Syntax – `if` Statement

- `if` statements allow for code to be executed only under certain conditions

```
if ( boolean condition ) {
  statements          // do these only if the condition is true

} else if ( boolean condition ) {
  statements          // do these only if all previous conditions
                      // are false and this condition is true

} else {
  statements          // do these only if all of the conditions
                      // are false
}
```

boolean condition is an expression that evaluates to true or false – typically a comparison involving ==, !=, <, <=, >, >= or ! or two or more conditions joined by && or ||

statements are any legal Processing statements e.g. drawing commands, function calls, animation variable updates, conditionals, …

  - there can be any number of `else ifs` (including none)
  - the final `else` is optional

Structural Patterns

- `if` – “to do or not to do” (do something or do nothing)

```
if ( boolean condition ) {
  statements          // do these only if the condition is true
}
```

- `if-else` – do one of two alternatives (do nothing is not an option)

```
if ( boolean condition ) {
  statements          // do these only if the condition is true
} else {
  statements          // do these only if the condition is false
}
```

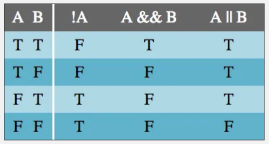
- `else if` – choose from more than two alternatives
  - any number of `else ifs` can be included

```
if ( boolean condition ) {
  statements          // do these only if the condition is true
} else if ( boolean condition ) {
  statements          // do these only if all previous conditions
                      // are false and this condition is true
} else {
  statements          // do these only if all of the conditions
                      // are false
}
```

include the final `else` only if do nothing is not an option

Logical Operators

- &&, `||`, and `!` are *logical operators*
  - they work with boolean values
  - && – and
    - both operands must be true for the whole thing to be true
  - `||` – or
    - one operand being true is enough; both operands must be false for the whole thing to be false
  - `!` – not
    - takes the opposite – if A is true, !A is false and vice versa



A	B	!A	A && B	A || B
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

https://ntgard.medium.com/https-medium-com-ntgard-boolean-logic-in-javascript-part-1-3371af974f19

On-the-Spot Decisions

decision about which alternative is based only on the current value of animation or system variable(s)

- How many alternatives are there?
- Is “do nothing” an option?

→ `if` structure

  - “to do or not to do” (do something or do nothing) → `if`
  - two alternatives → `if else`
  - more than two alternatives
    - “do nothing is not an option” → `if` / `else if` / … / `else`
    - “do nothing” is an option → `if` / `else if` / …

- What are the alternatives? → body of each part

- When does each alternative occur?
  → condition for each part

On-the-Spot Decisions

decision about which alternative is based only on the current value of animation or system variable(s)

- What are the alternatives?
  - blue, black circles: bounce or don’t bounce

    to bounce, reverse direction (flip the sign of the speed)

    as a rough fix for discrete time intervals, move the circle back up to the edge of the window if it has gone beyond (a simple but not perfect solution)
- How many alternatives are there?
  - “to do or not to do”
- Is “do nothing” an option?
  - yes (don’t bounce is “do nothing”)

  → if statement (no else)
- When does each alternative occur?
  - bounce when the circle has reached (or passed) the bottom of the window

(only one circle’s bouncing is shown)

```
// update position

// update speed

// bounce
if ( y2+20/2 >= height ) {
  y2 = height-20/2;
  vy2 = -vy2;
}
```

Prior Happenings Decisions

decision about which alternative depends on prior events / what has been going on

Implementation –

- need a *state variable* to keep track of the current alternative
  - associate each alternative with a different value
- need two `if` statements
  - one to use the state variable to determine the current behavior
  - one to update the state variable when the behavior changes

Prior Happenings Decisions

decision about which alternative depends on prior events / what has been going on

- What is the decision about? → state variable name
- How many alternatives are there? → state variable type
- Is “do nothing” an option? → `if` structure
- Which alternative do we start with? → state variable initialization
- What are the alternatives? → body of each part (usage `if`)
- For each alternative, when do we change to that alternative? → condition for each part (update `if`)

Prior Happenings Decisions

decision about which alternative depends on prior events / what has been going on

- What are the alternatives?
  - red circle: move up or move down
- What is the decision about?
  - direction of movement
- How many alternatives are there?
  - two
- Is “do nothing” an option?
  - no (red circle always moves)
- Which alternative do we start with?
  - move down
- For each alternative, when do we change to that alternative?
  - move down → reach bottom of window → move up → reach top of window → move down

```
float y1;       // center of red ball
boolean up;     // true if the ball is moving upwards

void setup () {
  size(700, 400);

  y1 = 0;
  up = false;
}
```

state variable

```
void draw () {
  ellipseMode(CENTER);
  background(255);

  // draw ball
  fill(255, 0, 0);
  ellipse(width/4, y1, 20, 20);

  // update position of ball
  if ( up ) {
    y1 = y1-2.5;
  } else {
    y1 = y1+2.5;
  }

  // change directions
  if ( !up && y1+20/2 >= height ) {
    up = true;
  } else if ( up && y1-20/2 <= 0 ) {
    up = false;
  }
}
```

usage if

update if