A beginner's introduction to JavaScript and p5.js

Europe Connexion's team*

February 20, 2026

Figure 1: p5.js logo

Abstract

This document is meant to help beginners and intermediates learn how to write code in JavaScript using the p5 library. The p5 library is a library optimized for beginners, that makes it easy and intuitive to draw and create simple programs. p5 uses JavaScript and is therefore a great way to learn the language and builds a strong foundation for learning other languages in the future. By the end of this document the reader should have a solid grasp on the syntax of the language, as well as how to operate within it using the p5.js library.

*Made in collaboration with Academia Main().

Contents

1 Introduction to p5.js

1.1 What is p5.js?

P5.js is a library for JavaScript. It is optimized for learning to code, and drawing easily. It is a ideal way to start learning the basics of coding. You learn a lot about JavaScript, and build a great foundation for learning other programming languages in the future.

Use https://p5js.org/reference/ to look up functions, and see all the build in p5 functions.

1.2 The foundation of p5.js

P5.js works using two primary functions:

1. function setup()
2. function draw()

The setup function is used for initializing the document. This typically involves creating a canvas. The canvas will be the foundation for your code, here you can draw a lot of different things that's being controlled by the program. The canvas works like a coordinate system, with an x- and y-axis. The origin is in the top left corner.

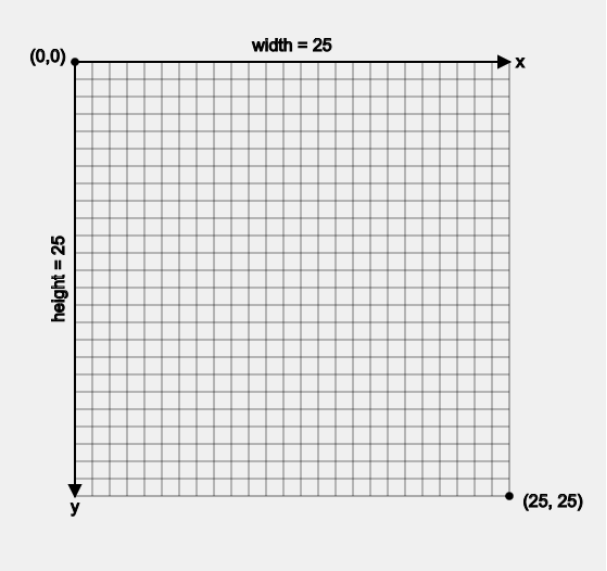

Figure 2: canvas in p5.js with coordinate system

The draw function will run each frame. The draw function is necessary to, for example, code animations. The draw function could for example code a circle bouncing between walls of the canvas.

1.3 Getting started

1.3.1 Visual Studio Code

When coding, we recommend using Visual Studio Code (VS Code). VS Code is a free, open-source code editor that is very effecient, and has a lot of features.

You can download VS Code at: https://code.visualstudio.com/download

Understanding The VSCode Interface

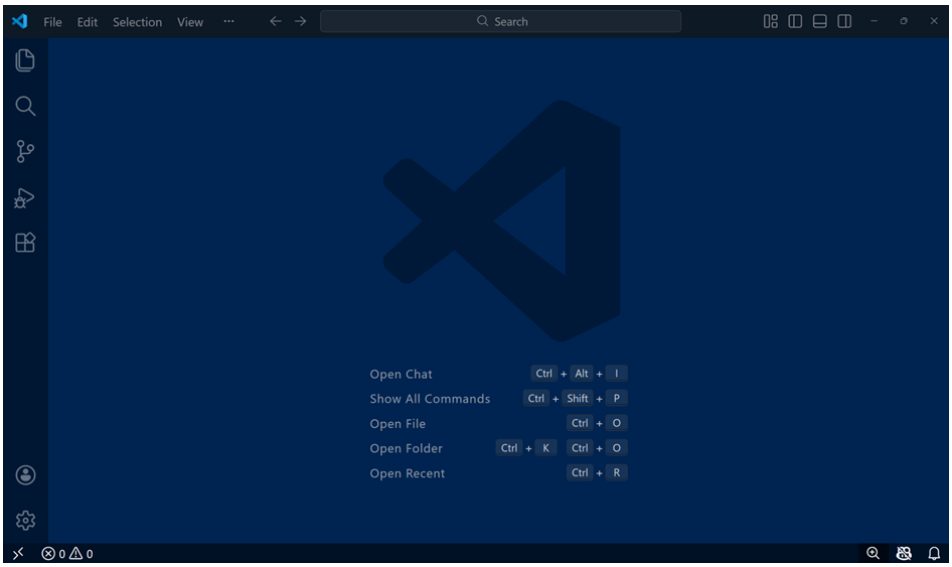

Figure 3: VSCode interface

The layout is very simple and modular. On the left-hand side you will notice a few icons. These will be explained below in further detail.

- Starting from the top, there is the paper-icon, called the Explorer. This tab allows you to view and manage your project files and folders.
- The four next icons are respectively Find & Replace, Source control (more on that later), Run & debug and the extension tab.
- At the bottom, you will find the Accounts and settings tab.

Essential Extensions

VSCode supports different extensions, that in many ways can improve your workflow. Below are three extensions we highly recommend you get:

P5 Project Creator ***(by Ultamatum)*** This extension makes it easier to create default p5.js projects in VS Code by generating a ready-to-use project structure with all the necessary files, which saves time compared to doing it manually.

p5js Snippets ***(by Acidic)*** This extension provides snippets for all functions in p5.js, essentially shortcuts, which elevate the coding experience by speeding up the process of coding whilst also reducing typing errors.

Live Server ***(by Ritwick Dey)*** This extension simply allows you to launch a local development server in your browser, which automatically refreshes when you save changes in the code editor. It's a very effecient and easy way to run your code locally.

To install an extension, first go the extensions tab, and search for the extension. Find and select the correct extension (check the author), then click install.

1.3.2 Creating A New Project

If you installed P5 Project creator before, creating a new project is very simple:

1. Create a project folder on your computer
2. Open VSCode and Press *File* in the top left corner, and select "Open Folder"
3. Find and select your project folder
4. Press view in the top bar of VSCode, and then press command palette
5. Write "Create P5 Project" and press enter.
6. Done!

The extension will create two files:

- Index.html
- Script.js

The file names (except file extension, of course!) can be changed, but these are the standard names. You will need both files to run your project. You will be writing your code inside the JavaScript file. The HTML file will only be used to load everything from the JavaScript file.

If you want to setup the files manually, still create a project folder. Then create the two files (Index.html and script.js) manually by right clicking inside your folder on VSCode and then select the option "*New File*". When you created both files paste the following code into your HTML and JavaScript file respectively.

HTML Template

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Exercise 1</title>

    <script src="https://cdnjs.cloudflare.com/
        ajax/libs/p5.js/1.4.0/p5.js"></script>
</head>

<body>
    <script src="sketch.js"></script>
</body>

</html>
```

JavaScript Template

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

function draw() {

}
```

1.3.3 Running the code

Now that the necessary files are set up, you can try running the file. This is done by either right-clicking on the index.html file in the Explorer tab and selecting the option "Open with Live Server", or simply by opening the index.html file and clicking the "Go Live"-button in the bottom right corner. This should open your browser, where you will be able to view your program.
Try adding the function *background(220);* inside the draw function. When you open the browser with your project you should see a lightgrey box. This is your canvas! Remember to save your project to see it.

2 Variables

2.1 What Is a Variable?

A variable acts as a container, that saves a value in the memory. You can think of it as a box with a name written on it. When we use the name, we get access to the content.

In JavaScript, creating a variable is typically done using "let".

```
let x = 100;
```

When creating this variable, three things happen:

1. We create a variable ***(let)***
2. We give it a name ***(x)***
3. We give it a value ***(100)***

It's important to note that choosing meaningful names for your variables can help a lot in your project. E.g.

```
let playerX = 100;
let speed = 5;
let score = 0;
```

It makes your code easier to read and understand.

2.2 Why Are Variables Important?

In p5.js your canvas is updated several times per second in the draw() function (around 60 times, depending on the framerate). If we want something to move or change, we must save values in variables to change them continuously.

```
let x = 50;

function draw() {
    x = x + 1;
}
```

In this example, we are constantly incrementing the variable x's value by 1. If a circle was drawn at the x value, we can imagine that the circle will move 1 pixel to the right each frame.

2.3 Datatypes in p5.js

Variables can contain a lot of different values. These values could be text, numbers and so on. The type of data you assign to the variables, are called datatypes.

Here is a list of possible data types, that you can assign to a variable:

Datatype	**Explanation**	**Example**
Integer *(number)*	An integer is a whole number.	let age = **18**;
Float *(number)*	A float is a decimal number.	let height = **180.5**;
String	A string is text. To make a string the value should be surrounded by quotation marks.	let name = **"erasmus"**;
Boolean	A Boolean is a binary value; it can either be True or False.	let hasGirlfriend = **false**; let working = **true**;

Note that JavaScript does not distinguish between Integers and floating-point numbers. Both types are classified as *number*.

Different datatypes can be added together. This will often affect the output data type. Try the following small exercise. Dont worry if you get it wrong, it can be a bit confusing.

Exercise 2.1: Guess the output datatype

a) let output = age + height;

b) let output = height + name;

c) let output = name + hasGirlfriend;

2.4 Arithmetic Operations in p5.js

Arithmetic operations are mathematical operators, and are used to manipulate numbers.

Operator	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

The operators can be used on their own, and will evaluate two elements. E.g. two variables.

```
let a = 8;
let b = 3;

let sum = a + b;
let difference = a - b;
let product = a * b;
let quotient = a / b;
let rest = a % b;
```

Using the operators on their own will not affect the variables value. If you add an equal sign *(=)* after any of the operators, the variable value would be updated E.g.

```
let score = 0;

score += 10; // adds 10 to the variable
                    value
```

Comments in JavaScript are written as "//"

In this code *score* changes from 0 to 10.

For incrementing a variable value with 1, use ++

For decrementing a variable value with 1, use - - E.g.

```
let score = 5;

score++; // score increases to 6
score--; // score decreases back to 5;
```

2.4.1 What Is Modulus *(%)*?

Modulus *(%)* returns the remainder when dividing one number by another. For example when dividing 10 by 3, you will get a remainder of 1.

Modulus can for example be used to check if a number is even or uneven:

```
if (number % 2 === 0) {
    console.log("Number is even");
}
```

This code prints "Number is even" to the console if a number defined by the *number* variable is even. Any number that is even will when divided by two, leave a rest of zero. Therefore, a number must be even, if it's rest of division by 2, is equal to zero.

2.5 *Coding practice: Variables*

Practice 2.1 Assign a variable "x" to the value 10 and print its value to the console.

Practice 2.2 Create two variables a = 8 and b = 2. Then print the sum, difference, product, quotient and the remainder of division (using modulus: %) of the two numbers.

Practice 2.3 Draw a circle in the middle of the canvas with a diameter defined by a variable d(choose any value, for example 100). Use the text() function with the variable, to write the diameter inside the circle.

Updating d, should update the diameter and the text displayed.

Practice 2.4 Make a mouse-tracker that displays relevant information about your mouse on canvas. Use mouseX and mouseY to write down the mouses current position, pmouseX and pmouseY to write down the mouses previous position. Then create two new variables, velocityX and velocityY that describes the mouses speed in x and y.

3 Conditional Execution

In this chapter we will learn how to make decisions in our programs using conditional execution. Up till this point, our code has run line by line, the same way every time. With conditional execution, we can make the program react differently depending on conditions.

3.1 What Is Conditional Execution?

Conditional execution means:

- Run this code ONLY if something is true

In JavaScript we use the "if-statement"

```
if(condition) {
    // code runs if condition is true
}
```

The condition statement must be true for the execution to take place.

```
let age = 20;

if (age >= 18) {
console.log("You are an adult");
}
```

Here we can see that the condition 20>=18 is true, and therefore "You are an adult" will be printed to the console. The operator ">=" is called a Comparison operator, that means greater than or equal to. If age was less than 18, nothing would be printed, because the condition was not met.

3.2 Comparison Operators

Comparison operators are used for comparisons. They always return a Boolean value (true/false), based on the comparison.

Symbol	**Operation**	**Explanation**
==	Equal	Checks if two values are equal
===	Strict equal	Checks if two values and their types are equal
!=	Not equal	Checks if two values are equal, returns true if they are not
!==	Strict not equal	Checks if two values or their datatypes are equal, returns true if any of these are not
>	Greater than	Checks if a value is greater than another
<	Lesser than	Checks if a value is lesser than another
>=	Greater than or equal	Checks if a value is greater than or equal to another
<=	Lesser than or equal	Checks if a value is lesser than or equal to another

3.3 if...else Statements

When we check if a condition is true or false, it can be useful to have one thing happen if it's true, and another thing happen if it's false. This is done using if...else statements.

```
if(condition) {
    // runs if true
} else {
    // runs if false
}
```

Example of usage:

```
let temperature = 17;

if (temperature > 20) {
    console.log("It is hot");
} else {
    console.log("It is cold");
}
```

Only one of the "blocks" will run. In this case the else-statement will run, and print "It is cold", because the condition *17 > 20* will return false.

3.3.1 else if statement

The previous example with temperature, was pretty naive, since it only had two options. The example would say that the weather was cold even if it was 19 degrees, which does not make sense for a lot of people. We can work around this by adding more than two options. This is done using the *else if* statement.

```
if (temperature > 20) {
    console.log("It is hot");
} else if (temperature > 15) {
    console.log("It is moderate");
} else {
    console.log("It is cold");
}
```

In this example, the statement will find the first true condition, and run the respective code. In this example 17 degrees would return moderate, instead of cold.

3.4 Logical operators

Sometimes it can make sense to check multiple conditions. This is done using logical operators.

Symbol	**Operation**	**Explanation**
&&	AND	Checks if two conditions are true. Returns true if both are.
||	OR	Checks the state of two conditions. Returns true if one of the conditions are true.

Usage examples:

```
// AND ( && ) usage:
if (temperature >= 22 && wind < 10) {
    console.log("It's beach weather!");
}

// OR ( || ) usage:
if (raining === true || temperature < 18) {
    console.log("Remember a jacket")
}
```

3.5 *Coding practice: Conditional Execution*

Practice 3.1

1. Code a small "drawing program" where a circle is drawn constantly at the cursor's location.
2. Use the built in mouseIsPressed variable to print to the console if the mouse button is held down.
3. Upgrade the program so the circles diameter increases when held down.

Practice 3.2 Figure out how key inputs work in p5.js, using p5js.org. Then draw a circle in the middle of the screen. When you press one of the arrow keys, the circle should move in that direction.

Practice 3.3 Using a variable x, draw a circle that moves along the x-axis on canvas with a specific speed. When the circle hits a wall, it should change direction so it never leaves the canvas.

Hint: multiplying the speed with -1 will change the direction.

Practice 3.4 Make a canvas that can be divided into three equal vertical slices (for example 600x400). Make a program that highlights one third of the canvas at a time with a rectangle, depending on whether the mouse is on the left, middle or right third of the canvas.

4 Loops and Arrays

Sometimes it is useful to repeat similar code multiple times. For example drawing three circles with diameters 1, 2 and 3. We draw a circle three times, but with varying diameters. It can be done manually like this:

```
circle(x,y,1); // Circle with diameter = 1
circle(x,y,2); // Diameter = 2
circle(x,y,3); // Diameter = 3
```

The three lines of code are very similar and can be optimized using a loop. In JavaScript we talk about two different types of loops. For-loops and while loops. The difference is that in a for loop, we now the number of iterations.

Try to think of the example with the circles. Should we use a for or while loop to optimize the code?

In this example the most optimal loop would be a for loop. It would, since we have to draw a circle three times i.e. we have three iterations. Note that a while loop also could be used for this example theoretically.

4.1 For loops

A for loop is used to repeat a block of code, a specific number of times. The structure for a for loop looks like this:

```
for (let i = 0; i < 4; i++) {
    // code to repeat
}
```

In the code above, we have a for loop repeating four times. First a variable i is being declared, and assigned the value 0. Each time the code has run, the last argument in the for loop ($i++$) runs. So in this example i is being incremented by one, for each iteration of the code.

If we add console.log(i) inside the for loop, like this:

```
for (let i = 0; i < 4; i++) {
    console.log(i)
}
```

And then run the code, the following will be printed to the console:

```
0,
1,
2,
3
```

Try the following thinking exercise:

Exercise 4.1:

- Why does it print 0,1,2,3 and not 1,2,3,4?
- What would the loop print if we changed the third argument from *i++* to *i+=2*?
- What would the loop print if we changed the first argument from *let i = 0* to *let i = 2*

Now that we now a bit about for loops, we can automize the example with three circles from before, like this:

```
for (let d = 1; d <= 3; d++) {
    circle(x,y,d);
}
```

(x,y) could be any chosen coordinates.

In this for-loop we use a variable d, that corresponds to the circles diameter. The for loop has three iterations, that look like this:

d = 1: circle(x, y, 1);

d = 2: circle(x, y, 2);

d = 3: circle(x, y, 3);

As we can see, it produces the same output as the manual example.

4.2 While loops

A while loop will repeat a specific block of code a unspecified number of times. The structure of a while loop, could look like this:

```
while(condition) {
    // code to repeat
}
```

For loops can be used instead of while loops in most scenarios. The primary use of while loops are when you don't know the number of iterations. For example when generating a random number like this:

```
let i = 0;

while(i !== 9) {
    i = random(0,9);
}
```

This (completely useless) code, will first define the variable i and then continue asssigning it a random value between 0 and 9, until the value randomly becomes 9.

Note that this loop would never actually stop, since the random() function produces floating-point numbers, so it would for example return a number like 1,2312355... We're waiting for i to become the integer 9, which will never happen.

The code would produce a infinite loop, where the condition will stay true indefinitely, which would ultimately crash your program.

Try to draw the same three circles as drawn last, this time using a while loop.

Exercise 4.2:

- Create a while loop that draws three circles with the diameters 1,2 and 3.

4.3 Arrays

Sometimes it is useful to store multiple values to one variable. Think of a graph containing multiple grades from a class. Instead of creating a new variable for each grade, we can instead create one variable containing all grades.

```
let grades = [86, 92, 100, 54, 32, 54, 95,
                                    56, 76, ...];
```

This is called an array. It's defined using square brackets: [ ]. Each data point inside the brackets are separated by commas.

Each grade inside the grades array have an index. Note that the indexes start at 0 and not 1. So for example the grade 86 is at index 0, grade 92 is at index 1, grade 100 is at index 2, and so on. Getting one of the grades can be done using *grades[i]* where i is the index. For example *grades[5]* would return 54.

```
let grades = [86, 92, 100, 54, 32, 54, 95,
                56, 76, 77, 14, 33, 86, 85,
                96, 100, 60, 75];

let sum = 0;

for (let i = 0; i < grades.length; i++) {
    sum += grades[i];
}

console.log("The average grade is: ",
```

```
sum / grades.length)
```

Output: "The average grade is: 70".

The code runs through all the grades and adds them together. When we have the sum of all grades we can divide them by the number of observations, which is the length of the array grades, to get the average grade.

The code is responsive, so adding new grades to the array, will also update the output(average grade).

4.3.1 Modify arrays using simple array methods

We can manipulate an array while the code is running, using differnt methods:

Method	**Explanation**
Array.push(element)	Adds element to the end of an array
Array.pop()	Removes the last element of an array
Array.unshift(element)	Adds element to the front of an array
Array.shift()	Removes the first element of an array

4.4 *Coding practice: Loops and Arrays*

Practice 4.1 Draw 100 circles at random locations on canvas, using a for loop. Then do it with a while loop.

Practice 4.2 Use a for loop to draw all numbers between 0 and 100 on canvas on random locations. The number should be blue if it's even, and red if it's uneven.

Hint: use i%2

Practice 4.3 Define an array "spanishGreetings" and give it these values: "Hola", "Buenos dias", "Buenas tardes" and "Buenas noches". Now create a for loop that runs through all elements of the array and writes the greeting in a list downwards. Add the value "Mucho gusto" to the array, and check if it's written under the previous element.

Practice 4.4 Create a 8x8 chess board on a 800x800 canvas. (changing between black and white rectangles). Use a for loop nested in another for loop.

5 Functions

In this chapter we learn about functions - one of the most important concepts of programming.

Functions allow us to:

1. Organize code and structure large programs
2. Avoid repetition by creating reusable logic
3. Build more advanced programs

5.1 What is a function?

A function is a reusable block of code that performs a specific task.

A function can take inputs, and return outputs based on your inputs. The basic syntax for a function looks like this:

```
function functionName(inputs) {
    // code to run
}
```

An example of a simple function, that does the same thing everytime, ergo, does not take an input, may look like this:

```
function sayHello() {
    console.log("Hello!");
}
```

The function can be run (called) anywhere in the code, using: *sayHello()*

If you don't call the function, it won't run.

5.2 Funtion parameters

Functions can also be used as some kind of a "template" if you pass inputs(parameters) to it. Parameters work as a kind of variable used inside functions. You can pass them on to the function, where they can be used to produce unique outputs. This is a simple example, where we updated the *sayHello* function, so it can take a parameter; *word*:

```
function say(word) {
    console.log(word);
}
```

Now when calling the function *say*, we can pass on an arbitrary value to the function, that it will print. For example calling *say("Hello!")* would do the same as calling *sayHello()*. We could also pass on other words to the *say* function, for example *say("¡Buenos Dias!")*, which would print "¡Buenos Dias!". Note that the system does not know what the input *word* means.

word could also be replaced by another name like *greeting* or something abstract like x, without changing what the function does.

```
function greet(name) {
    console.log("Hello " + name);
}

greet("Anna"); // Prints: "Hello Anna"
greet("Jonas"); // Prints: "Hello Jonas"
```

Try this short exercise.

Exercise 5.1:

a) What is the parameter being passed on to the *greet* function?

b) Create the code for a function that takes the parameter *age*, and then prints "Your age is ..." to the console.

5.3 Function outputs

Other than being able to do small tasks like printing things to the console, functions can also be used to do more advanced operations. We can pass more parameters to the function, and return a output based on the inputs.

```
function sum(a, b) {
    return a + b;
}
```

Now when calling the function using two values, for example: *sum(3,5)*, it will return the sum of the inputs. So, passing 3 and 5, would return 8.

```
console.log(10+sum(3,5));

// Prints 18 to the console.
```

We can imagine how powerful the function could be if we use an array as a parameter. If we think of the grading example from section 4.3, we can make a function that calculates the average grade easily. We're gonna use a shortened version of the *grades* array for simplification.

```
let grades = [86, 92, 100, 54];

function calculateAverageGrade(gradeArray) {
    // We take the parameter gradeArray.
    // Note that this could be any arbitrary
       array with grades.

    let sum = 0;
```

```
    for(let i = 0; i < gradeArray.length; i++) {
        sum += gradeArray[i];
        // This for-loop calculates the sum of
           all elements in gradeArray
    }

    return sum / gradeArray.length;
}

let averageGrade = calculateAverageGrade(grades);

console.log("The classes average grade is: "
                + averageGrade);
```

Note that variables defined inside functions, e.g. *gradeArray*, are only accesible inside the function.

5.4 *Coding practice: Functions*

6 Mini projects

When you feel confident in your basic coding skills, improve them by using them in practice. We made a couple of small projects that should help you use different parts of programming to make larger and more diverse programs. The projects gradually increase in difficulty.

Project 1: Reaction Clicker Game

Create a simple reaction game, where a red circle appears at a random location on the canvas. When the player clicks the circle, they gain 1 point, kept in a "score" variable. The circle (after being clicked) moves to a new random position and it can then be clicked again. Display the "score" variable on screen.

Project 2: Falling Objects Dodging Game

Create a game where a player controls a blue circle at the bottom of the canvas using arrow keys. Red circles randomly fall from the top. If a red circle hits the player a "Game Over" screen appears. Keep track of the time survived (can be frames).

Project 3: Two-player Competitive Target Game

Make a simple two-player game, where two players compete to catch targets. One player (a white circle) can move around canvas with the WASD-keys,

the second player (a blue circle) can move around canvas with the arrow keys. The players shouldn't be able to go outside the canvas. At the start of each game, a small red circle (the target) should appear at a random position. If one of the players touches the target, it will disappear and a new one will appear at a new random position. The player that catch the target will get a point.

When you have the base game, add your own innovative feature. It could be moving targets, obstacles on the canvas or anything else!

7 Attachments

7.1 Summary list for exercises

Exercise 2.1:

a) number(float) | output = 198.5

b) string | output = “180.5erasmus”

c) string | output = “erasmusfalse”

Exercise 4.1:

1. Possible answer: the variable i starts at 0. The value will be incremented by 1 as long as i is lesser than 4. So in the first iteration we have i = 0. The code runs and then $i{+}{+}$ is run. i is now equal to one, and the same continues all the way up to three, which is the last integer that's lesser than 4.

2. 0, 2

3. 2, 3

Exercise 4.2:

```
let d = 1;

while(d <= 3) {
    circle(x,y,d);
}
```

Exercise 5.1:

a) *name*

b) Possible answer:

```
function printAge(age) {
    console.log("Your age is " + age);
}

printAge(21); // Prints: "Your age is 21"
```