

Expressions & Conditionals

CSE 120 Winter 2019

Instructor:

Justin Hsia

Teaching Assistants:

Ann Shan,

Sam Wolfson,

Eunia Lee,

Travis McGaha

Pei Lee Yap,

Delivery Robot Engulfed in Flames, Honored on Campus With Candlelight Vigil

“The University of California, Berkeley, lost a beloved member of their campus last week, when a delivery robot was engulfed in flames outside the student union.

“The courier rovers started delivering food to students about two years ago. The service seems to have been operating relatively smoothly until... a Kiwibot spontaneously combusted while on the job.

“After an investigation, the company revealed ... ‘a defective battery was put in place of a functioning one. This caused an exceedingly rare occurrence of the battery experiencing thermal runaway.’ ”

- <https://gizmodo.com/delivery-robot-engulfed-in-flames-honored-on-campus-wi-1831145871>



James Wenzel 🐱
@ratherbright



so a delivery robot caught fire on berkeley's campus and students set up a candlelight vigil for it

♡ 6,092 11:42 PM - Dec 14, 2018

Administrivia

- ❖ Assignments:
 - Animal Functions due tonight (1/28)
 - Reading Check 4 due Thursday @ 3:30 (1/31)
 - Jumping Monster due Friday (2/1)
 ↑ significantly harder!
- ❖ “Big Ideas” this week: Digital Distribution

Outline

- ❖ **Expressions & Operators**
- ❖ **Conditionals**
 - `if`
 - `else`
 - `else if`

Expressions

- ❖ “An **expression** is a combination of one or more values, constants, variables, operators, and functions that the programming language interprets and computes to produce another value.”

- [https://en.wikipedia.org/wiki/Expression_\(computer_science\)](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:

```
drawMouse(rowX+4*50, rowY, rowC);
```

Operators

❖ Built-in “functions” in Processing that use special symbols:

new {

- 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

❖ 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)

AND (&&)			OR ()			NOT (!)	
x	y	x && y	x	y	x y	x	!x
false	false		false	false		false	true
false	true		false	true		false	true
true	false		true	false		true	false
true	true		true	true		true	false

Handwritten notes:

- Red arrow from "put in-between" points to the space between x and y in the AND and OR tables.
- Red arrow from "put in front" points to the ! in the NOT table.

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 “ $x \bmod y$ ” and returns the remainder after y divides x

- For short, we say “mod” instead of modulus

❖ Example Uses:

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

Conditionals Worksheet

❖ Work just on **Page 1 (Questions 1-6)**

❖ Operators:

- Arithmetic: + - * / %
- Relational: < > <= >=
- Equality: == !=
- Logical: && || !

❖ Data Types:

- Arithmetic: give numbers, get number
- Relational: give numbers, get Boolean
- Logical: give Boolean, get Boolean
- Equality: give same type, get Boolean

Modulus Example in Processing

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

`x_pos = 459;`

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

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

Handwritten annotations:

- `x_pos = 459;` is written in red.
- `x_pos + 3` in the first code line is circled in red, with `462` written above it.
- `(x_pos + 3)` in the second code line is circled in red, with `462` written below it.
- `460` in the first code line is written in red.
- `2` in the second code line is written in red.
- A red arrow points from the text “right edge of canvas” down to the `460` in the first code line.
- A red arrow points from the text “left side of canvas” up to the `2` in the second code line.

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:

last week ■ Function calls and `return` calls

today ★ Conditional/branching statements

next week ■ Loops

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

The diagram illustrates the flow of control between two functions. The top function is `void draw() { row(...); // other code }`. The bottom function is `void row(...) { // call animal function }`. A blue arrow labeled "function call" points from the `row(...)` line in the `draw()` function to the `void row(...)` function. A green arrow labeled "function return" points from the closing brace of the `row(...)` function back to the `draw()` function, indicating the return path.

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

Outline

- ❖ Expressions & Operators
- ❖ **Conditionals**
 - `if`
 - `else`
 - `else if`

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

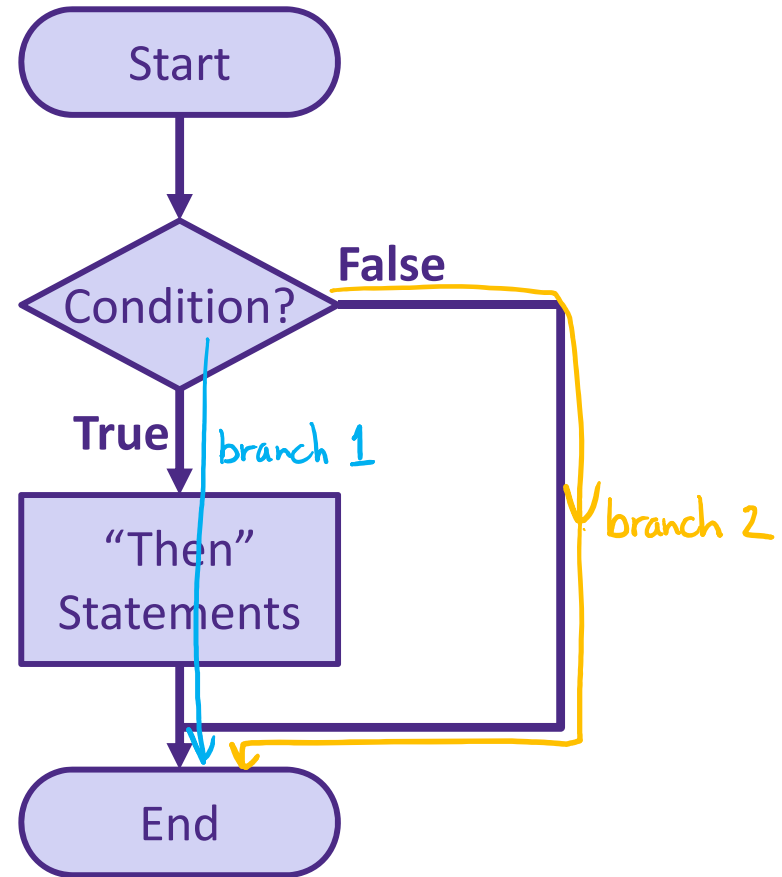
If-Statements

❖ Basic form:

```

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

Handwritten annotations: A blue arrow labeled "true" points from the condition to the body. A yellow arrow labeled "false" points from the condition to the closing brace.



❖ Example conditions:

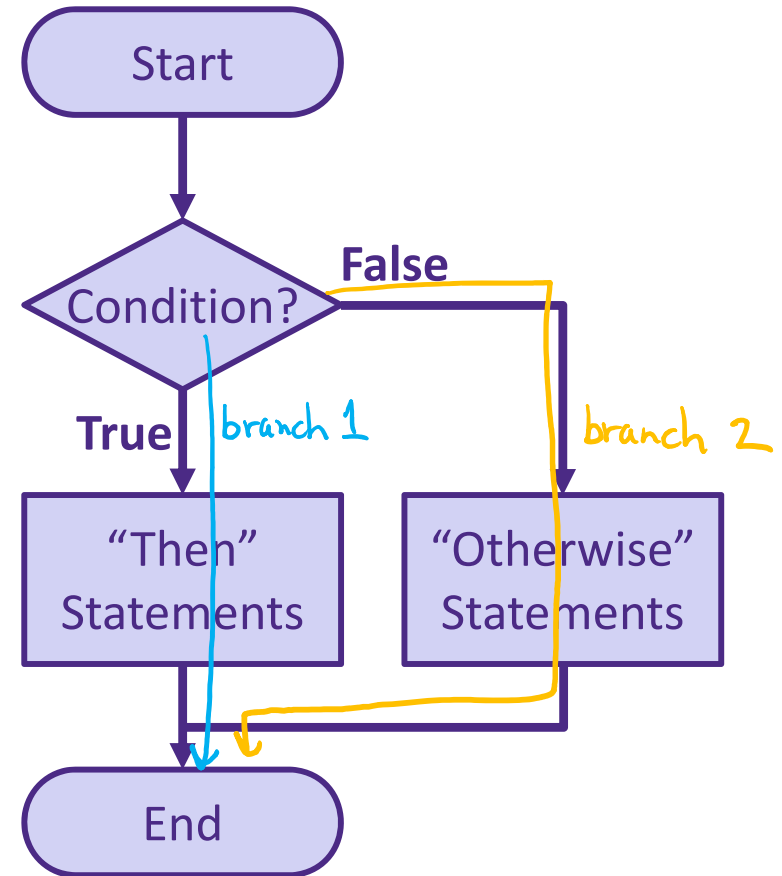
- Variable: `if (done == true)` } *boolean* } equivalent
- Variable: `if (done)` }
- Expression: `if (x_pos > 460)`
- Expression: `if (x_pos > 100 && y_pos > 100)`
number *cond 1* *AND* *cond 2*

If-Statements

- ❖ With else clause:

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

Handwritten annotations: A blue arrow labeled "true" points from the condition to the "then" block. A yellow arrow labeled "false" points from the condition to the "else" block.



If-Statements

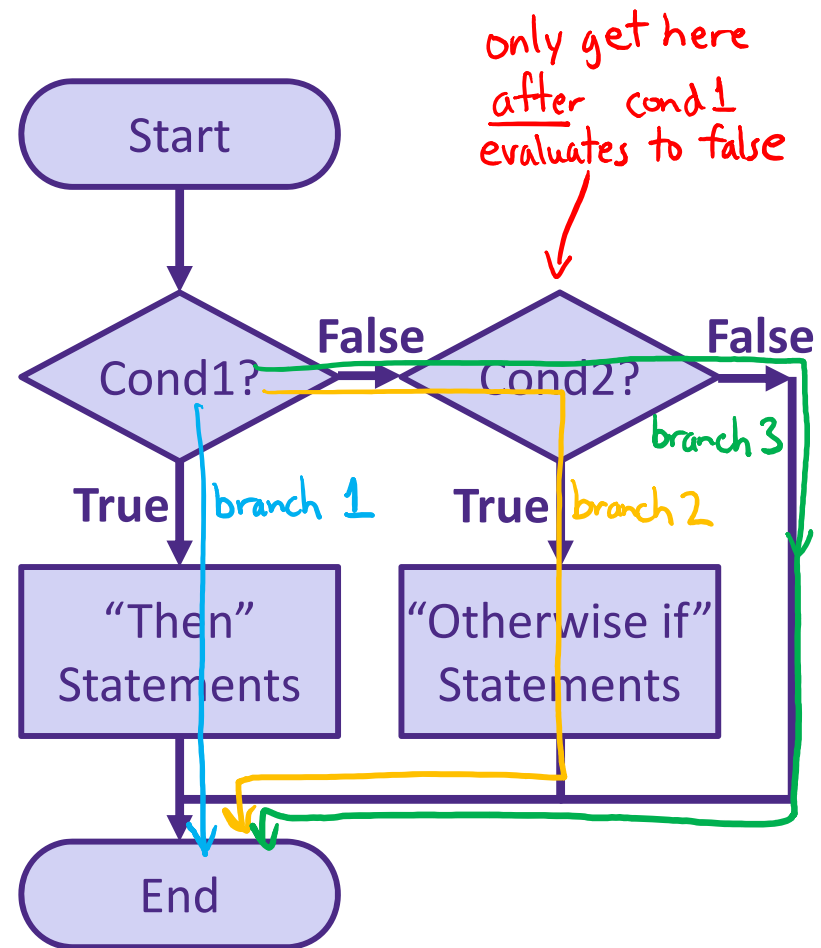
❖ With else if clause:

```

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

Handwritten annotations:

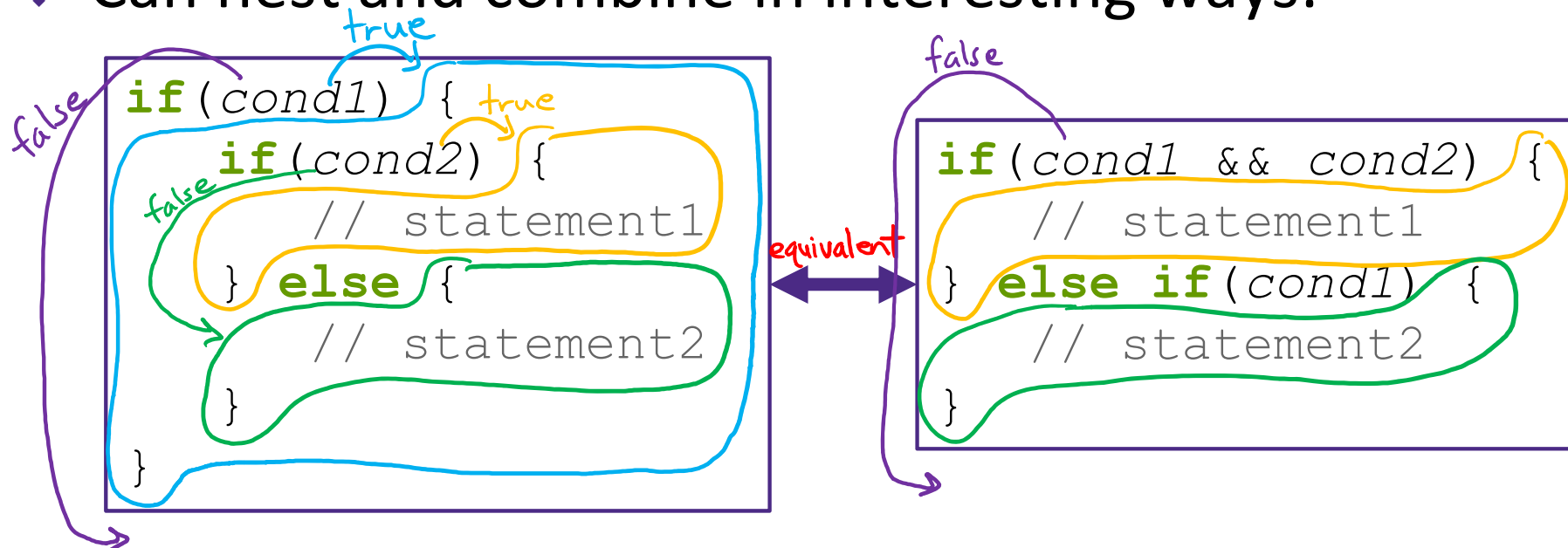
- Blue arrows: `if (cond1)` is true, leading to "then" statements.
- Orange arrows: `if (cond1)` is false, leading to `else if (cond2)`.
- Green arrows: `else if (cond2)` is true, leading to "otherwise if" statements.
- Green arrows: `else if (cond2)` is false, leading to the end.



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:



Practice 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");  
}
```

Handwritten annotations on the code:

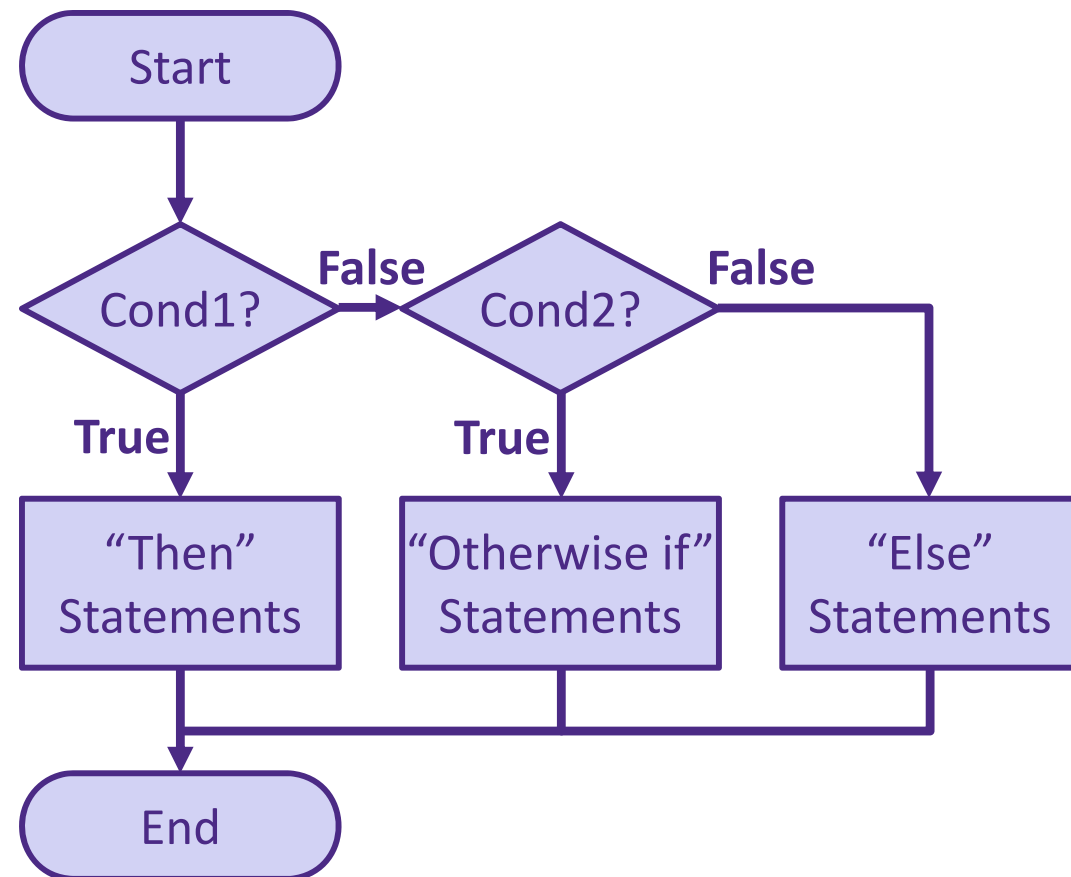
- Red arrow pointing to `==`: equal to
- Red arrow pointing to `>=`: greater than or equal to
- Red arrow pointing to `<`: less than
- Red arrow pointing to `||`: OR
- Color coding: `F` (False) in yellow, `T` (True) in blue, `F` (False) in green.

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

Conditionals Worksheet

❖ Work on Page 2 (Questions 7-9)

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



Processing Demo: Drawing Dots

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

```
14 void draw() {  
15   if(mousePressed) {  
16     fill(0, 0, 255); // blue if mouse is pressed  
17   } else {  
18     fill(255, 0, 0); // red otherwise  
19   }  
20   ellipse(mouseX, mouseY, 5, 5); // draw circle  
21 }
```



Jumping Monster

- ❖ Using *expressions* and *conditionals* in conjunction with *variables* and *user input* (Wed) to control what is drawn as well as motion:

