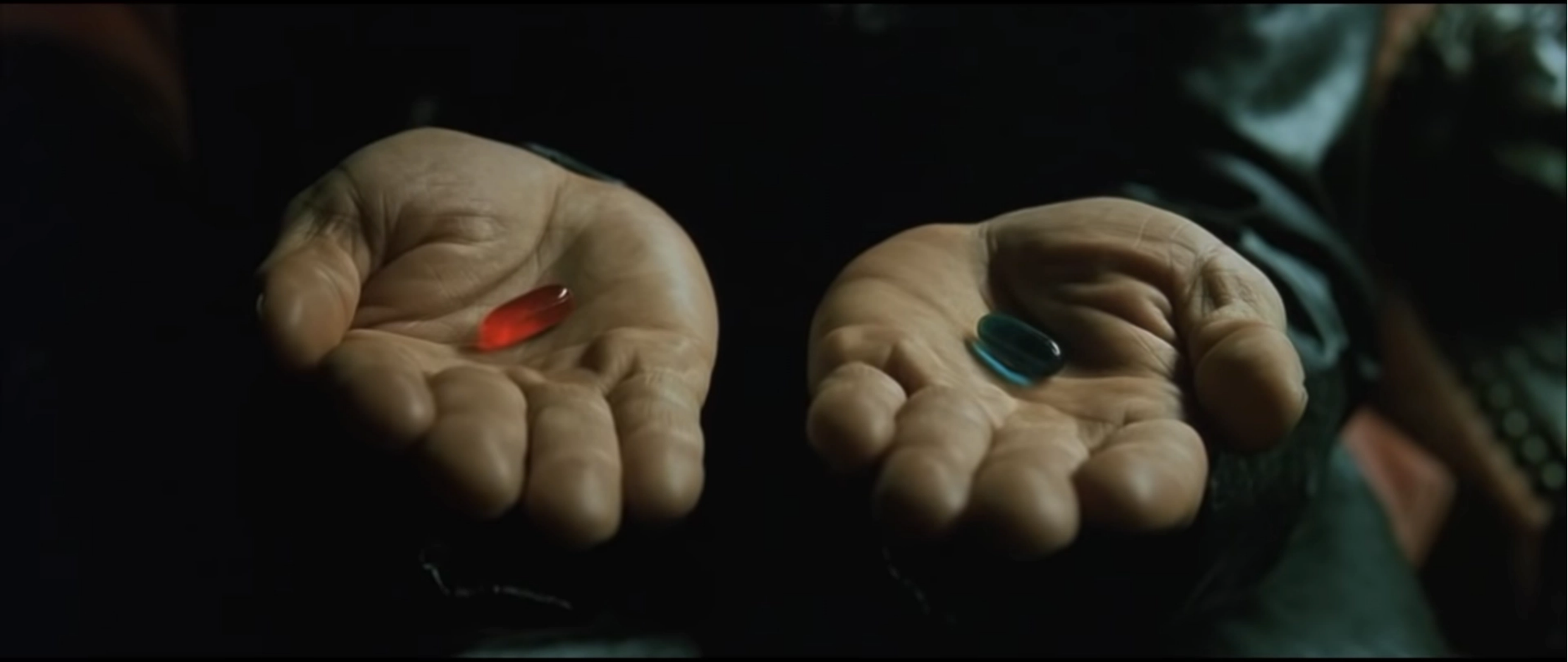
avionschool

Lesson 6.0 Programming Introduction

BATCH 4

DECEMBER 1, 2020


Pitch

LVX VENIT IN MVNDVN ET DILEXERVNT HOMINES MAGIS TENEBRAS QVAM LVCEM. Io. 3. 19

Maxima pars hominum cecis immersa tenebris
Volvitur assidue, et studio letatur inani:
Adspice ut obiecti obtutus in hereat umbris,
Vt VERI simulacra omnes mirentur amenti.

Et stolidi vanâ ludantur imagine rerum:
Quam pauci meliore luto, qui in lumine puro
Secreti à stolidâ turbâ, ludibria cernunt
Rerum umbras rectas, expendunt omnia lance:

Hi positâ erroris nebulâ dignoscere possunt
Vera bona, atque alios cecâ sub nocte latentes
Extrahere in claram lucem conantur, at illis
Nullus amor lucis, tanta est rationis egestas.

C.C. Harlemensis Inv.
Saenredam Sculpsit.
Henr. Hondius excudit.
1604.

H.L. SPIEGEL FIGVRARI ET SCVLPI CVRAVIT. AC DOCTISS. ORNATISSQZ D. PETRO PAAW IN LVGDVN. ACAD. PROFESSORI MEDICO D.D.

Code

- source code
- set of special instructions to tell the computer what to do
- data that controls the actions performed on the computer's memory
- collection of statements
- a machine with lots of separate parts interconnecting with each other and contributing to the operation of the whole
- program - a building of thought (costless, weightless)
- keeping programs under control is the main problem of programming
- the art of programming is the skill of controlling complexity
- developed in practice, not learned from a list of rules

Syntax

- rules for valid format and combinations of instructions
- Grammar

Program

- collections of statements, which together describe all the steps that it takes to perform your program's purpose.

Problem: Add the numbers from 1 to 10 and print out result.

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

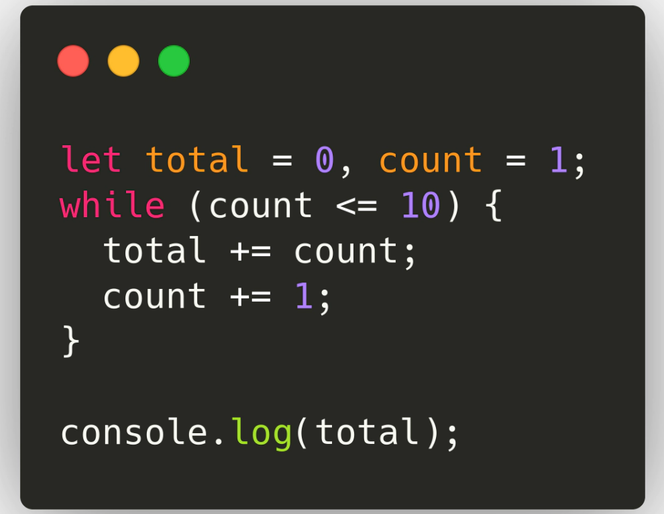
1. Store the number 0 in memory location 0.
2. Store the number 1 in memory location 1.
3. Store the value of memory location 1 in memory location 2.
4. Subtract the number 11 from the value in memory location 2.
5. If the value in memory location 2 is the number 0, continue with instruction 9.
6. Add the value of memory location 1 to memory location 0.
7. Add the number 1 to the value of memory location 1.
8. Continue with instruction 3.
9. Output the value of memory location 0.

```
 Set “total” to 0.
 Set “count” to 1.
[loop]
 Set “compare” to “count”.
 Subtract 11 from “compare”.
 If “compare” is zero, continue at [end].
 Add “count” to “total”.
 Add 1 to “count”.
 Continue at [loop].
[end]
 Output “total”.
```

1 + 2 + … + 10 = 55

Statements

- a group of words, numbers, and operators that performs a specific task in a computer language.
- semicolon



```
let total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}

console.log(total);
```

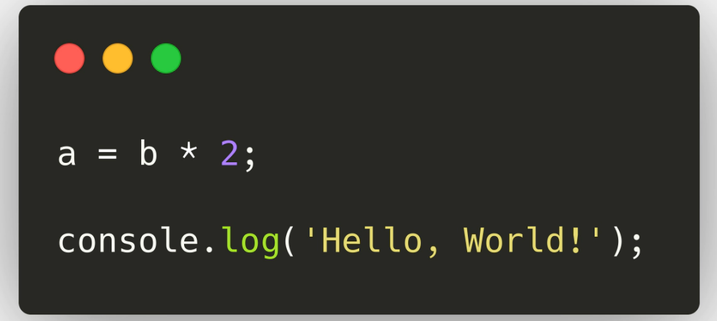
Semicolon

The Rules of Automatic Semicolon Insertion

1. when the next line starts with code that breaks the current one (code can spawn on multiple lines)
2. when the next line starts with a `}`, closing the current block
3. when the end of the source code file is reached
4. when there is a `return` statement on its own line
5. when there is a `break` statement on its own line
6. when there is a `throw` statement on its own line
7. when there is a `continue` statement on its own line

Expression

- any reference to a variable or value, or a set of variables and values combined with operators
- building blocks of statements
- general expression / expression statement, call expression



```
a = b * 2;

console.log('Hello, World!');
```

- 2 is a *literal value expression*
- b is a *variable expression*, which means to retrieve its current value
- b * 2 is an *arithmetic expression*, which means to do the multiplication
- a = b * 2 is an *assignment expression*, which means to assign the result of the b * 2 expression to the variable a

Program Execution

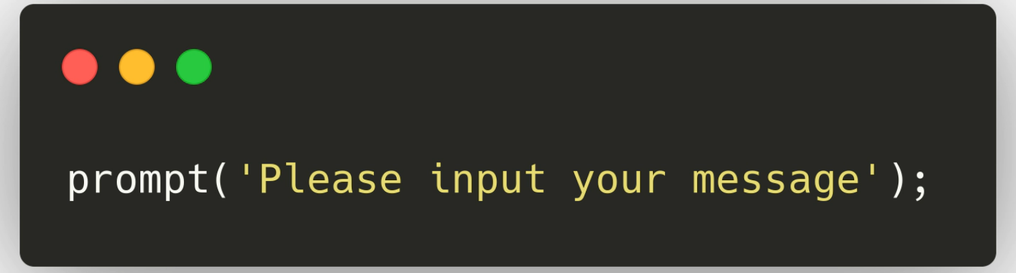
- Q: How do collections of programming statements tell the computer what to do?
- A: The program needs to be "executed" (i.e. running the program)
- statements - readable, developer friendly (but we know the computer still understands by bits)
- interpreter , compiler - used to translate the code you write into commands that the computer understand
- on some language, this translation of commands is typically done from top to bottom, line by line, every time the program is run, which is usually called interpreting the code
- on other languages, the translation is done ahead of time, called compiling the code, so when the program runs later, what's running is actually the already compiled computer instructions ready to go. (Java, C, C++, Ruby)

Output

- log - function call
- console - object reference

Input

- information received from the user
- prompt



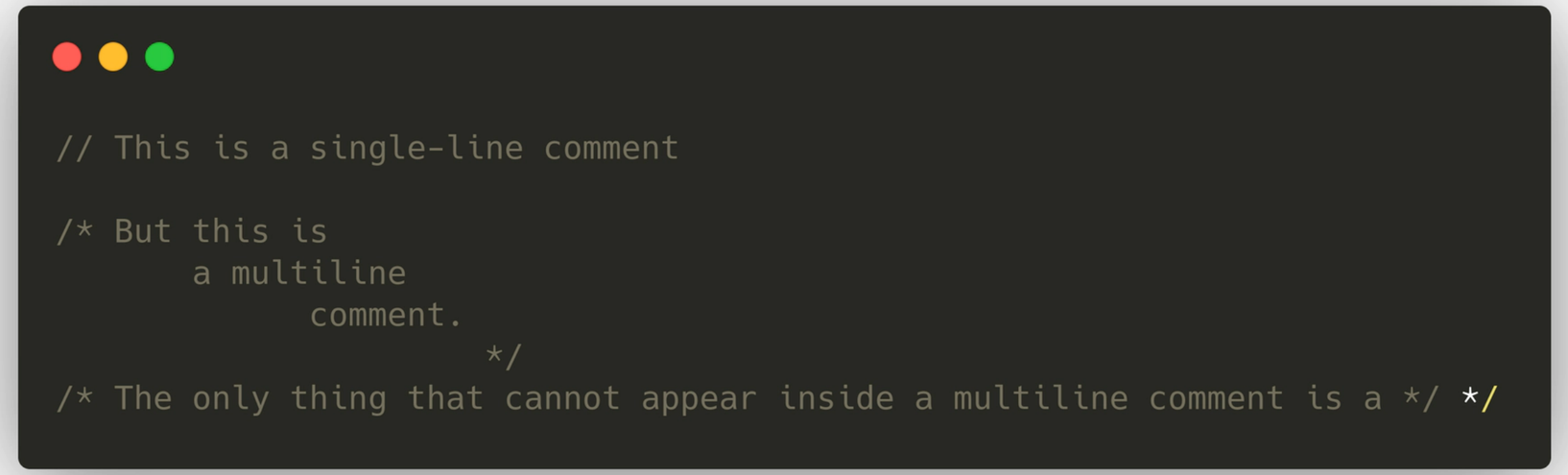
Operators

- how we perform actions on variables and values
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators

Code Comments

- writing code is that it's not just for the computer. Code is every bit as much, if not more, for the developer as it is for the compiler.
- Your computer only cares about machine code, a series of binary 0s and 1s, that comes from *compilation*.
- You should strive not just to write programs that work correctly, but programs that make sense when examined. You can go a long way in that effort by choosing good names for your variables and functions.
- Code without comments is suboptimal.
- Too many comments (one per line, for example) is probably a sign of poorly written code.
- Comments should explain *why*, not *what*. They can optionally explain *how* if that's particularly confusing.

Code Comments



```
// This is a single-line comment

/* But this is
       a multiline
             comment.
                      */
/* The only thing that cannot appear inside a multiline comment is a */ */
```

Variables

- hold values to be used by the program
- think of the characters you see in your code as symbolic container or placeholders for the values themselves
- by contrast, a value itself is called a *literal value*

```
var symbolicPlaceholder = 'value';

7;
```

Variables

- Most useful programs need to track a value as it changes over the course of the program, undergoing different operations as called for by your program's intended tasks.
- Your computer only cares about machine code, a series of binary 0s and 1s, that comes from *compilation*.
- You should strive not just to write programs that work correctly, but programs that make sense when examined. You can go a long way in that effort by choosing good names for your variables and functions.
- other languages use types for value instead of variables, weak typing, dynamic typing - allows a variable to hold any type of value at any time.
- It's typically cited as a benefit for program flexibility by allowing a single variable to represent a value no matter what type form that value may take at any given moment in the program's logic flow.

Types & Values

- types - different representations for values
- coercion- forced conversion between types
- static typing or type enforcement - to prevent unintended value coercions

Types & Values

- types - different representations for values
- coercion- forced conversion between types
- static typing or type enforcement - to prevent unintended value coercions
- note that amount holds a running value that changes over the course of the program, illustrating the primary purpose of variables: managing program *state* - *t*racking the changes to values as your program runs.

```
var amount = 99.99;

amount = amount * 2;

console.log( amount );      // 199.98

// convert `amount` to a string, and
// add "$" on the beginning
amount = "$" + String( amount );

console.log( amount );      // "$199.98"
```

Constants

- Another common usage of variables is for centralizing value setting. This is more typically called *constants*, when you declare a variable with a value and intend for that value to *not change* throughout the program.

```
var TAX_RATE = 0.08;    // 8% sales tax

var amount = 99.99;

amount = amount * 2;

amount = amount + (amount * TAX_RATE);

console.log( amount );              // 215.9784
console.log( amount.toFixed( 2 ) ); // "215.98"
```

Blocks

In code, we often need to group a series of statements together

```
var amount = 99.99;

// a general block
{
    amount = amount * 2;
    console.log( amount );  // 199.98
}
```

Conditionals

- Let say, you want to buy a phone which costs $99.99. Of course, you need first to consult the current state of your wallet or bank account to answer whether you can buy one or not.

Conditionals

```
var bank_balance = 302.13;
var amount = 99.99;

if (amount < bank_balance) {
    console.log( "I want to buy this phone!" );
}
```

Conditionals

- Obviously, this is just a simple "yes or no" question.

```
var bank_balance = 302.13;
var amount = 99.99;

if ( amount < bank_balance ) {
    console.log( "I'll confirm the transaction!" );
}
// otherwise:
else {
    console.log( "No, thanks." );
}
```

Conditionals

- What if the phone store employee asked you: "Do you want to add on the extra screen protectors to your purchase, for $9.99?"

```
const ACCESSORY_PRICE = 9.99; // // can we afford the extra purchase?

var bank_balance = 302.13;
var amount = 99.99;

if ( amount < bank_balance ) {
    console.log( "I'll confirm the transaction!" );
}
// otherwise:
else {
    console.log( "No, thanks." );
}
```

Loops

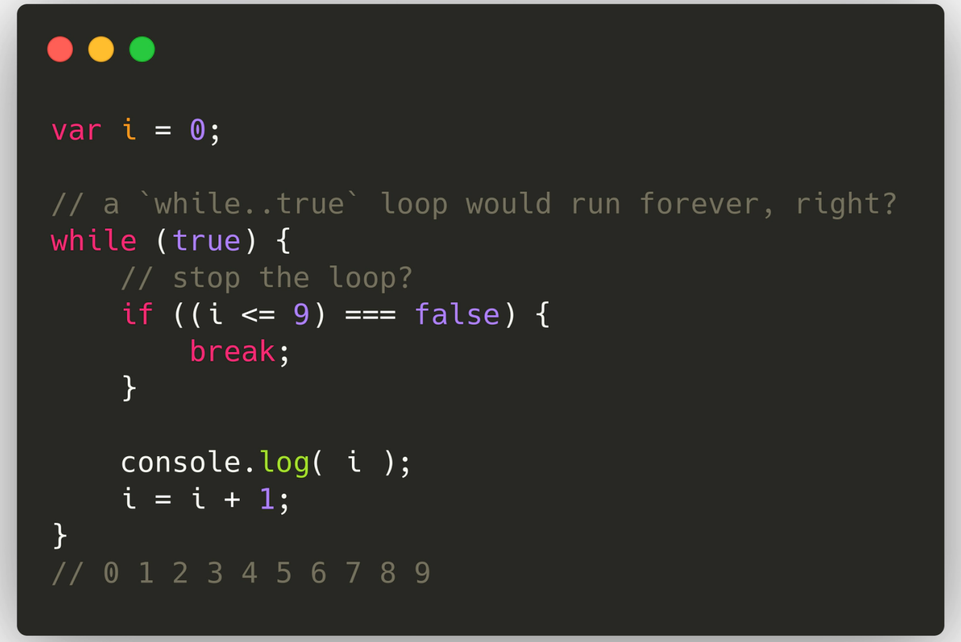
- Repeating a set of actions until a certain condition fails -- in other words, repeating only while the condition holds -- is the job of programming loops; loops can take different forms, but they all satisfy this basic behavior.
- During busy times, there's a waiting list for customers who need to speak to the phone store employee. While there's still people on that list, she just needs to keep serving the next customer.
- A loop includes the test condition as well as a block (typically as `{ .. }`). Each time the loop block executes, that's called an *iteration*.

Loops

```
while (numOfCustomers > 0) {
    console.log( "How may I help you?" );

    // help the customer...

    numOfCustomers = numOfCustomers - 1;
}
```

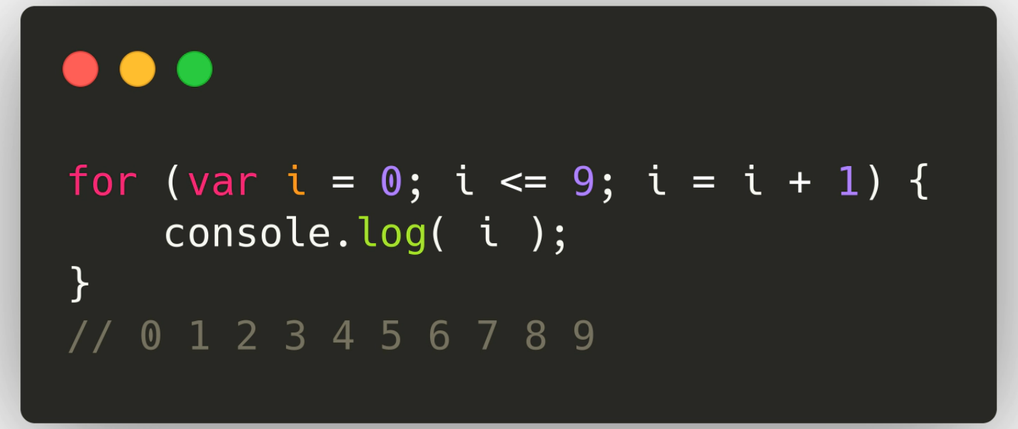
```
do {
    console.log( "How may I help you?" );

    // help the customer...

    numOfCustomers = numOfCustomers - 1;
} while (numOfCustomers > 0);
```

Loops



```
var i = 0;

// a `while..true` loop would run forever, right?
while (true) {
    // stop the loop?
    if ((i <= 9) === false) {
        break;
    }

    console.log( i );
    i = i + 1;
}
// 0 1 2 3 4 5 6 7 8 9
```

Loops



```
for (var i = 0; i <= 9; i = i + 1) {
    console.log( i );
}
// 0 1 2 3 4 5 6 7 8 9
```

Functions

- The phone store employee probably doesn't carry around a calculator to figure out the taxes and final purchase amount. That's a task she needs to define once and reuse over and over again. Odds are, the company has a checkout register (computer, tablet, etc.) with those "functions" built in.
- Similarly, your program will almost certainly want to break up the code's tasks into reusable pieces.
- A function is generally a named section of code that can be "called" by name, and the code inside it will be run each time.

Functions

```
function printAmount() {
    console.log( amount.toFixed( 2 ) );
}

var amount = 99.99;

printAmount(); // "99.99"

amount = amount * 2;

printAmount(); // "199.98"
```

Functions

```
function printAmount(amt) {
    console.log( amt.toFixed( 2 ) );
}

function formatAmount() {
    return "$" + amount.toFixed( 2 );
}

var amount = 99.99;

printAmount( amount * 2 );      // "199.98"

amount = formatAmount();
console.log( amount );          // "$99.99"
```

Functions

```
const TAX_RATE = 0.08;

function calculateFinalPurchaseAmount(amt) {
    // calculate the new amount with the tax
    amt = amt + (amt * TAX_RATE);

    // return the new amount
    return amt;
}

var amount = 99.99;

amount = calculateFinalPurchaseAmount( amount );

console.log( amount.toFixed( 2 ) );
```

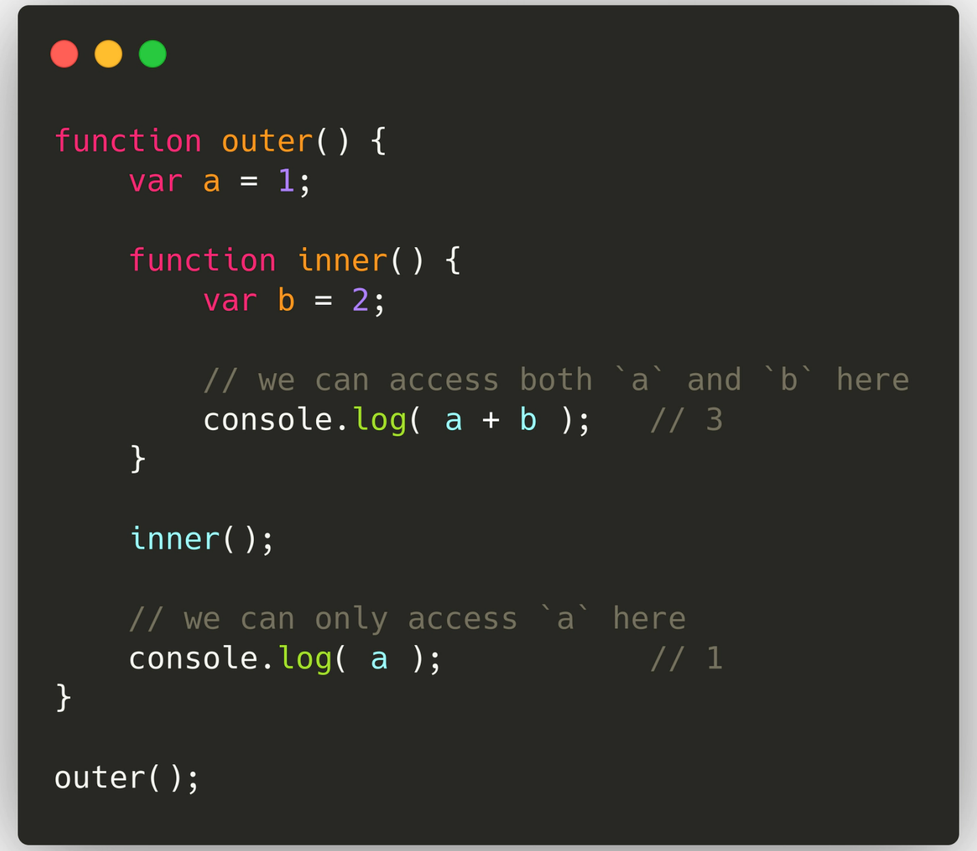
Scope

- If you ask the phone store employee for a phone model that her store doesn't carry, she will not be able to sell you the phone you want. She only has access to the phones in her store's inventory. You'll have to try another store to see if you can find the phone you're looking for.
- lexical scope

Scope

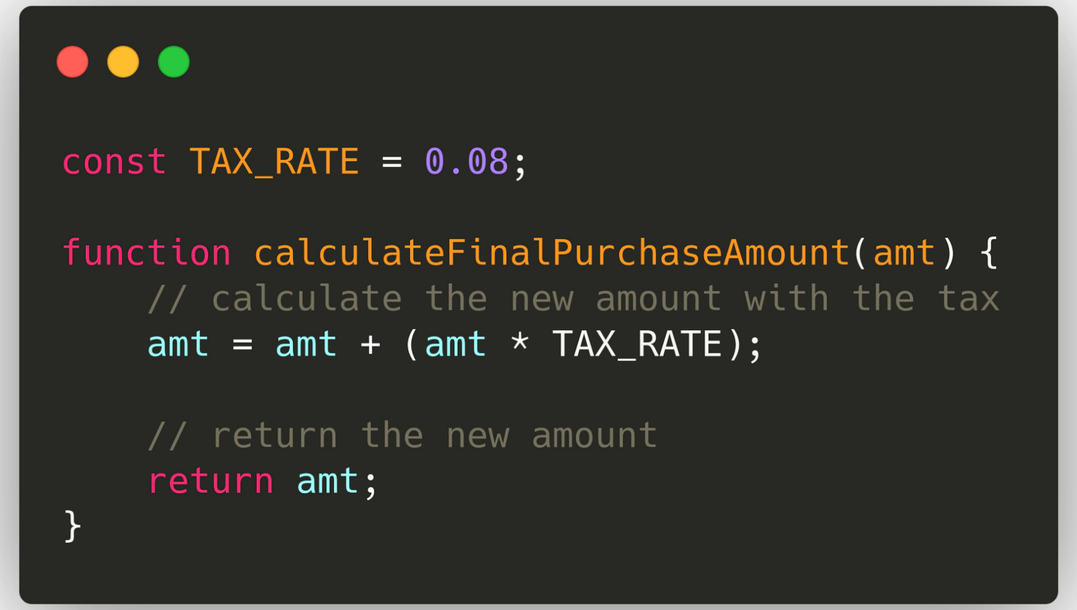
```
function one() {
    // this `a` only belongs to the `one()` function
    var a = 1;
    console.log( a );
}

function two() {
    // this `a` only belongs to the `two()` function
    var a = 2;
    console.log( a );
}

one();      // 1
two();      // 2
```

Scope



```
function outer() {
    var a = 1;

    function inner() {
        var b = 2;

        // we can access both `a` and `b` here
        console.log( a + b );   // 3
    }

    inner();

    // we can only access `a` here
    console.log( a );           // 1
}

outer();
```

Scope



```
const TAX_RATE = 0.08;

function calculateFinalPurchaseAmount(amt) {
    // calculate the new amount with the tax
    amt = amt + (amt * TAX_RATE);

    // return the new amount
    return amt;
}
```

Activity: Practice!

- Write a program that accepts the following input from user: (a) bank account balance and (b) how many phones the user wants to buy. Use the prompt() function on browser to ask these two things.
- You should set up some constants for the tax rate(0.08), phone price(99.99), screen protector price(9.99).
- You will keep purchasing phones as many as the user wanted and as long as the user still has remaining balance.
- You'll also buy a screen protector for each phone as long as you can still buy another phone the next time even without the accessory. You can buy a phone without an accessory.
- Calculate the total price of your purchase. Please take note of the tax.
- Check the amount against your bank account balance to see if you can afford it or not.
- After you've calculated your purchase amount, prompt() the breakdown: how many phones and accessories you can buy,and the calculated purchase amount properly formatted, with the question if the user wants to proceed with the transaction or not.
- Whether the user typed 'yes' or 'no', alert the respective summary of the transaction: how many phones and accessories bought, total price spent, and remaining bank account balance.
- You should define functions for calculating the tax and for formatting the price with a "$" and rounding to two decimal places.

Review

- Learning programming doesn't have to be a complex and overwhelming process.

- You need *operators* to perform actions on values.
- You need values and *types* to perform different kinds of actions like math on `numbers` or output with `strings`.
- You need *variables* to store data (aka *state*) during your program's execution.
- You need *conditionals* like `if` statements to make decisions.
- You need *loops* to repeat tasks until a condition stops being true.
- You need *functions* to organize your code into logical and reusable chunks.

- Code comments are one effective way to write more readable code, which makes your program easier to understand, maintain, and fix later if there are problems.
- Finally, don't neglect the power of practice. The best way to learn how to write code is to write code.