Expressions & Control Flow

CSE 120 Winter 2018

Instructor:	**Teaching Assistants:**		
Justin Hsia	Anupam Gupta,	Cheng Ni,	Eugene Oh,
	Sam Wolfson,	Sophie Tian,	Teagan Horkan

Twitter: More than 677,000 U.S. users engaged with Russian troll accounts

"Twitter said… it's notifying 677,775 people in the U.S. who either followed, retweeted or liked a tweet from accounts of the Kremlin-linked troll farm known as the Internet Research Agency during the 2016 election period… The company also said that… it's identified an additional 13,512 Russian-linked bot accounts that tweeted around the election, bringing the total to 50,258.

"Facebook unveiled a portal last month to allow users to learn of any Facebook or Instagram contact they may have had with Russian internet trolls. Facebook has said that Russian-linked posts were viewed by up to 126 million people during that period."

- https://www.politico.com/story/2018/01/19/twitter-users-russian-trolls-437247

Administrivia

- Assignments:
  - Animal Functions due tonight (1/22) ← before 11:59 pm
  - Reading Check 3 due *before lab* on Thursday (1/25) ← either 2:30 pm (AA) or 4:00 pm (AB)
  - Jumping Monster due Friday (1/26) ↑ significantly harder!
- "Big Ideas" this week: The Internet

Outline

- **Expressions & Operators**
- Conditionals
  - if-statement
- Loops
  - while-loop
  - for-loop

Expressions

- "An expression is a combination of one or more *values*, *constants*, *variables* ✓, *operators* (today's lecture), and *functions* ✓ that the programming language interprets and computes to produce another value."
  - https://en.wikipedia.org/wiki/Expression_(computer_science)

- Expressions are *evaluated* and resulting value is used
  - Assignment: `x = x + 1;`
  - Assignment: `x_pos = min(x_pos + 3, 460);`
  - Argument: `ellipse(50+x, 50+y, 50, 50);`
  - Argument: `mouse(rowX+4*50,rowY,rowC);`

expressions must be evaluated first

larger expression

Operators

- Built-in "functions" in Processing that use special symbols:
  - Multiplicative: * mult, / div, % modulus
  - Additive: + add, - sub

new

  - Relational: < less than, > greater than, <= less than or equal to, >= greater than or equal to
  - Equality: == equal to, != not equal to
  - Logical: && and, || or, ! not

- Operators can only be used with certain data types and return certain data types
  - Multiplicative/Additive: 1+2 give numbers, get number (3)
  - Relational: 1 < 5 give numbers, get Boolean (true)
  - Logical: true && true give Boolean, get Boolean (true)
  - Equality: color(0) == color(255) give same type, get Boolean (false)

Operators

- Built-in “functions” in Processing that use special symbols:
  - Multiplicative: * mult, / div, % modulus
  - Additive: + add, - sub
  - Relational: < less than, > greater than, <= less than or equal to, >= greater than or equal to
  - Equality: == equal to, != not equal to
  - Logical: && and, || or, ! not

- Logical operators use Boolean values (true, false)

put in-between

AND (&&)

x	y	x && y
false	false	false
false	true	false
true	false	false
true	true	true

OR (||)

x	y	x || y
false	false	false
false	true	true
true	false	true
true	true	true

NOT (!)

put in front

x	!x
false	true
true	false

Operators

- Built-in “functions” in Processing that use special symbols:
  - Multiplicative: * mult, / div, % modulus
  - Additive: + add, - sub
  - Relational: < less than, > greater than, <= less than or equal to, >= greater than or equal to
  - Equality: == equal to, != not equal to
  - Logical: && and, || or, ! not

- In expressions, use parentheses for evaluation ordering and readability
  - e.g. `x + (y * z)` is the same as `x + y * z`, but easier to read

    order of operations!

    `(x + y) * z` is required if you want addition to happen first.

Modulus Operator: %

- x % y is read as "x mod y" and returns the remainder after y divides x
  - For short, we say "mod" instead of modulus

0/3 = 0 remainder 0
1/3 = 1 remainder 1

- Practice:
  - 0 % 3 is 0
  - 1 % 3 is 1
  - 2 % 3 is 2
  - 3 % 3 is 0
  - 4 % 3 is 1
  - 5 % 3 is 2
  - 6 % 3 is 0

Modulus Operator: %

- `x % y` is read as "x mod y" and returns the remainder after y divides x
  - For short, we say "mod" instead of modulus

- Example Uses:
  - Parity: is even if divisible by 2
    Number `n` is even if `n%2 == 0`
  - Leap Year: divisible by 4 (e.g. 2016, 2020)
    Year `year` is a leap year if `year%4 == 0`
  - Chinese Zodiac: (12 Zodiac animals)
    `year1` and `year2` are the same animal if `year1%12 == year2%12`

Modulus Example in Processing

- Use mod to “wrap around”
  - Replace min/max function to “connect” edges of drawing canvas

right edge of canvas

x_pos = 459;

462

```
x_pos = min(x_pos + 3, 460); // stores 460
x_pos = (x_pos + 3) % 460;   // stores 2
```

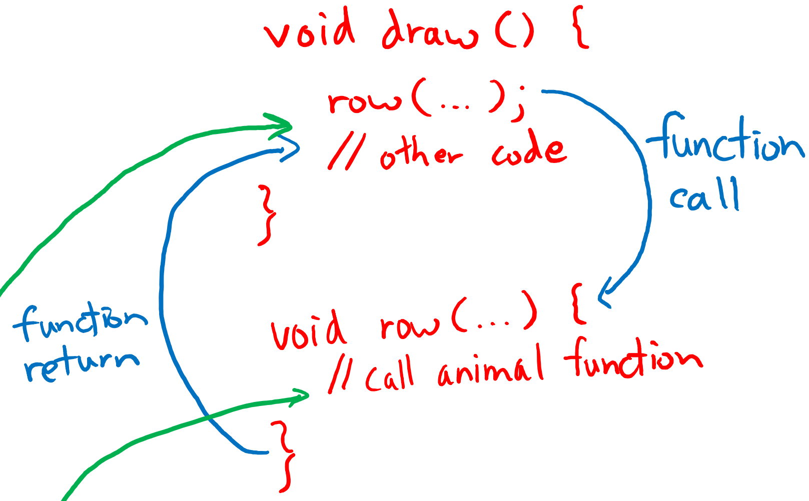
462

left side of canvas

Control Flow

- The order in which instructions are executed

- We typically say that a program is executed in sequence from top to bottom, but that's not always the case:
  - ✓ Function calls and `return` calls
  - today { Conditional/branching statements
  - Loops



```
void draw() {
  row(...);
  // other code
}

void row(...) {
  // call animal function
}
```

- Curly braces {} are used to group statements
  - Help parse control flow
  - Remember to use indentation!

Outline

- Expressions & Operators
- **Conditionals**
  - **if-statement**
- Loops
  - while-loop
  - for-loop

If-Statements

- Sometimes you don't want to execute *every* instruction
  - Situationally-dependent

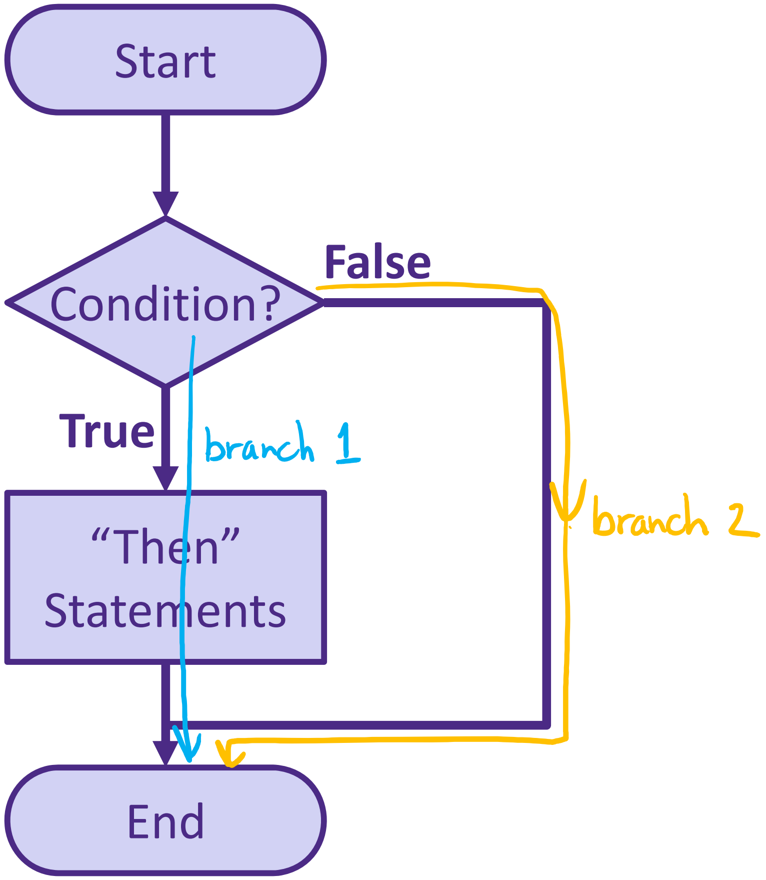
- Conditionals give the programmer the ability to make decisions
  - The next instruction executed depends on a specified *condition*
    - The condition must evaluate to a `boolean` (*i.e.* `true` or `false`)
    - Sometimes referred to as "branching"
  - This generally lines up well with natural language intuition

If-Statements

- Basic form:

false true

```
if(condition) {
    // "then"
    // statements
}
```



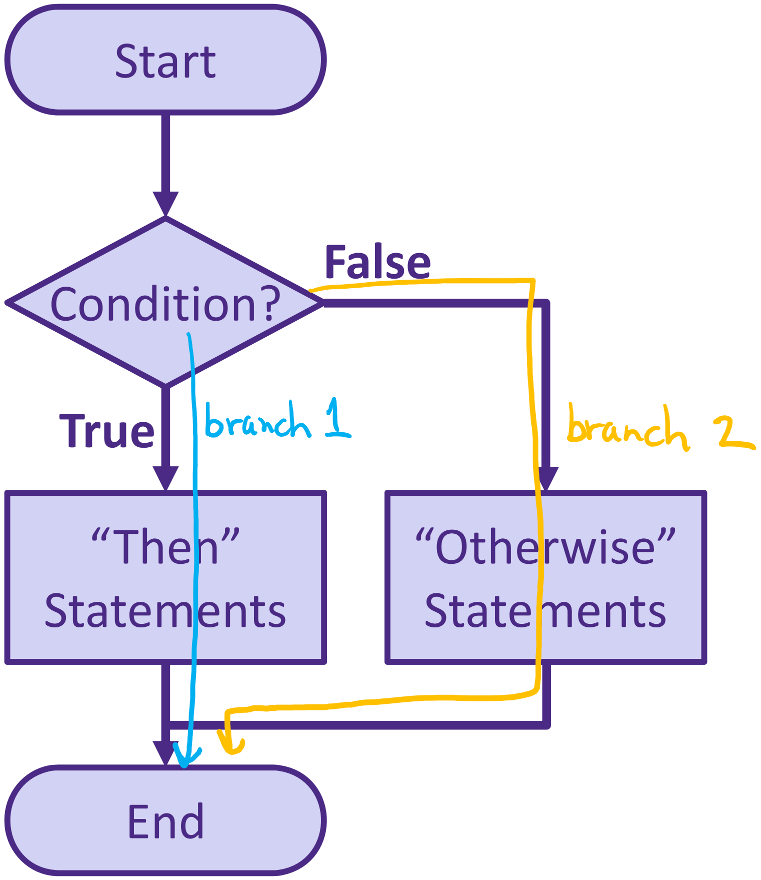
- Example conditions:
  - Variable: `if( done == true )` boolean
  - Variable: `if( done )` equivalent
  - Expression: `if( x_pos > 460 )`
  - Expression: `if( x_pos > 100 && y_pos > 100 )` number cond 1 AND cond 2

If-Statements

- With `else` clause:

true false

```
if(condition) {
    // "then"
    // statements
} else {
    // "otherwise"
    // statements
}
```



If-Statements

- With `else if` clause:

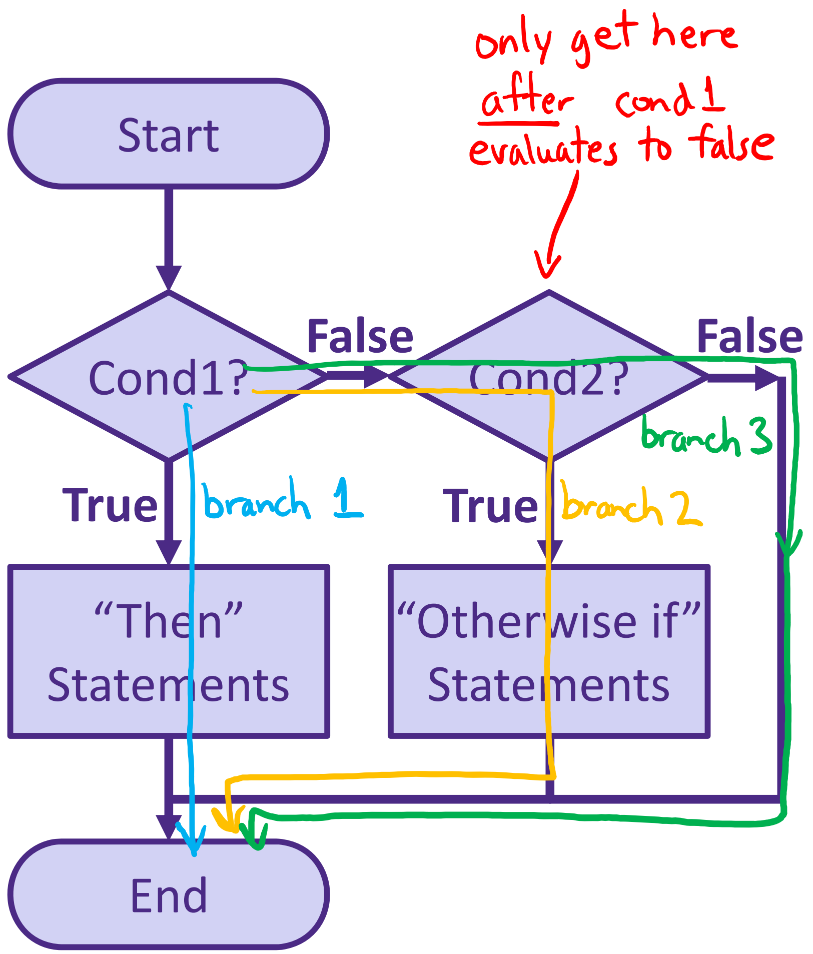
```
if(cond1) {
   // "then"
   // statements
} else if(cond2) {
   // "otherwise if"
   // statements
}
```

true

false false

true

false

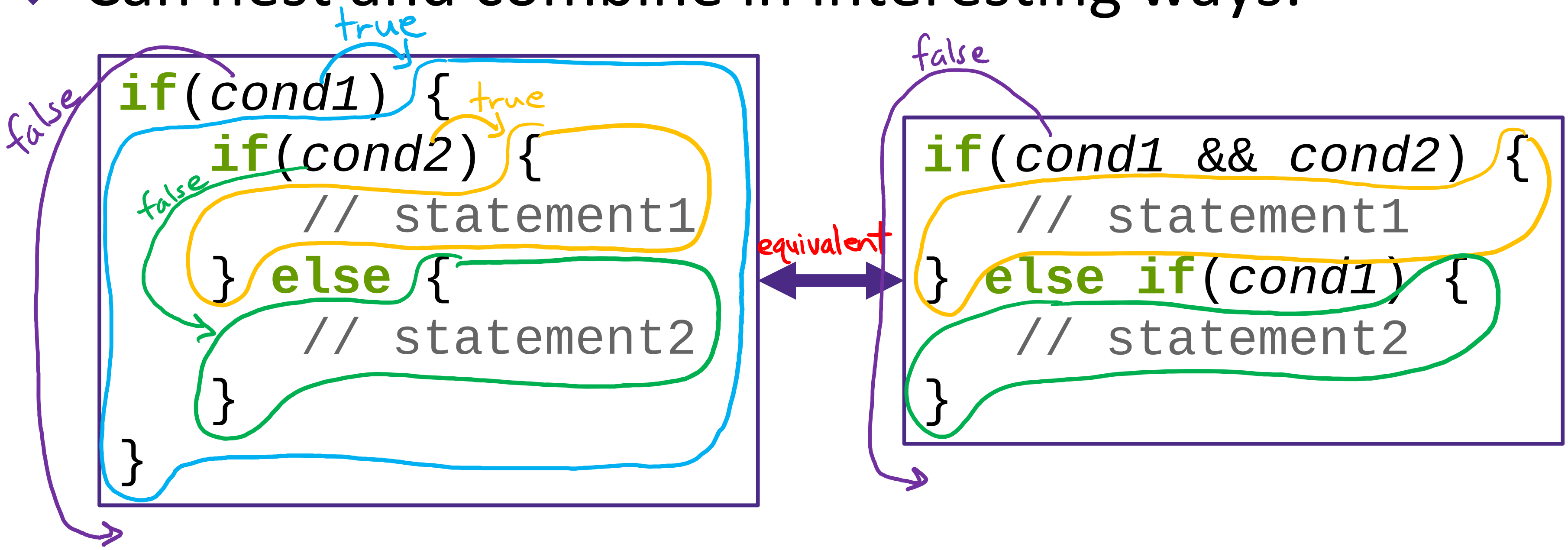


If-Statements

- Notice that conditionals *always* go from Start to End
  - Choose one of many *branches*
  - A conditional must have a single `if`, as many `else if` as desired, and at most one `else`

↑ "catch all" / default

- Can nest and combine in interesting ways:



```
if(cond1) {
   if(cond2) {
      // statement1
   } else {
      // statement2
   }
}
```

```
if(cond1 && cond2) {
   // statement1
} else if(cond1) {
   // statement2
}
```

Peer Instruction Question

- Which value of x will get the following code to print out "Maybe"?

A. 1 No

[✓ B. 3] Maybe

C. 5 Yes

D. 7 No

E. We're lost...

```
if (x == 5) {
   print("Yes");
} else if ((x >= 6) || (x < 2)) {
   print("No");
} else {
   print("Maybe");
}
```

equal to

F, F, T, F

greater than or equal to

F, F, T

less than

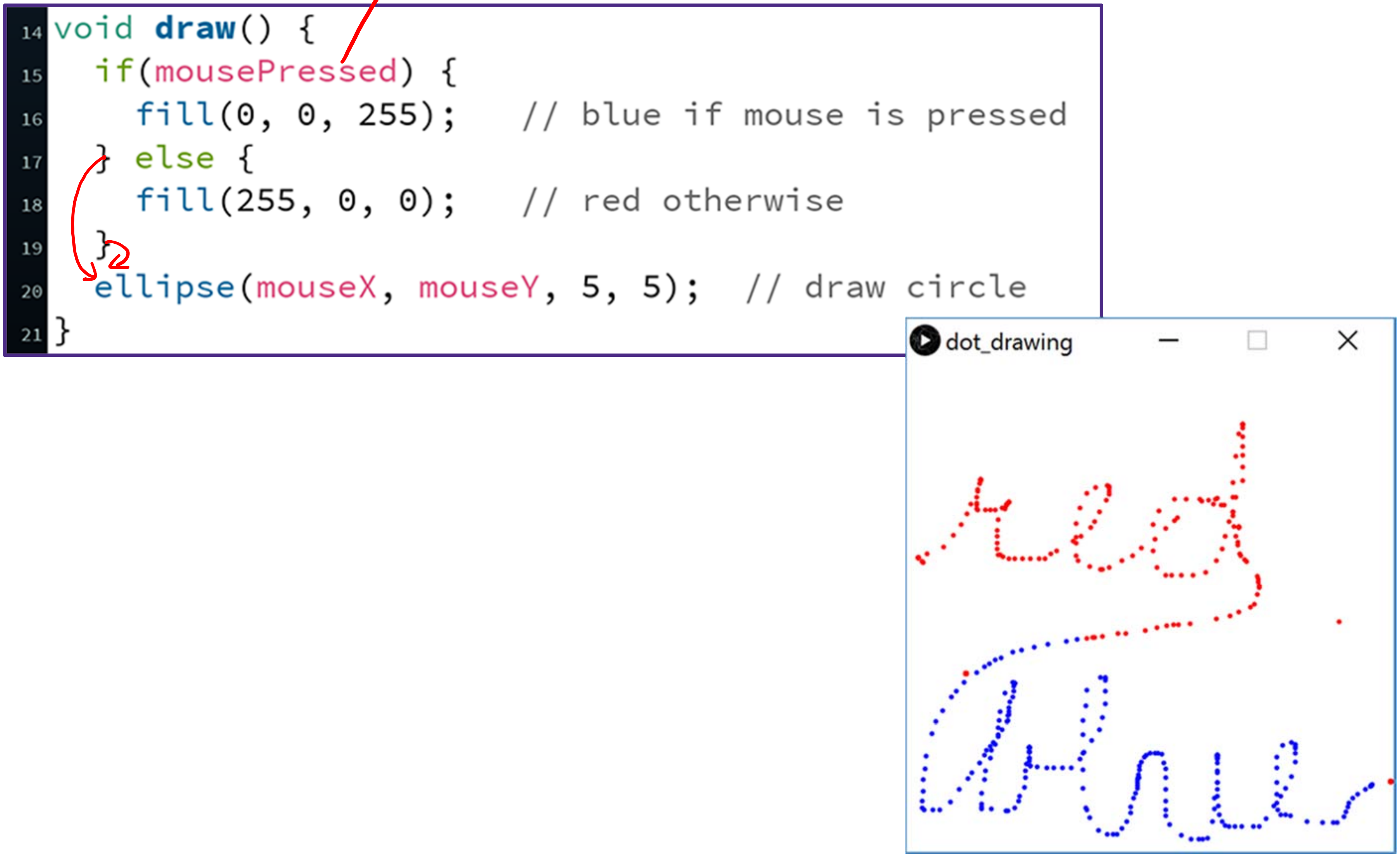
T, F, F

OR

- Think for a minute, then discuss with your neighbor(s)
  - Vote at http://PollEv.com/justinh

Processing Demo: Drawing Dots

true, if mouse is physically being pressed down
false, otherwise



```
void draw() {
  if(mousePressed) {
    fill(0, 0, 255);    // blue if mouse is pressed
  } else {
    fill(255, 0, 0);    // red otherwise
  }
  ellipse(mouseX, mouseY, 5, 5);  // draw circle
}
```

Outline

- Expressions & Operators
- Conditionals
  - if-statement
- **Loops**
  - **while-loop**
  - **for-loop**

Looping

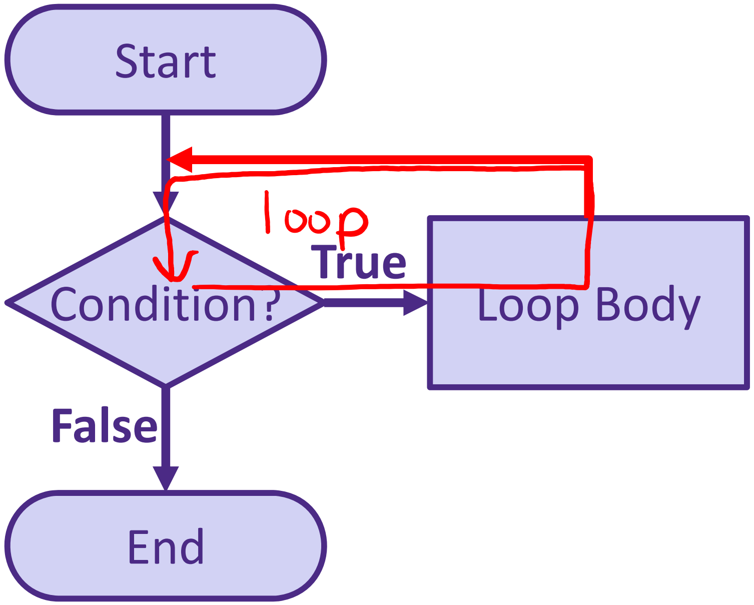
- Sometimes we want to do the same (or similar) things over and over again
    - Looping saves us time from writing out all of the instructions

- Loops control a sequence of *repetitions*

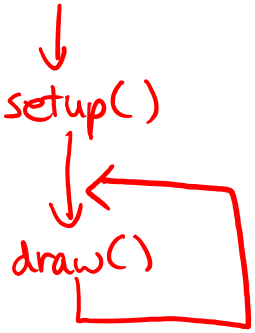
While-Loop

- Basic form:

```
                x<10
while(condition) {
    // loop
    // body
   x = x + 1;
}
```



- Repeat loop body until condition is `false`
  - Must make sure to update conditional variable(s) in loop body, otherwise you cause an infinite loop
- `draw()` is basically a `while(true)` loop

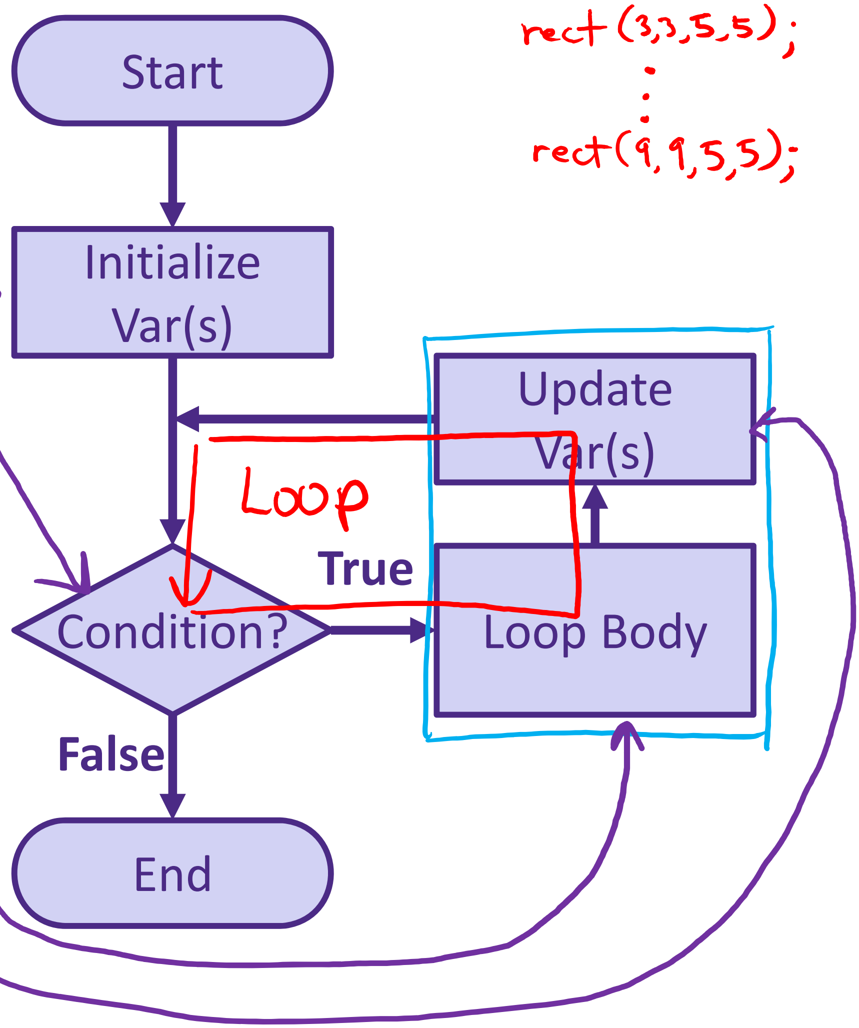


While-Loop

example: rect(1,1,5,5);
rect(2,2,5,5);
rect(3,3,5,5);
:
rect(9,9,5,5);

- More general form:

```
 int x = 1;
 // init cond var(s)
                x < 10
while(condition) {
    rect(x,x,5,5);
    // loop body
    x = x + 1;
    // update var(s)
}
```

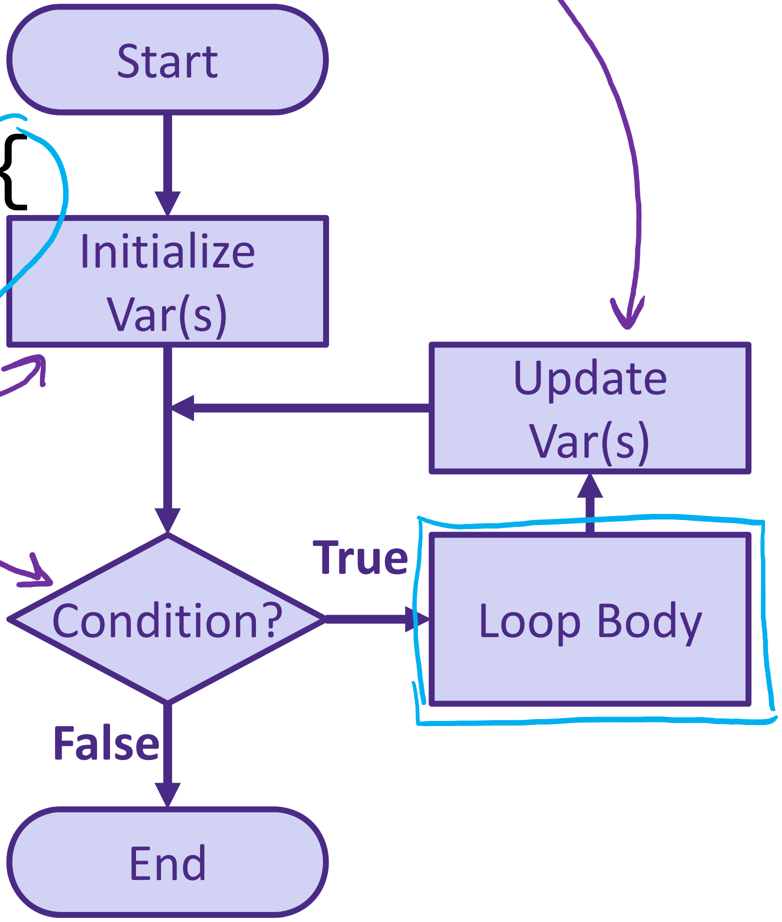


- This occurs so commonly that we create a separate syntax for it!

For-Loop

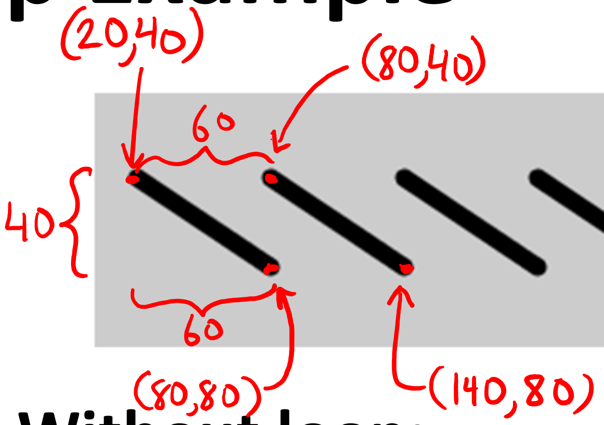
separated by semicolons

```
for(init; cond; update){
    // loop body
}
```



- First runs *init* expression(s)
- Then checks *cond*
- If `true`, runs loop body followed by update statement(s)

For-Loop Example



Without loop:

```
line(20, 40, 80, 80);
line(80, 40, 140, 80);
line(140, 40, 200, 80);
line(200, 40, 260, 80);
line(260, 40, 320, 80);
line(320, 40, 380, 80);
line(380, 40, 440, 80);
```

+60 +60

+60 +60

always 40 always 80

With loop:

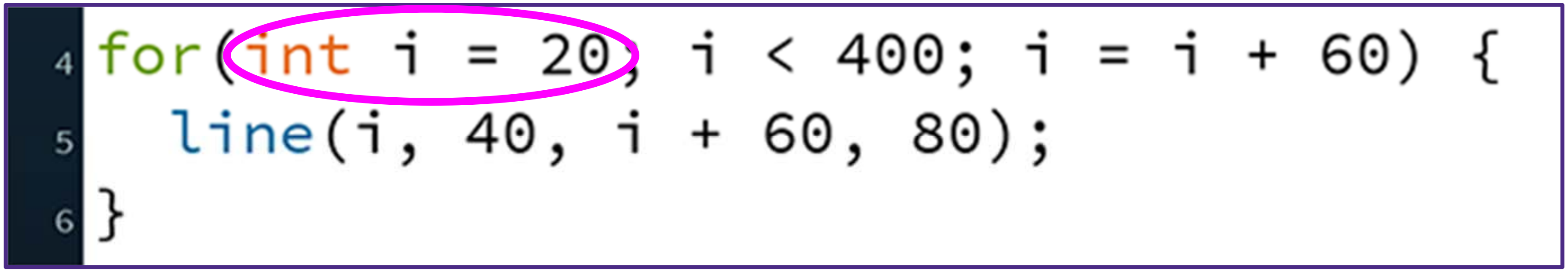
init cond update stops once i=440

```
for(int i = 20; i < 400; i = i + 60) {
  line(i, 40, i + 60, 80);
}
```

20 20

80 80

Understanding the For-Loop



```
for (int i = 20; i < 400; i = i + 60) {
  line(i, 40, i + 60, 80);
}
```

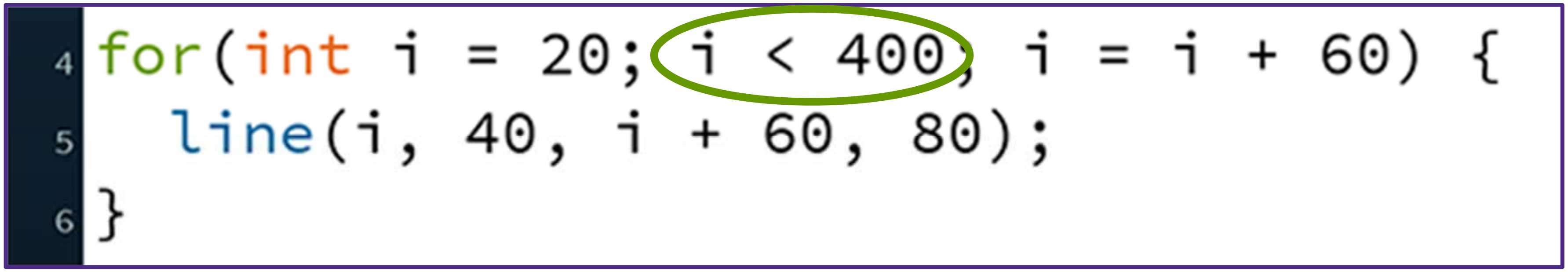
initialization

- Choice of variable name(s) is not critical
  - Represent the value(s) that vary between different executions of the loop body
  - Think of as temporary variable(s)

- Variable scope: variable `i` only exists *within this loop*

Understanding the For-Loop

condition

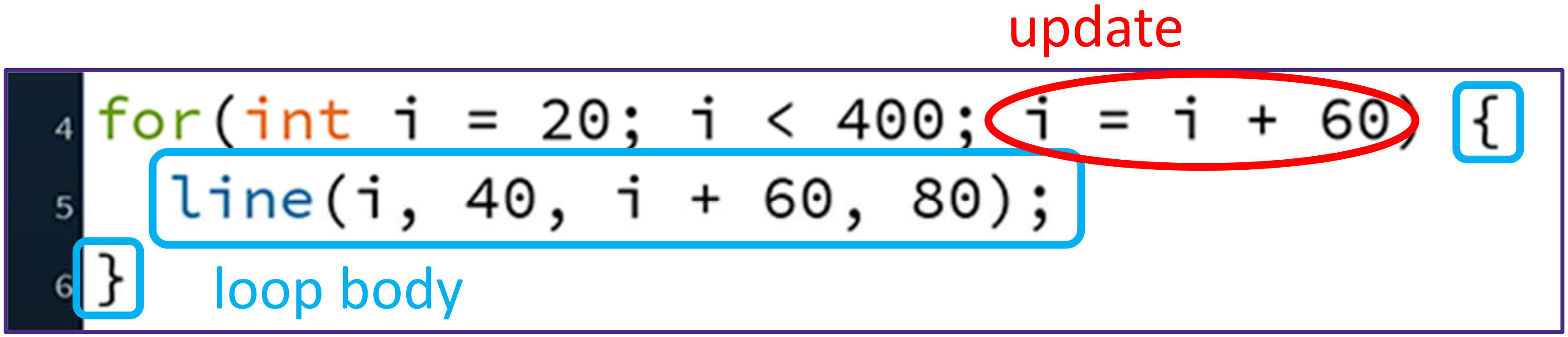


```
for(int i = 20; i < 400; i = i + 60) {
  line(i, 40, i + 60, 80);
}
```

- Condition evaluated *before* the loop body and must evaluate to `true` or `false`
  - Reminder:

>	greater than
<	less than
>=	greater than or equal to
>=	less than or equal to
==	equal to
!=	not equal to

Understanding the For-Loop



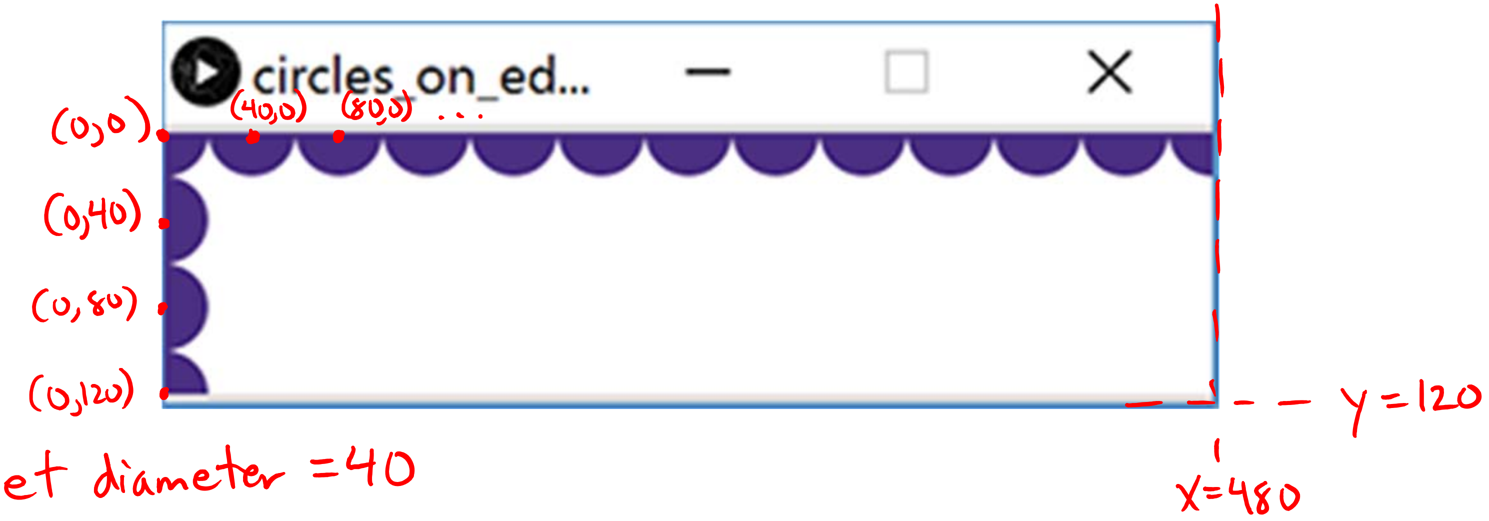
```
for(int i = 20; i < 400; i = i + 60) {
  line(i, 40, i + 60, 80);
}
```

update

loop body

- Update is an assignment that is executed *after* the loop body
- Loop body is enclosed by curly braces {} and should be *indented* for readability

Processing Demo: Circles on Canvas Edge



let diameter =40

left edge:

want
```
ellipse (0,0,40,40);
ellipse (0,40, 40,40);
ellipse (0,80, 40,40);
ellipse (0,120, 40,40);
```

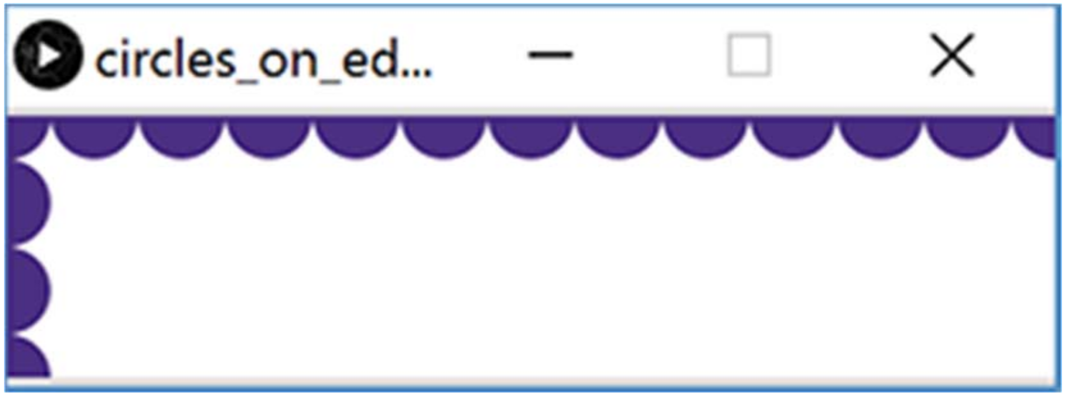
↑ +40 each time in this argument

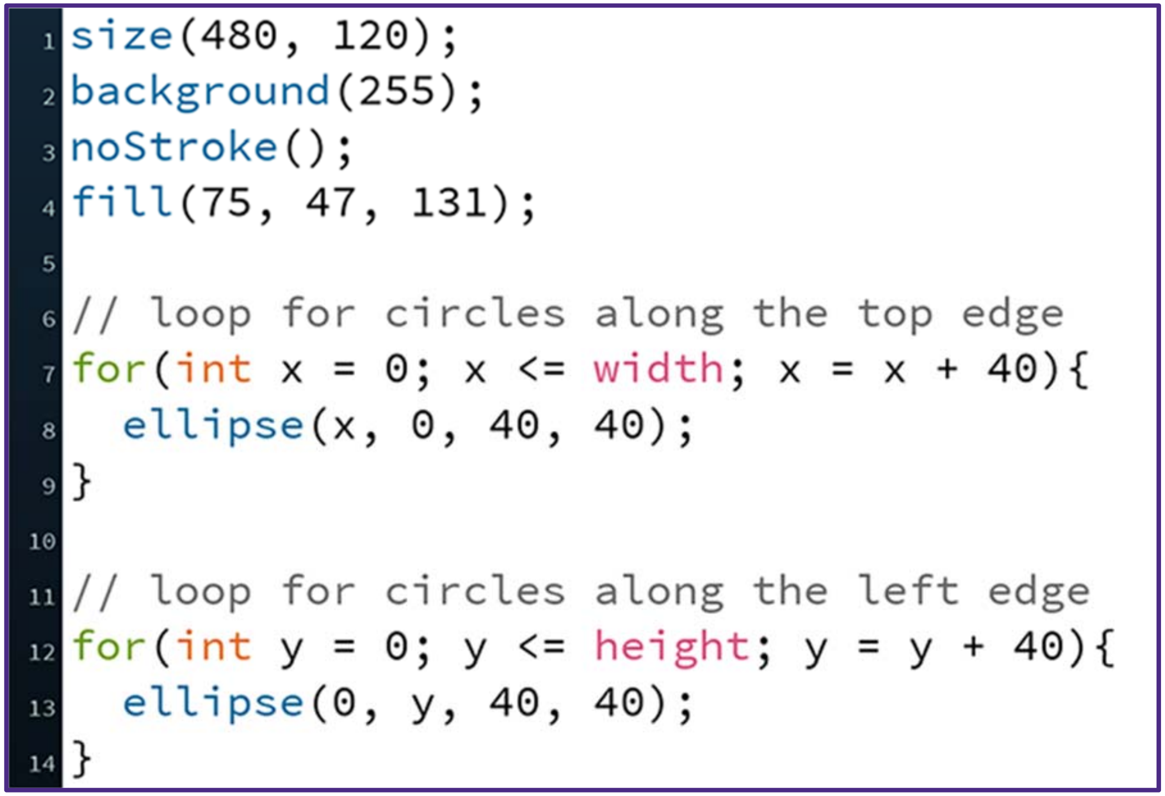
equivalent ⟺

```
for(int i=0; i <= 120; i=i+1) {
    ellipse (0, 40*i, 40, 40);
}
```

could also substitute 40 with a variable for diameter!

Processing Demo: Circles on Canvas Edge





```
size(480, 120);
background(255);
noStroke();
fill(75, 47, 131);

// loop for circles along the top edge
for(int x = 0; x <= width; x = x + 40){
  ellipse(x, 0, 40, 40);
}

// loop for circles along the left edge
for(int y = 0; y <= height; y = y + 40){
  ellipse(0, y, 40, 40);
}
```