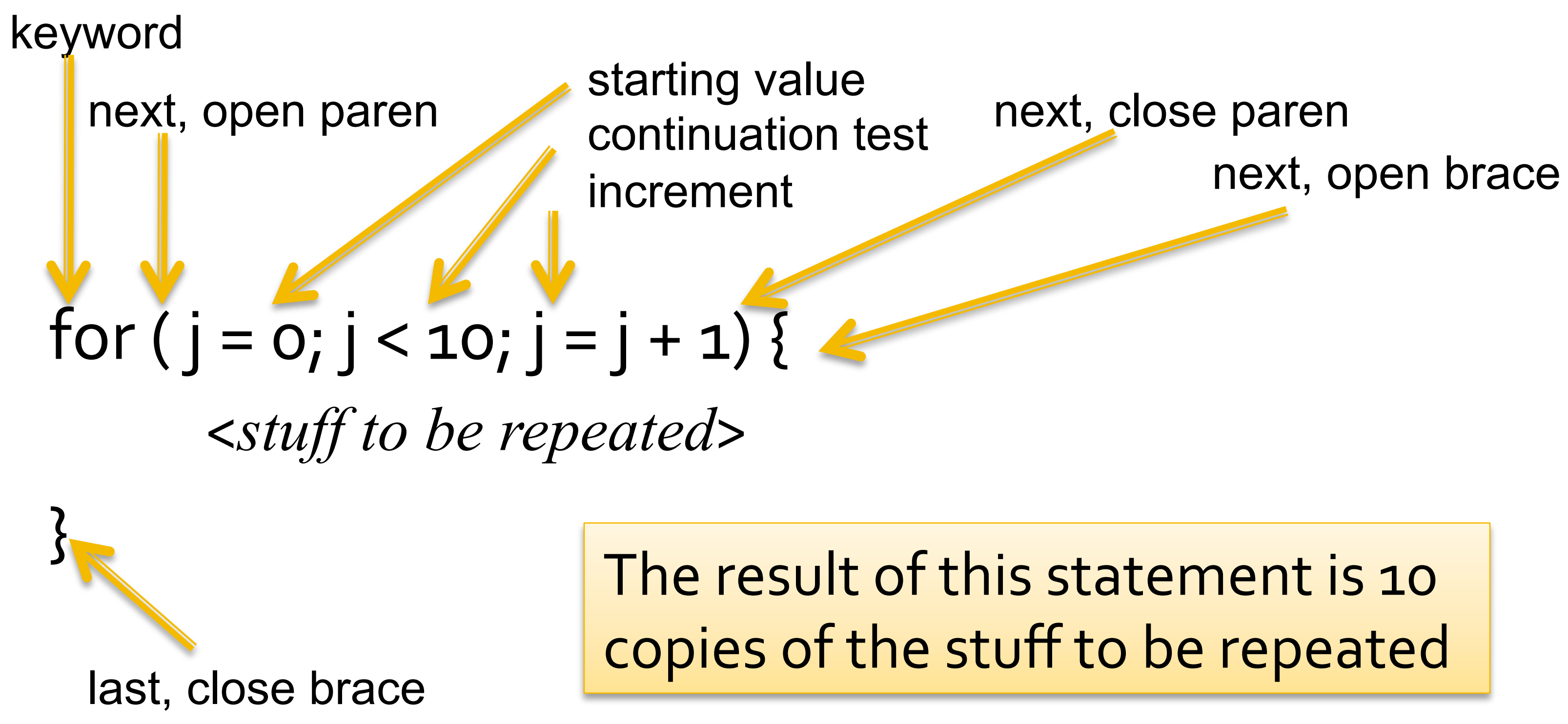
- Repeating commands is a powerful way to use a computer … we could repeat them, but all programming systems have a way to loop:
  - Lightbot 2.0 used recursion, a function calling itself
  - Symbolic Lightbot prefixed a number, 2:Step
- Processing (and other modern languages) use a **for** loop:

```
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 …



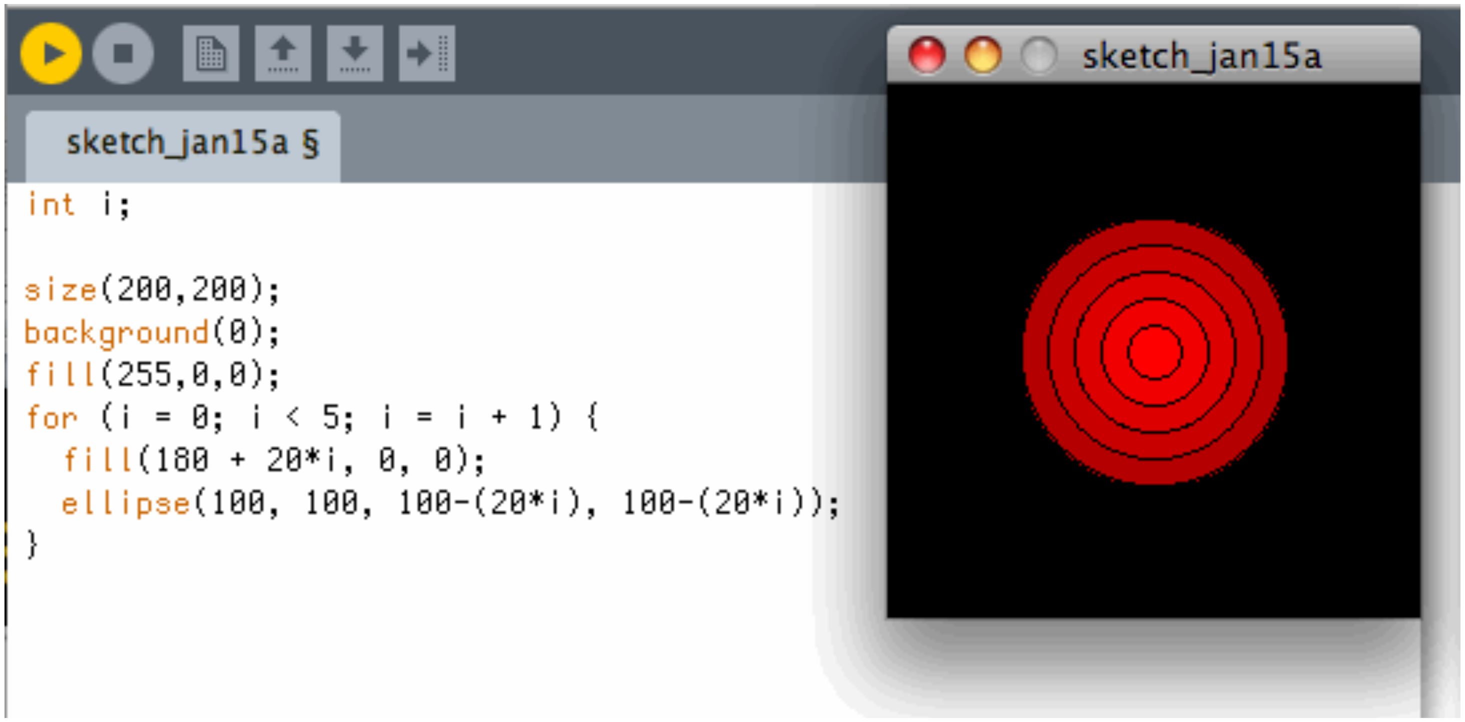
```
for ( j = 0; j < 10; j = j + 1) {
        <stuff to be repeated>

}
```

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

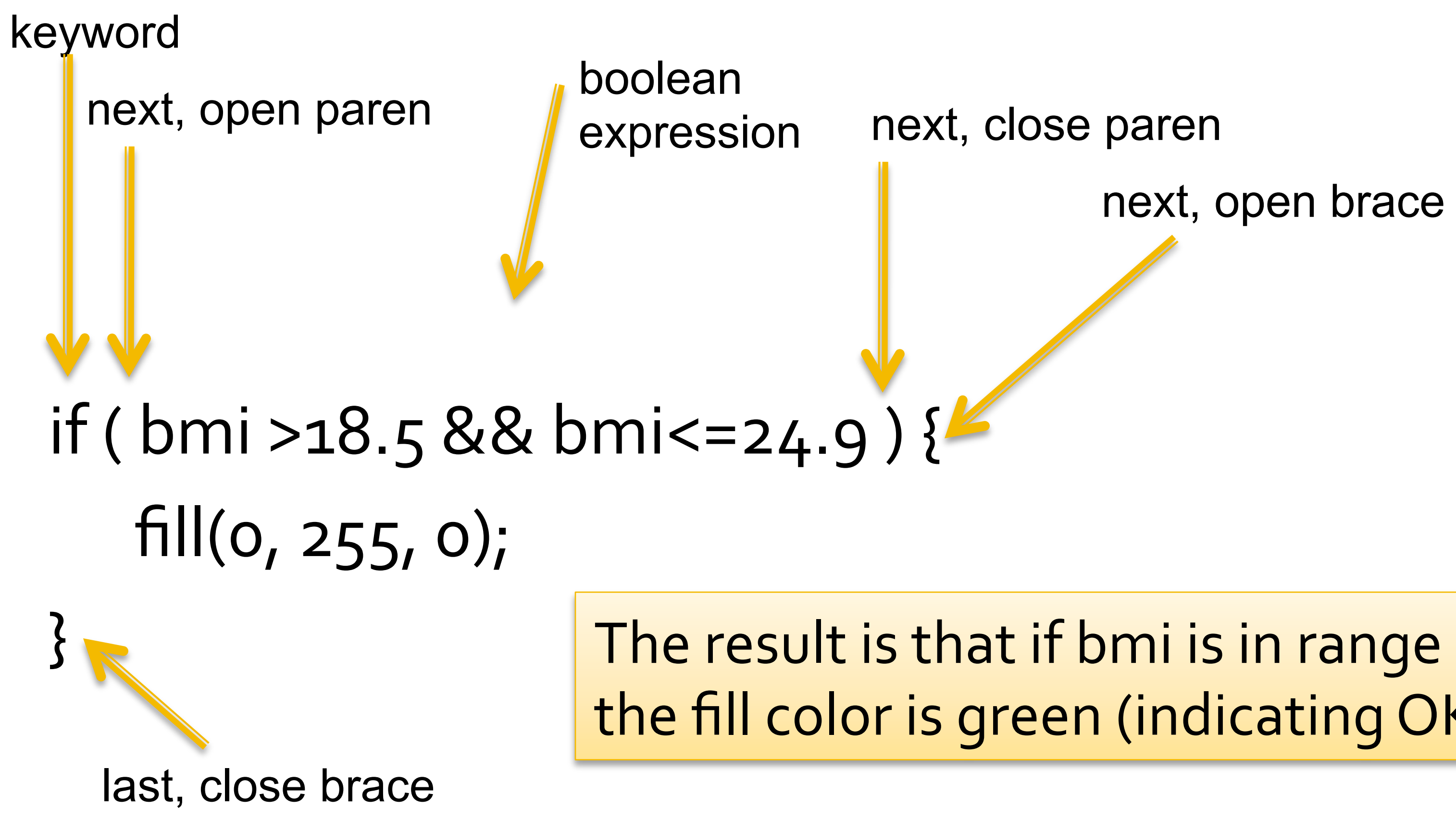
- The instructions of a program are executed sequentially, one after another … sometimes we want to skip some: Say “Hello” to the **If**
- **If** also has a required form

```
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



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

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;
}
```

open brace, immediately after "else"

keyword

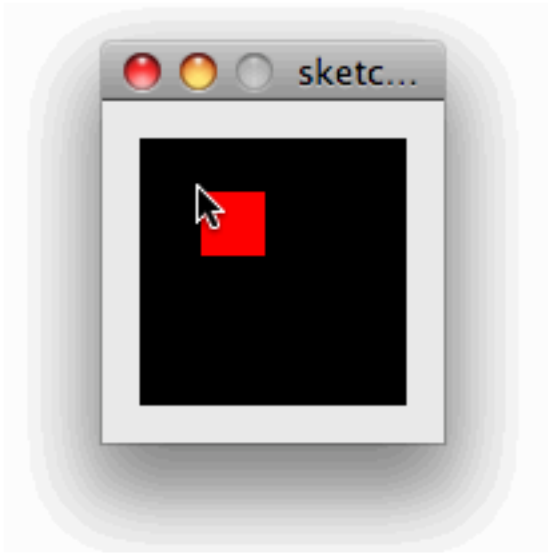
finally, close brace

Else must follow if because it does the test

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

If/Else, The Demo

- Let's go to processing for an 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!