San Diego Global Knowledge University

Programming Fundamentals

Session #2

Introduction

- Programming is the act of constructing a program – a set of instructions telling a computer what to do.
- A programming language is an artificially constructed language used to instruct computers.

JavaScript

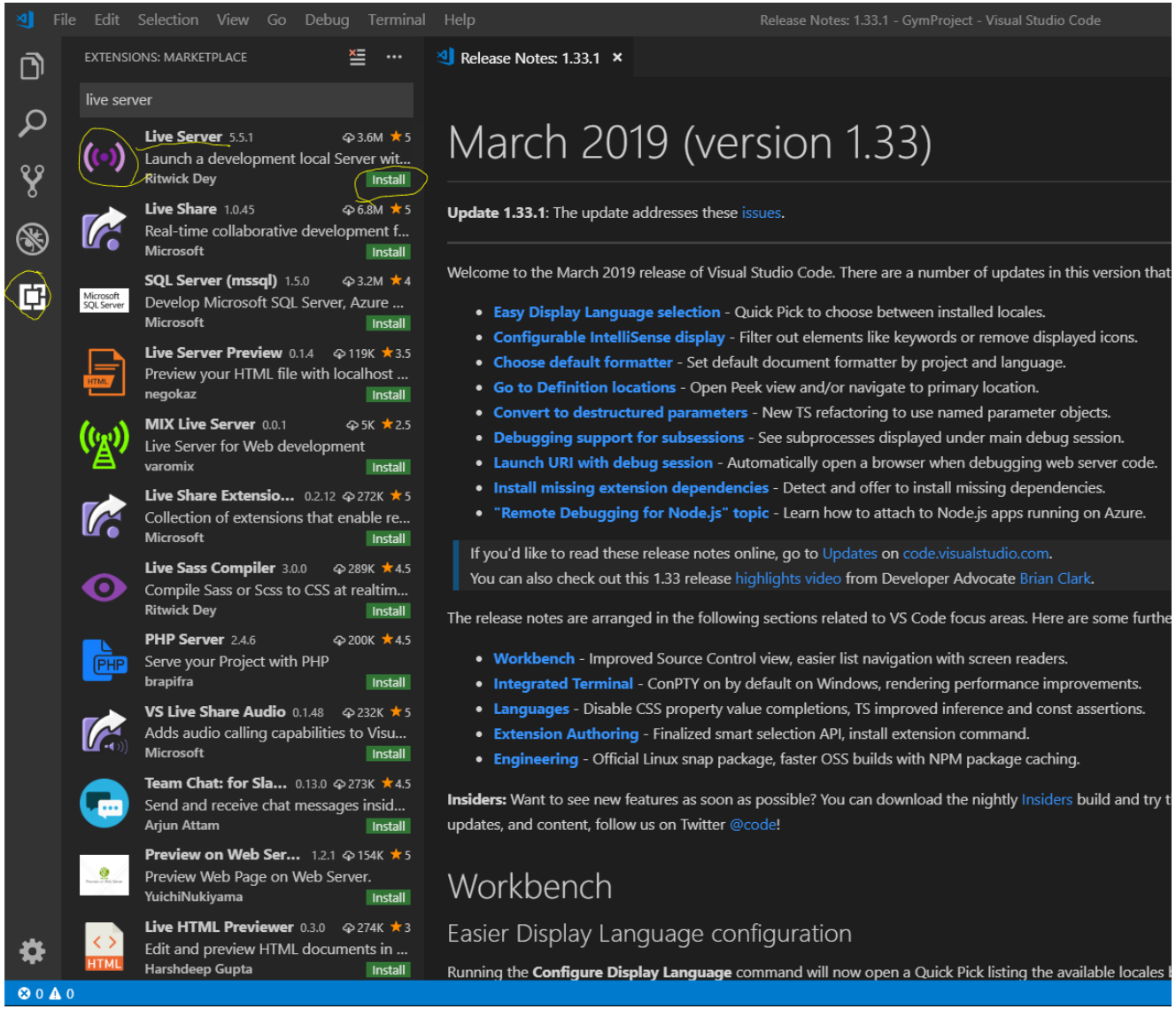
- Scripting language
- Every browser support JS, so you don’t need to install any addiotional material.

Use the console on the browser to display “hello world”.

- JS is a programming language (variables, functions, loops).

Extensions

- JavaScript (ES6)
- Beautify
- Bracket pair colorizer



Syntax Basics

Understanding statements, variable naming, whitespace, and other basic JavaScript syntax.

Numbers

Values of the *number* type are, unsurprisingly, numeric values. In a JavaScript program, they are written as follows:

>13

Fractional numbers are written by using a dot.

> 9.81

For very big or very small numbers, you may also use scientific notation by adding an *e* (for *exponent*), followed by the exponent of the number.

>2.998e8 (That is 2.998 × 108 = 299,800,000).

Calculations with whole numbers (also called *integers*) smaller than the aforementioned 9 quadrillion are guaranteed to always be precise. Unfortunately, calculations with fractional numbers are generally not.

Arithmetic

The main thing to do with numbers is arithmetic.

>100 + 4 * 11

The + and * symbols are called *operators.*

>(100 + 4) * 11

Multiplication and division

```
2 * 3;
2 / 3;
```

Incrementing and decrementing

```
var i = 1;

var j = ++i;  // pre-increment:  j equals 2; i equals 2
var k = i++;  // post-increment: k equals 2; i equals 3
```

Strings

The next basic data type is the *string*. Strings are used to represent text. They are written by enclosing their content in quotes.

```
`Down on the sea`
"Lie on the ocean"      "This is the first line\nAnd this is the second"
'Float on the ocean'
```

Strings cannot be divided, multiplied, or subtracted, but the + operator *can* be used on them. It does not add, but it *concatenates*—it glues two strings together. The following line will produce the string "concatenate":

```
"con" + "cat" + "e" + "nate"
```

Strings

Backtick-quoted strings, usually called *template literals*, can do a few more tricks. Apart from being able to span lines, they can also embed other values.

```
`half of 100 is ${100 / 2}`
```

Boolean

It is often useful to have a value that distinguishes between only two possibilities, like “yes” and “no” or “on” and “off”. For this purpose, JavaScript has a *Boolean* type, which has just two values, true and false, which are written as those words.

```
console.log(3 > 2)
// → true
console.log(3 < 2)
// → false

console.log("Itchy" != "Scratchy")
// → true
console.log("Apple" == "Orange")
// → false
```

Other similar operators are:
>= (greater than or equal to),
<= (less than or equal to),
== (equal to),
and != (not equal to)

Logical operators

There are also some operations that can be applied to Boolean values themselves. JavaScript supports three logical operators: *and*, *or*, and *not*.

```
console.log(true && false)
// → false
console.log(true && true)
// → true

console.log(false || true)
// → true
console.log(false || false)
// → false
```

The && operator represents logical *and*.
The || operator denotes logical *or*.
Not is written as an exclamation mark (!).

`!false` gives `true`.

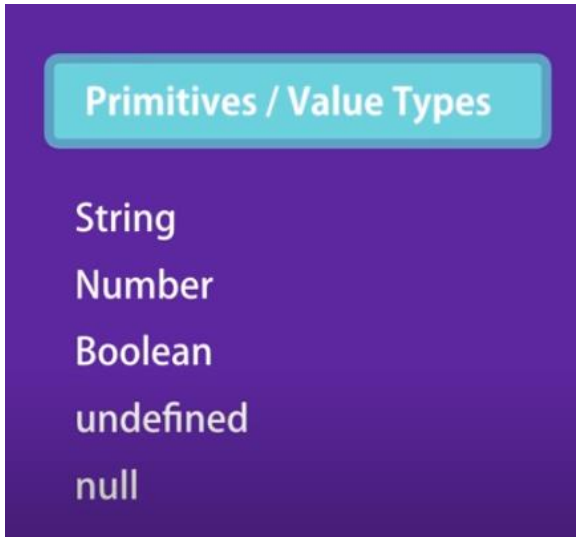
```
1 + 1 == 2 && 10 * 10 > 50
```

Program structure

- Create a HTML file.
- Add the script tag.
- Link the console with the script (at the end of the body section).
- Display “Hello world”.


Semicolons in writing
Semicolons in programming
iq.opengenus.org

Primitive / Value Types



Variables


JavaScript Variables
variable Initialization
variable naming
value assigned
var x=”Tech Altum”;

Variables

`snake_case`

Pros: Concise when it consists of a few words.

Cons: Redundant as hell when it gets longer.

```
push_something_to_first_queue, pop_what, get_whatever…
```

`PascalCase`

Pros: Seems neat.

```
GetItem, SetItem, Convert, ...
```

Cons: Barely used. (why?)

`camelCase`

Pros: Widely used in the programmer community.

Cons: Looks ugly when a few methods are n-worded.

```
push, reserve, beginBuilding, ...
```

Variables

```
/*myName is declared and initialized
   at the same time*/
var myName = "Dee";


/*myName is declared first and
initialized on a new line*/
var myName;
myName = "Dee";
```

Variables

```
let name = 'Mosh'; // String Literal
let age = 30; // Number Literal
let isApproved = false; // Boolean Literal
let firstName = undefined;
let lastName = null;
```

Variables

How does a program keep an internal state? How does it remember things? To catch and hold values, JavaScript provides a thing called a *binding*, or *variable*:

```
let caught = 5 * 5;
```

```
let ten = 10;
console.log(ten * ten);
// → 100
```

```
let mood = "light";
console.log(mood);
// → light
mood = "dark";
console.log(mood);
// → dark
```

```
let luigisDebt = 140;
luigisDebt = luigisDebt - 35;
console.log(luigisDebt);
// → 105
```

```
let one = 1, two = 2;
console.log(one + two);
// → 3
```

```
var name = "Ayda";
const greeting = "Hello ";
console.log(greeting + name);
// → Hello Ayda
```

Exercise #1

- Store the following into variables: number of children, partner's name, geographic location, job title.
- Output your fortune to the screen like so: "You will be a X in Y, and married to Z with N kids."

Exercise #2

1) Save the following information in variables using the correct data type: name, email, password, age, country, salary (monthly).

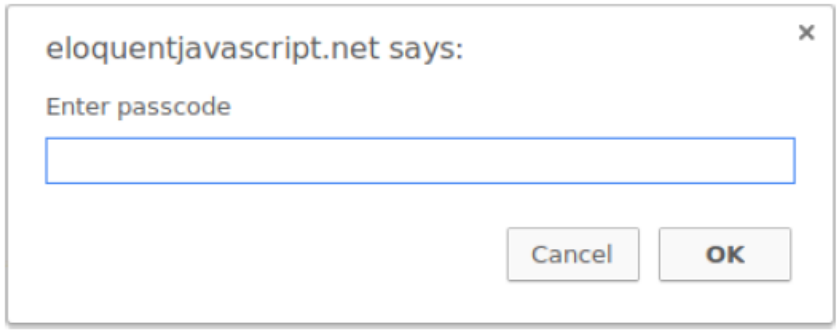
2) Display on the console the following information:

Name, email and salary (annual).

Functions

A lot of the values provided in the default environment have the type *function*. A function is a piece of program wrapped in a value. Such values can be *applied* in order to run the wrapped program.

```
prompt("Enter passcode");
```



Exercise: display on the console “Welcome <name>”

Executing a function is called *invoking*, *calling*, or *applying* it.

Exercise #3

- Modify the exercise #1, and receive from the prompt the information of the user.

Exercise #4

Want to find out how old you'll be? Calculate it!

- Store your birth year in a variable.
- Store a future year in a variable.
- Calculate your 2 possible ages for that year based on the stored values.
- For example, if you were born in 1988, then in 2026 you'll be either 37 or 38, depending on what month it is in 2026.
- Output them to the screen like so: "I will be either NN or NN in YYYY", substituting the values.

San Diego Global Knowledge University

Programming Fundamentals

Session #3

Control Flow

When your program contains more than one statement, the statements are executed as if they are a story, from top to bottom. This example program has two statements. The first one asks the user for a number, and the second, which is executed after the first, shows the square of that number.

```
let theNumber = Number(prompt("Pick a number"));
console.log("Your number is the square root of " +
            theNumber * theNumber);
```

Conditional Execution

Not all programs are straight roads. We may, for example, want to create a branching road, where the program takes the proper branch based on the situation at hand. This is called *conditional execution*.

Conditional execution is created with the if keyword in JavaScript.

```
let theNumber = Number(prompt("Pick a number"));
if (!Number.isNaN(theNumber)) {
  console.log("Your number is the square root of " +
              theNumber * theNumber);
}
```

```
if (1 + 1 == 2) console.log("It's true");
// → It's true
```

Conditional Execution

You often won't just have code that executes when a condition holds true, but also code that handles the other case.

```
let theNumber = Number(prompt("Pick a number"));
if (!Number.isNaN(theNumber)) {
  console.log("Your number is the square root of " +
              theNumber * theNumber);
} else {
  console.log("Hey. Why didn't you give me a number?");
}
```

If you have more than two paths to choose from, you can "chain" multiple if/else pairs together. Here's an example:

```
let num = Number(prompt("Pick a number"));

if (num < 10) {
  console.log("Small");
} else if (num < 100) {
  console.log("Medium");
} else {
  console.log("Large");
}
```

Conditional Execution

If you have more than two paths to choose from, you can “chain” multiple if/else pairs together. Here’s an example:

```
let num = Number(prompt("Pick a number"));

if (num < 10) {
  console.log("Small");
} else if (num < 100) {
  console.log("Medium");
} else {
  console.log("Large");
}
```

In-class exercise #2

Write a JavaScript program that accept two integers and display on the console the larger.

- Get the integers from the prompt.
- Hint: You need to use variables and conditional statements (if and else).

While and do Loops

Consider a program that outputs all even numbers from 0 to 12. One way to write this is as follows:

```
console.log(0);
console.log(2);
console.log(4);
console.log(6);
console.log(8);
console.log(10);
console.log(12);
```

That works, but the idea of writing a program is to make something *less* work, not more. If we needed all even numbers less than 1,000, this approach would be unworkable. What we need is a way to run a piece of code multiple times. This form of control flow is called a *loop*.

While loop

Looping control flow allows us to go back to some point in the program where we were before and repeat it with our current program state. If we combine this with a binding that counts, we can do something like this:

```
let number = 0;
while (number <= 12) {
  console.log(number);
  number = number + 2;
}
// → 0
// → 2
// …   etcetera
```

A statement starting with the keyword while creates a loop. The word while is followed by an expression in parentheses and then a statement, much like if. The loop keeps entering that statement as long as the expression produces a value that gives true when converted to Boolean.
The number binding demonstrates the way a binding can track the progress of a program. Every time the loop repeats, number gets a value that is 2 more than its previous value.

Do loop

A do loop is a control structure similar to a while loop. It differs only on one point: a do loop always executes its body at least once, and it starts testing whether it should stop only after that first execution. To reflect this, the test appears after the body of the loop.

```
let yourName;
do {
  yourName = prompt("Who are you?");
} while (!yourName);
console.log(yourName);
```

In-class exercise #2

Display on the console the numbers from 0 to 100 using:

a) The while loop

b) The do loop

Indenting code

- We’ve been adding spaces in front of statements that are part of some larger statement. These spaces are not required—the computer will accept the program just fine without them. In fact, even the line breaks in programs are optional. You could write a program as a single long line if you felt like it. The role of this indentation inside blocks is to make the structure of the code stand out.

For loop

JavaScript and similar languages provide a slightly shorter and more comprehensive form, the for loop.

```
for (let number = 0; number <= 12; number = number + 2) {
  console.log(number);
}
// → 0
// → 2
// …   etcetera
```

The part before the first semicolon *initializes* the loop, usually by defining a binding.
The second part is the expression that *checks* whether the loop must continue.
The final part *updates* the state of the loop after every iteration. In most cases,
this is shorter and clearer than a while construct

For loop

```
let result = 1;
for (let counter = 0; counter < 10; counter = counter + 1) {
  result = result * 2;
}
console.log(result);
// → 1024
```

In-class Exercise #3

Write a JavaScript program to construct the following pattern and display it on the console, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
```

In-class exercise #4

Write a JavaScript program which iterates the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

(This is actually an interview question that has been claimed to weed out a significant percentage of programmer candidates. So if you solved it, your labor market value just went up.)

Updating variables

Especially when looping, a program often needs to “update” a binding to hold a value based on that binding’s previous value.

```
counter = counter + 1;
```

JavaScript provides a shortcut for this.

```
counter += 1;
```

```
for (let number = 0; number <= 12; number += 2) {
  console.log(number);
}
```

For `counter += 1` and `counter -= 1`, there are even shorter equivalents: `counter++` and `counter--`.

Switch-case

It is not uncommon for code to look like this:

```
if (x == "value1") action1();
else if (x == "value2") action2();
else if (x == "value3") action3();
else defaultAction();
```

There is a construct called switch that is intended to express such a “dispatch”
in a more direct way.

```
switch (prompt("What is the weather like?")) {
  case "rainy":
    console.log("Remember to bring an umbrella.");
    break;
  case "sunny":
    console.log("Dress lightly.");
  case "cloudy":
    console.log("Go outside.");
    break;
  default:
    console.log("Unknown weather type!");
    break;
}
```

San Diego Global Knowledge University

Programming Fundamentals

Session #4

In-class exercise #5

Write a JavaScript program which compute, the average marks of the following students Then, this average is used to determine the corresponding grade.

Student Name	Marks
David	80
Vinoth	77
Divya	88
Ishitha	95
Thomas	68

The grades are computed as follows:

Range	Grade
<60	F
<70	D
<80	C
<90	B
<100	A