# Chapter 1: Into Programming

code,Js

## Computer language:

Source code. Built primarily for the developer - other devs or your future self.

Syntax & grammar combine to write a valid or coherent combo of char to make the computer perform a task or receive a statement.

## Statement:

Group of words, numbers and operators that perform a specific task.

```
a = b * 2;
```

Above, assignment statement: comprised of **literal** value, an **operator** and **variables**.

## Expressions:

Combine to create statements. The arithmetic statement above is created by literal value expression, variable expression and arithmetic expression, finally assignment expression.

A general expression can stand alone like, `b * 2;` and is called **expression statement**. It is not used commonly, as they are not particularly useful.

A more common expression is **call expression**: the entire statement is the function call expression itself: `alert( a );`
Operator precedence and expression precedence obey specific rules

## Executing a Program:

Generally an interpreter or compiler translates code to machine language. Compiling required an extra step before runtime, and interpreting generally means the code is compiled at runtime, line by line.
Js is not strictly an interpreted language, as the engine compiles the program on the fly then immediately runs the compiled source code.

## Run Code in Console:

`Option + Cmd + i`
In console: `Shift + Return` for multiple lines of input
Input: `b = 21;`

Output: `console.log( b );`
`alert();` and `prompt();` are output and input function calls as well.

## Operators:

Assignment, `=`
Math, `+`, `-`, `/`, `*`, `%`
Compound Assignment `+=`. `-=`, `*=`, `/=`, `%=`
Increment/Decrement, `++`/–
Object property access, `obj.a` or `obj[a]`
Equality, `==` loose-equals, `===` strict-equals, `!=` loose not-equals, `!==` strict not-equals
Comparison, `<`, `>`, `<=`, `>=`
Logical, `&&`, `||`

## Types:

### Primitive values:

`number`
`string`
`boolean`

string literal “`hello`"
boolean literal `true`
number literal `19`
arrays, objects, functions

### Type Coercion:

implicit vs. explicit - Js implicitly uses ***coercion*** to convert values. String to number and vice versa.
Can perform this explicitly using built-in function `Number(..);`
This type coercion is labeled a weakness by some and a strength by the author, who urges to learn it in depth

## Code Comments:

// single-line comment
/* multi-line
    comment */

Comments should be the why, not what of code. Your future self will thank you.

## Variables:

Js uses weak-typing, or dynamic-typing on variable assignments, as opposed to static-typing languages like

Java which exercise type-enforcement.

Dynamic typing allows Js variables to hold any type without type-enforcement.

Variables hold a running value that changes over course of program, managing program ***state***.

***State***: Tracking the changes to values as the program runs.

***Constants*** are variables, declared: (in ES6)
They can still be changed, nothing really prevents the values from changing, except a warning, (failure in strict mode)
`const MY_CONSTANT = 999;`

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

var amount = 99.99;

amount = amount * 2;

amount = amount * (1 + TAX_RATE)

console.log(amount);            // 215.9784
console.log(amount.toFixed(2));   // 215.98
// Adapted From: Kyle Simpson. “You Don’t Know JS: Up & Going.” iBooks.
```

## Blocks:

Group series of statements together, typically attached to a control structure like conditional or loop.
`{ .. }`

## Conditionals:

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

if (amount < bank_balance) {
  console.log("I want to buy this.");
} else {
  console.log("Can't afford it, sorry.");
}
```

Values coerced to boolean, truthy and falsey. “”, 0, false,NaN are all falsey.

## Loops:

`while`

`while..do`

```javascript
// conditional tested before first iteration
while (numCustomers > 0) {
  console.log("How can I help ya?");
  // Helps customer
  numCustomers -= 1;
}

// versus after the first iteration:
do {
  console.log("How can I help ya?");
  // Helps customer
  numCustomers -= 1;
} while (numCustomers > 0);
```

`for`

```javascript
for (var i = 0; i <= 9; i++) {
  console.log(i);
}
```

## Functions:

Named section of code to be called by name and code executed.

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

var amount = 99.99;
printAmount();

amount *= 2;
printAmount();
```

Optionally take arguments as parameters, and optionally return a value.

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

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

var amount = 99.99;
printAmount(amount * 2);
amount = formatAmount();
console.log(amount);
```

## Scope:

Lexical scope is the namespace for a collection of given variables available to a context such as a block of function. Each function has own scope in Js.

Variable names must be unique.
`const` allows these values to penetrate lexical scope.
Scope can be nested as inner and outer function. The inner function has access to the outer scoped variables, but not vice versa.

## Practice!

- Write a program to calculate the total price of your phone purchase. You will keep purchasing phones (hint: loop!) until you run out of money in your bank account. You'll also buy accessories for each phone as long as your purchase amount is below your mental spending threshold.
- After you've calculated your purchase amount, add in the tax, then print out the calculated purchase amount, properly formatted.
- Finally, check the amount against your bank account balance to see if you can afford it or not.
- You should set up some constants for the "tax rate," "phone price," "accessory price," and "spending threshold," as well as a variable for your "bank account balance."
- You should define functions for calculating the tax and for formatting the price with a "$" and rounding to two decimal places.
- Bonus Challenge: Try to incorporate input into this program, perhaps with the `prompt(..)` covered in "Input".