Impact Lab 2023
Lecture 2: Variables, Conditionals, and Loops, Oh My!

Summer 2023

School of Computing and Data Science

Wentworth Institute of Technology

p5.js

Wentworth
Computing & Data Science

Topics for Today

- **Variables**
- Conditionals
- Bouncing Ball
- `while` Loops
- `for` Loops
- Functions(?)

Wentworth
Computing & Data Science

Variables: Fundamental Concepts

Each variable can hold only one value (kinda...we do have arrays and objects too).

Over time, as the program executes, **the value of the variable can change**.

So, the variable stores a value so that we can use it at any time throughout the program.

This is how programming variables are different from their algebraic ones, which generally only represent one unknown value.

Wentworth
Computing & Data Science

Variables: Names

Programming languages have a few rules for variable names. For JavaScript:

Must start with a letter (upper or lower case) or an underscore.

May only contain letters, numbers and underscores.

Names are case sensitive

Good Examples:

```
count
x
user_input2
hit_points
```

Invalid Examples:

```
42
#yolo
a-b
```

Wentworth
Computing & Data Science

Variables: Declarations

Every variable must be declared before you can use it.

```
let value = 0;
```

You may also see:

```
var otherValue = 0;
```

The difference between **`var`** and **`let`** has to do with scope.

For now, I'll use **`let`** whenever we declare a variable.

Style Note:

I'll use camelCase in the code I write. The uppercase helps read the variable name.

Wentworth
Computing & Data Science

Variables: Initialization

Before you can use a variable, you **must** (should) give it a value!

This is not strictly true, some languages will just use a random value or zero (which may not good for your program).

Generally, JavaScript will just give the variable a value of **`undefined`**, which won't stop your program from running, but be very careful when using other languages.

Wentworth
Computing & Data Science

Printing Variables

Printing variables to the console is just as easy as printing **hello world!** (from last time)

Remember, **`setup()`** runs only once, and **`draw()`** runs every "frame", so for now, we'll be printing in **`setup()`**.

```
function setup() {
  let value=10;
  print(value);

  print("my value: "+value);

  print(`my value is: ${value}`);
}
```

There are many ways to create and print variables.

Wentworth
Computing & Data Science

Mathematical Operators

Mathematical Operators are (generally) used with numeric types.

Addition (+):	`total = part1 + part2;`
Subtraction (-):	`left_over = total - used;`
Multiplication (*):	`force = mass * acceleration;`
Division (/):	`item_wt = total / num_items;`

Wentworth
Computing & Data Science

Predefined Variables

p5.js has many predefined variables that store specific values related to p5.js functionality.

```
function draw() {
  background(0);
  circle(mouseX,200,50);
}
```

mouseX and **mouseY** constantly update to store the pixel position of the mouse on the canvas.

What happens in this **draw()**?

Wentworth
Computing & Data Science

Drawing Program

Let's move background(0) to setup.

What happens now and why?

Use **mouseX** and **mouseY** as the position of the circle.

```
Function setup(){
  background(0);
}

function draw() {
  circle(mouseX,mouseY,50);
}

function mousePressed(){

}
```

p5.js has other special functions like setup and draw. Lets add **mousePressed()** to our program.

Wentworth
Computing & Data Science

Topics for Today

- Variables
- **Conditionals**
- Bouncing Ball
- **while** Loops
- **for** Loops
- Functions(?)

Wentworth
Computing & Data Science

Control Flow

Control flow refers to the order in which your program statements are executed.

So far, all of our programs have been executed straight-through from the first to the last.

In general, you will need more complex control flow. For example, you may need to choose between two or more possibilities.

Wentworth
Computing & Data Science

Control Flow

Thought process:

Which greeting to give?

If I am looking at a friend:

Say “Yo”

If I am looking at an enemy:

Glare

JavaScript uses **if-else** statements to choose between alternatives

Wentworth
Computing & Data Science

Control Flow

Example:

```
if(id == friend){
  print(“Yo”);
}
else {
  print(“*Glare*”);
}
```

Generic form:

```
if(BOOLEAN EXPRESSION){
  YES/TRUE STATEMENT(S) ABOVE
}
else {
  NO/FALSE STATEMENT(S) ABOVE
}
```

Wentworth
Computing & Data Science

Control Flow

Each **`if-else`** statement allows your program to do one of two different things

In other words, you EITHER use one set of statements or the other, never both

The statements in-between the **`{ }`** for the **`if`** and the **`else`** can contain any code you want: printing things, using the math library, drawing things, even more **`if-else`** statements!

Wentworth
Computing & Data Science

Boolean Expressions

Boolean expressions (like **boolean** variables) are either true or false, and are composed of comparisons

Comparison Operators

`==`	Equal To	`if(x == 10)`
`!=`	Not Equal To	`if(x != 5)`
`<`	Less Than	`if(hours < 40)`
`<=`	Less Than or Equal To	`if(dollars <= 100)`
`>`	Greater Than	`if(dollars > 100)`
`>=`	Greater Than or Equal To	`if(hours >= 40)`

Complex **`Boolean`** Expressions

Multiple comparisons can be combined in the same expression using the logical “and” and the logical “or” operators.

`&&` is the logical “and” operator

`(hours > 20) && (hours <= 40)`

The entire expression is true **if and only if** both comparisons are true, otherwise it is false

`||` is the logical “or” operator

`(x < 17) || (y < 22)`

The entire expression is true if **either** comparison is true and only false when **both** are false

Exercise: Odd/Even

Write an if-else statement that checks a number and determines whether it is even or odd.

Even numbers are divisible by 2

Odd numbers are not divisible by 2

Wentworth
Computing & Data Science

Multiple Choice

What if you have/want more than two options?

The standard **`if-else`** only gives you two choices

That’s where the **`else if`** comes in. You can add **`else if`** statements between the **`if`** and **`else`** (any number of them!)

The key is that each will be tested in order until one of the conditions is true. If none are true, the final **`else`** will execute.

You can actually completely omit the **`else`** if you want

Wentworth
Computing & Data Science

Multiple Choice

```
if(EXPRESSION1){
  RUN THIS STATEMENT IF EXPRESSION1 IS TRUE
}
else if(EXPRESSION2){
  RUN THIS STATEMENT IF EXPRESSION2 IS TRUE
}
else {
  RUN THIS STATEMENT IF BOTH EXPRESSIONS ARE FALSE
}
```

Wentworth
Computing & Data Science

Exercise: Grade

Write a set of **`if-else`** statements that take a numeric score and outputs a letter grade:

if the score is $>=90$, output A

if the score is $>=80$ and <90, output B

if the score is $>=70$ and <80, output C

if the score is $>=60$ and <70, output D

if the score is <60, output F

Wentworth
Computing & Data Science

`if-else` Summary

Use **`if-else`** statements when you need to choose between two or more possibilities.

Always put boolean expression in parentheses

Multiple expressions can be put together with the logical "and" (**`&&`**) and the logical "or" (**`||`**) operators.

Always put each comparison in its own set of parentheses

Don't forget to use "==" for equality, not "="

Wentworth
Computing & Data Science

Topics for Today

- Variables
- Conditionals
- **Bouncing Ball**
- **`while`** Loops
- **`for`** Loops
- Functions(?)

Wentworth
Computing & Data Science

Bouncing Ball

We're going to create a circle that bounces around the canvas region.

This in-class project will allow us to talk about a few topics:

- **Physics**, the ball needs to move
- Boundary Checks
- Graphics

Wentworth
Computing & Data Science

Physics Lesson

$$\frac{dx}{dt} = v$$

A change in position over time is velocity

$$\frac{dv}{dt} = a$$

A change in velocity over time is acceleration

Our ball won't do any acceleration, we'll only work about a 'kick' to change it's direction

Wentworth
Computing & Data Science

Physics Lesson

Using the definition of the derivative:

$$\frac{df(t)}{dt} = \frac{f(t+dt) - f(t)}{dt}$$

Function at the next time

Function at the current time

So,

$$\frac{dx}{dt} = v$$

$$\frac{x(t+dt) - x(t)}{dt} = v$$

$$x(t+dt) = v \cdot dt + x(t)$$

What is *dt*?

Wentworth
Computing & Data Science

Physics Lesson

What is *dt*?

Change in time between frames

$\mathcal{O}(16ms)$

Change in Position

$$x(t+dt) = v \cdot dt + x(t)$$

Change in Velocity

$$v(t+dt) = a \cdot dt + v(t)$$

Ultimately, all we need to know is the previous position/velocity!

We'll supply a value of **dt** to our move method. Ideally, it should be related to our animation update time.

We'll also adjust the velocity values so the ball movement looks good.

cience

Setup

```
let frameTime=60;
let dt=1/frameTime;
let xPosition=100;
let yPosition=100;
let xVelocity=0;
let yVelocity=0;

function setup() {
  frameRate(frameTime);
  createCanvas(400, 400);
  yVelocity=100;
  xVelocity=58;
}

function draw() {
  background(220,0,200);

   //bouncy code here

  fill(0);
  circle(x,y,10);
}
```

We'll start with some initial code that gets some variables ready.

Declaring variables outside of setup/draw allows us to use the variables anywhere in the code.

Wentworth
Computing & Data Science

Topics for Today

- Variables
- Conditionals
- Bouncing Ball
- **`while`** Loops
- **`for`** Loops
- Functions(?)

Wentworth
Computing & Data Science

Loops

Often, you need to repeat the same computation, action, or sequence of steps many times.

I WILL NOT EXPOSE THE IGNORANCE OF THE FACULTY.
I WILL NOT EXPOSE THE IGNORANCE OF THE FACULTY.
I WILL NOT EXPOSE T

Like writing "I will not expose the ignorance of the faculty." 100 times

Wentworth
Computing & Data Science

Loops

Of course, you could use 100 **`print()`** statements to do this, but that's a lot of copy and paste work.

Instead, programming languages have control flow mechanisms called loops that allow you to loop over (repeat) the same section of code as many times as you need.

Two of the most common types of loops are **`while`** loops and **`for`** loops

Wentworth
Computing & Data Science

`while` Loops

The **`while`** loop is used to repeat a set of statements **`while`** some condition is **`true`**.

```
let iteration = 1;
while (iteration <= 100) {
  print("Repeat this line\n");
  iteration = iteration + 1;
}
```

Notice how there is a boolean expression after the **`while`**. Also, Within the loop the iteration variable is incremented.

Wentworth
Computing & Data Science

while Loops

```
while (BOOLEAN EXPRESSION) {
  STATEMENT1;
  STATEMENT2;
  ...
}
```

The loop body is between the { and }

The loop body executes over and over as long as the expression is true.

These boolean expressions are the same as the **if/else if** statements

Each repetition of the loop is called an **iteration**.

Wentworth Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

Current Value of i: 7

Wentworth Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
```

Current Value of i: 7

Wentworth Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
```

Current Value of i: 3.5

Wentworth Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
```

Current Value of i: 3.5

Wentworth
Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
3.5
```

Current Value of i: 3.5

Wentworth
Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
3.5
```

Current Value of i: 1.75

Wentworth
Computing & Data Science

while Loop Behavior

```
let i=7;

while (i > 1) {
  print(i);
  i = i / 2;
}

print("Done.");
```

Console Output

```
7
3.5
```

Current Value of i: 1.75

Wentworth
Computing & Data Science

`while` Loop Behavior

```
  let i=7;

  while (i > 1) {
➡   print(i);
    i = i / 2;
  }

  print("Done.");
```

Console Output

```
7
3.5
1.75
```

Current Value of `i`: `1.75`

Wentworth
Computing & Data Science

`while` Loop Behavior

```
  let i=7;

  while (i > 1) {
    print(i);
➡   i = i / 2;
  }

  print("Done.");
```

Console Output

```
7
3.5
1.75
```

Current Value of `i`: `0.875`

Wentworth
Computing & Data Science

`while` Loop Behavior

```
  let i=7;

  while (i > 1) {
    print(i);
    i = i / 2;
➡ }

  print("Done.");
```

Console Output

```
7
3.5
1.75
```

Current Value of `i`: `0.875`

Wentworth
Computing & Data Science

`while` Loop Behavior

```
  let i=7;

  while (i > 1) {
    print(i);
    i = i / 2;
  }

➡ print("Done.");
```

Console Output

```
7
3.5
1.75
Done.
```

Current Value of `i`: `0.875`

Wentworth
Computing & Data Science

Topics for Today

- Variables
- Conditionals
- Bouncing Ball
- **`while`** Loops
- **`for`** Loops
- Functions(?)

`while` Loops...Again...

`while` loops are often used to repeat a task a fixed number of times, which often leads to similar structure in the body of the loop

Counter variable (and initialization)

Counter variable in the boolean expression

```
let count;
count = 1;
while (count <= 8){
   print(count+” squared is “+count*count);
   count++;
}
```

Increment the counter variable

`for` Loops

`for` loops are specialized loops that have these features built in (and you'll probably use them for most loops that you write)

Counter variable (an initialization)

Counter variable boolean expression

```
let count;
for (count = 1; count <= 8; count++) {
   print(count+” squared is “+count*count);
}
```

Increment the counter variable

Generic Form

```
for (INITIALIZATION; BOOLEAN_EXPRESSION; UPDATE) {
   STATEMENT1;
   STATEMENT2;
   …
}
```

Again, the loop body is between the **`{}`**

`INITIALIZATION` is done one time before the first loop iteration.

`UPDATE` is done every loop iteration after the end of the loop body.

`BOOLEAN_EXPRESSION` is checked every loop iteration after **`UPDATE`** (and once after **`INITIALIZATION`**)

Example: Summing Values

Say we have a math equation:

$$\sum_{i=0}^{10} i$$

```
let sum=0;
for (let i = 0; i <= 10; i++) {
   sum+=i;
}
print(sum);
```

Wentworth
Computing & Data Science

Gotchas

There are only two semicolons!

Between the initialization and the boolean

Between the boolean and the update

No semicolon after the update

No semicolon after the parentheses

If you are doing and increment, be sure you use **`i++`** or **`i=i+1`**, not just **`i+1`** (it doesn't do anything)

Wentworth
Computing & Data Science

`for` and `while`

Both loops work basically the same way

The main difference is the **initialization** and **update** process that is defined within the syntax of the **for** loop.

Which one is better?

Wentworth
Computing & Data Science

Topics for Today

- Variables
- Conditionals
- Bouncing Ball
- **`while`** Loops
- **`for`** Loops
- **Functions(?)**

Wentworth
Computing & Data Science

Predefined Functions

JavaScript and p5.js includes many predefined functions for common tasks

We’ve already been using functio

Function call

```
Function setup(){
  createCanvas(400,400);
}
```

Arguments

If the function has a return value, we “capture” it using an =

Terminology Notes

We use **parameters** to refer to the list of variables that a method requires (in parentheses)

They are place holders for the values that will be used when the method is called (or dummy variables)

We use **arguments** to refer to the specific values and/or variables that are passed to the method when you invoke the method

Also, other languages refer to methods as **functions**, **procedures**, **methods**, or **subroutines**

Example

```
function setup(){
  let number=15;
  let cube=0;
  let log2=0;

  cube=Math.pow(number,3);
  print(number+” cubed = “+cube);

  print(“square root of “+number+” = “+Math.sqrt(number));

  log2=Math.log2(number);
  print(“log2 of “+number+” = “+log2);
}
```

Pogrammer Defined Functions

JavaScript allows you to define your own functions to meet the needs of your specific program

All of the functions that you create will have a **signature** and a **body** that you define

The signature includes the method name and parameter list.

The body is the set of JavaScript statements that will be executed when the method is invoked

Generic Form

```
function METHOD(PARAMETER_1, PARAMETER_2, …, PARAMETER_N)
```

A method can have any number of parameters, even zero.

A method has either zero or one return value(s)

- The return value is commonly the result of a function.
- The return value can be assigned to a variable when calling the function.
- Alternatively, the method call can be placed directly in another JavaScript expression.