# [2-1]

JavaScript Fundamentals P1

# What is Programming?

Programming is the act of constructing a *program* --a set of precise instructions telling a computer what to do.

- It allows you to do things in seconds that would take forever by hand.
- It provides a wonderful exercise in abstract thinking.
- It's hard... but there is a relatively small set of fundamental principles.
- It's fun... but also *terribly* frustrating at times.

> The computer programmer is a creator of universes for which [s]he alone is responsible. Universes of virtually unlimited complexity can be created in the form of computer programs.
>
> - Joseph Weizenbaum, Computer Power and Human Reason

# What is JavaScript?

- JavaScript was introduced in 1995 as a way to add programs to web pages in the Netscape Navigator. 🤯
- JavaScript has almost nothing to do with the programming language named Java.
- JavaScript was stable for a long time at version 5 (ES5) from 2009 to 2015.
- In 2015, ES2015 was released and included TONS of new functionality.
- There have been yearly releases since then, ES2017, ES2018, ES2019...

# What is JavaScript?




JavaScript is a multi-paradigm language, combining tools for

- Imperative Programming
- Functional Programming
- Object-oriented Programming

The style of programming we'll learn to use during our course is best described as procedural.

# [2-1]

JavaScript Types

# Six basic types in JavaScript



- Booleans
- Numbers
- Empty Values
- Strings
- Arrays
- Objects

If you are ever unsure of the type of a given value, you can use **typeof <VALUE>**

# Strings: Interpolation



```
let pi = 3.14;
let diameter = 30;
let radius = diameter / 2;

// declare a string
let introduction = `The area of a circle is πr².`

// declare a string with interpolation
let example = `A ${diameter}cm pizza has an area of ${pi * radius * radius}cm².`

// Concatenate the strings
let text = introduction + ' ' + example

console.log(text);
```

```
> The area of a circle is πr². A 30cm pizza has an area of 706.5cm².
```

# Arrays: Accessing the values in an array



⚠️⚠️ Indexing starts at 0 ⚠️⚠️

```
let anArray = ['bacon', undefined, 900, true]
```

1. What is the value of anArray[0]? **‘bacon’**
2. How do we access the value 900? **anArray[2]**

# Objects: Accessing the values in an object

Values in an object can be accessed with

- dot notation
- bracket notation

```
let person = { name: 'Bob', age: 23 };
```

1. How do we access ‘Bob’? **person.name** or **person[‘name’]**

# JavaScript Operators

- Arithmetic
- Comparison operators
- Logical operators

Comparison operators and logical operators are usually combined in an expression to create a boolean value, like this **2 > 1 && -1 < 0** which is **true**.

# Exercise 1

Look at these expressions below and determine whether they evaluate to true or false

| | |
|---|---|
| true \|\| false | |
| false && false | |
| 1 < 2 && 2 > 1 | |
| 31 < 13 \|\| 1 < 2 && 3 > 1 | |
| 400 <= 400 && 399 < 400 && (30 > 31 \|\| 400 > 31) | |
| true && false && false \|\| false && true | |
| true && false \|\| true \|\| false | |
| true && false && false \|\| false && true ? true && false && false \|\| false && true : 1 < 2 && 2 > 1 | |

# Exercise 2

Given this data structure:

```
let data = [0, [], [], [1, 2, 3, [4]]];
```

| | |
|---|---|
| How would you access the value `0`? | |
| How would you access the value `3`? | |
| How would you access the value `4`? | |

# Exercise 3

```
{ label: 'corn', price: 5.3 + '$' };
{ ISBN: 53532, isAvailable: true, author: 'Nakamoto' };
```

- List the number of properties for each object.
- For each property, indicate its key and its value.
- For each property value, indicate its type.

| | |
|---|---|
| | |
| | |
| | |

# Exercise 4

```
// Given
let person = { name: 'Bob', age: 23 };
let name = 'John';
```

What is the value of the following expressions?

| | |
|---|---|
| person.name | |
| person[‘name’] | |
| person[name] | |

# Exercise 5

```
// Given
let person = { name: 'Bob', age: 23 };
let key = 'name';
```

What is the value of the following expressions?

| | |
|---|---|
| person.key | |
| person[‘key’] | |
| person[key] | |

# [2-1]

Expressions and Statements

# Expressions / Statements

**Expressions** are the language equivalent to phrases, or a partial statement.

**Statements** express an action to be carried out.

*The simplest possible statement is an expression that ends in a* ***;***

# Expressions / Statements

- An **expression** is content to just produce a value, which can then be used by the enclosing code.
- A **statement** stands on its own. It should *affect the world*.
  - display something on the screen.
  - change the internal state of the machine in a way that will affect the statements that come after it.

# [2-1]

Variables

# JS Variables (or bindings)

- Variables are used to hold data values.
- Variables are essentially data containers.
- They can also hold functions
- There are 3 ways to declare variables.

```
var name = 'Rick';

const profession = 'Scientist';

let grandson = 'Morty';
```

# Defining Variables

- Use either **const** or **let**
  - **const** for variables that *will never change*.
  - **let** for variables that *may change*.

- Never use **var**. It really should be deprecated...

# Defining Variables

Naming conventions

- Can be just about anything
- Cannot contain spaces.
- Can contain numbers but cannot start with a number.

Make your variable names *meaningful.* This will ease the debugging process considerably.

# Defining Variables

Camel Case, underscores and dashes

- Camel Case: myVariableName
- Underscores: my_variable_name or MY_VARIABLE_NAME
- Dashes: **doesn’t work! Will definitely break your code.**

In the course, we use **camelCase** most of the time.

# Defining Variables

Reserved Words

The following words cannot be used as variable names. JavaScript reserves them for its own internal use.

break case catch class const continue debugger default delete do else enum export extends false finally for function if implements import interface in instanceof let new package private protected public return static super switch this throw true try typeof var void while with yield

# [2-1]

Control Flow

# JS Control Flow

JavaScript executes a program from start to finish, in order.

We will often need a program to do things *conditionally.*

# The if statement

The **condition** must be true for the code block to run.

```
if (condition) {
  // do something
}
```

## Example 1

```
if (number > 16) {
  console.log('Hold');
}
```

## Example 2

```
let allegiance = '';
if (forceUseGood === true) {
  allegiance = 'jedi'
} else {
  allegiance = 'dark side'
}
```

# Exercises (5 min)

Go to codesandbox...

```
// 1. If it rains, I stay home.
let currentWeather = 'rainy';



// 2. If I am hungry, I eat.
let hunger = true;



// 3. If it's 10pm, I go to bed. If not, I write code.
let currentHour = 22;
```

# Problem

Here is a program that outputs all even numbers between 0 and 12.

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

This is tedious, but manageable.

*What if we wanted to output all of the even number between 0 and 1000?*

# Solution (loops)

This is where loops come in!

# The while loop

Output all of the even numbers between 0 and 1000

```
let number = 0;
while (number <= 1000) {
  console.log(number);
  number = number + 2;
}
```

1. number is set to 0.
2. while loop condition is true ( 0 < 1000), so it
   a. Outputs the number
   b. Adds 2 to the number
3. number is now 2
4. repeats until number is 1002.

# The while loop

Another example of a while loop...

Write a program that outputs 2 to the power of 10.

# The while loop

**while** is a great tool for some situations, but it can be a little tedious.

We need to

1. Define a counter
2. Define a condition for the loop to run
3. Increment the counter everytime the loop runs.

```
let counter = 0;
let value = 2;
while (number <= 1000) {
  console.log(number);
  number = number + 2;
}
```

# The for loop

Let’s convert that last example to use a **for** loop.

```
let counter = 0;
let value = 2;
while (number <= 1000) {
  console.log(number);
  number = number + 2;
}
```

```
for (let i = 0; i <= 1000; i++) {
  If (i % 2 === 0) {
    console.log(number);
  }
}
```

⚠️ The **only** way to break out of a **for** loop is for the condition to be false. ⚠️

# The for loop

⚠️ The **only** way to break out of a **for** loop is for the condition to be false. ⚠️

- If you write a **for** loop that *always* evaluates to true, your loop will continue forever.
- This is a bad thing... it will crash your environment (browser window, node env, etc.).
    - To fix it, you will need to force-quit the environment.

# Exercises!

Work in pairs to solve the three exercises.

You have 20 minutes.