# Begin to Code: Building Apps and Games in the Cloud

Rob Miles

Initial Draft

# Glossary

Welcome to the glossary. I hope you find it useful. Don't forget that you can try the code samples out in the console in the Developer Tools part of your browser. This is a great way to build your understanding of what the code does.

## Application

An application performs something that a user wants to do, whether it is play a game or write a book. Until recently an application was a single piece of software that you copied onto your device to use. However, nowadays you can also use applications from within your web browser. The browser creates a sandbox which is a safe environment for an application to run in. The sandbox provides all the services that the application requires and strictly controls that access the application has to the underlying computer. The application uses an API to make requests of the underlying system and these requests are satisfied by the browser rather than the operating system of the computer. The application itself may be made up of multiple co-operating pieces of software which work together to give the user experience. And the application may use cloud components to do this.

## Application Programming Interface (API)

A set of functions or methods exposed by a piece of software to allow other software to make use of facilities it provides. The facilities could be a single action (for example a mathematical calculation such as the sine of a value) or a more complex transaction (or example load a file of text).

## Application Software

Application software provides something for a user. It might be a game to play, a word processor or a simple text editor. Some applications are provided as part of an operating system, for example Windows 11 is supplied with a Paint program and text editor. However, users can also install applications on their computers to perform more advanced tasks. It is also possible to run applications via a modern browser. Part (or all) of the application may run on the local machine with other components provided by processes in the cloud.

## Argument

An argument is a value given to a call of a function. Within the function the matching parameter value is replaced by the argument that was supplied in the call.

```
doAddition (3,4);
```

In the statement above the arguments are the values 3 and 4 which are passed into the doaddition function.

```
function doAddition(p1, p2) {
    let result = p1 + p2;
    alert("Result:" + result);
}
```

The number of arguments to a function should match the number of parameters. If an argument is not supplied the value of the matching parameter will be set to undefined when the function runs. If a call has more arguments than parameters the excess arguments are ignored.

# Assign

# Block

A block of code in JavaScript is a number of statements that have been lumped together so they can be treated as a whole. You create a block by enclosing statements in braces (curly brackets). You put code into a block so that you can use a single conditional statement to control several actions:

```
if ( age>70 ) {
    console.log("Age too large. Using 70");
    age = 70;
}
```

The above code logs a message if the age is greater than 70. It also sets the value of age to 70. Both these statements are controlled by the condition testing the value of age. You also put code into a block when you create the body of a function:

# Class

# Cloud

# Closure

# Code

The word code is generally thought to mean the same as program. However, the original meaning of code was the very low-level instructions that the computer hardware runs to implement a program. A program might contain a statement that adds one to then value of a variable. The code for this might end up being a sequence of instructions to fetch the variable, add one to it and then store it back into computer memory. The phrase “machine code” refers to the code implemented by a particular type of hardware.

# Condition

# Context

Everything we do takes place in a particular context. You behave one way in a church, and another in a football stadium. JavaScript establishes the context of an operation by looking at the type of the operands that it is working on. For example, the + operator will perform numeric addition if applied to two numbers but will concatenate if applied to two strings.

# Compiler

A compiler is a program that takes in program text and then converts into machine code that can run directly on a computer. The compilation process takes place before the program runs.

Actually, this is a bit of an oversimplification. Sometimes a compiler doesn’t make machine code, sometimes it makes an intermediate code which is either performed or compiled when the code itself runs. However, all languages have a compilation process that takes place before a program is allowed to run.

Different programming languages have compilers that apply different levels of rigor to the program code they are looking at. The JavaScript compiler is on the relaxed side. It will accept programs that contain syntax discrepancies that would be rejected in other languages. This can result in JavaScript programs failing because of errors that would be detected in other languages. This is something we have to be mindful of when we write code.

# Computer

A computer is hardware that can run programs. A computer on its own is no use to you. It must have software to make it useful. Some computers run only one program. Others allow the user to select programs and add new ones. A computer is frequently part of another device, for example a mobile phone. Some devices also contain computers running "embedded" software to make them work. An example of this would be a remotely controllable light bulb.

# Data

# Declaration

Declaration of something tells JavaScript that the thing exists.

```
let x;
```

The above statement tells JavaScript of the existence of a variable with the name x. It doesn't give any information about the type of x or its initial contents. A declared variable contains the value of ***undefined***.

# Definition

When we define a variable, we are telling JavaScript all about it.

```
let age = 21;
```

The statement above defines a variable called age. JavaScript can infer that the variable is a number and that the initial value of the variable will be 21.

# Expression

# Function

A function is an object that contains a “body” made up of JavaScript statements that are performed when the function is called. A function also contains a name property which gives the name of that function. Functions can be called by name, at which point the program execution is transferred into the statements that make up the function body. When the function completes the execution returns to statement after the one called the function. Functions can be made to accept values to work on and a function can also return a result value.

Functions reduce program size. A behavior (sequence of program statements) which is used in many locations in a program can be made into a function which is called each time it is needed. Functions make programs easier to maintain because a fault in a function only needs to be fixed once. Functions can also be used to make programs easier to understand and work on. A task can be performed by several different functions. Each function can be written and tested independently of the others and the functions combined to make the finished solution.

# Global

# Identifier

An identifier is a name created by the programmer to refer to something that a program needs to keep track of. JavaScript has rules that determine how the identifier is constructed:

- The identifier must start with a letter (A-Z or a-z), a dollar $, or an underscore _
- The identifier can contain letters, digits (0-9), a dollar $, or an underscore _

It is best if your identifiers describe the thing they are connected to. An identifier called “a” is not very useful. One called “age” would be more useful. One called “ageInYears” would be very useful. If you want to use multiple words in an identifier the convention in JavaScript is to capitalize the first letter of each word inside the identifier. This is called “camel case” because the capital letters stick up like the humps on a camel’s back. Note that the case of the letters in the identifier is significant. JavaScript would regard “AgeInYears” and “ageInYears” as different identifiers. If a program fails with a complaint that it can’t find something, make sure that the identifier you are using is correct.

If you are choosing an identifier for a function or method a good idea is to use a *verb-noun* structure. The identifier displayMenu would work well for a function that displayed a menu.

If you are choosing a name for a class the identifier should start with a capital letter.

# Infinity

The range of JavaScript numbers includes the value of Infinity. A variable will be set to Infinity if the result of the calculation generates that value.

```
var x = 1/0;
```

The above statement sets the value of x to Infinity. If the value of x is printed it will display the value "Infinity". The implementation of infinity works in the way you might expect. If you add 1 to infinity you get Infinity. If you divide Infinity by any value it retains the value of Infinity. And if you divide any number by Infinity you get the result of zero. If you divide Infinity by Infinity you don't get one however, you get Not a Number (NaN). A program can set a variable to the value of Infinity and can also test for the value:

```
if(x==Infinity) console.log("result too large");
```

# Internet

The internet is based on a set of open standards created by the United States Department of Defense which are known as TCP/IP. This stands for "Transport Control Protocol/Internet Protocol" and sets out how to build local networks and connect them together to create a "network of networks" that can span the globe. The standards provide an addressing scheme for each physical device on the network and a resource naming scheme allowing users to identify services by names rather than their physical address. The Internet is not the only network that uses TCP/IP. You can create your own TCP/IP network in your bedroom if you like.

You might think of an internet connection as a cable running between two machines. However, this is not how it works. The sender breaks the data being sent into small packets which are sent individually. It is a bit like me sending you a loaf of bread by posting each slice in a different letter. Software in the receiver takes the packets, puts them in the correct order and passes them up to the application that is using the connection. The application using the connection (for example a browser loading a web page) is completely unaware of all this packing and unpacking, it just receives the data and does something with it.

The internet was designed to provide communications at a time when nuclear war looked distinctly possible. It uses a mesh of systems called “servers”, each of which maintains a “map” of the network and passes incoming packets of data across the network to its destination. If one of the servers suddenly becomes unavailable the internet servers around it will automatically route packets around it. Client machines connect to the servers to send and receive packets of data.

The Department of Defense decided that the best way to get lots of network connectivity around the world was to make the TCP/IP standard public and give away all the software that they wrote to make it work. This made it much easier for hardware manufacturers (and even hobbyists) to connect their machines together. The internet became very popular very quickly. It provides something that was revolutionary at the time it was introduced. The internet transfers packets of data anywhere in the world at the same cost – which is zero once you are connected. This was a genuine game changer for computing. Before the internet you had to lay your own cables or rent them from the telephone company if you wanted to connect machines together. And international communications were extremely expensive.

However, once you have connected a machine to the internet you can send packets and expect them to arrive at their destination irrespective of where that is. A packet may take longer to arrive at a more distant destination as it is passed from system to system on its journey, but longer journeys don’t cost extra. Two successive packets to the same destination may travel by completely different routes, but they both would get there. The job of the internet is to hide all this complexity and give users the impression that they are directly connected to a machine, even though it is on the other side of the planet. The internet can function across many forms of physical media including telephone lines, wired connections, wireless networks and cellphones.

You can think of the internet as a system of rails that connect places together. But just like a railway, the internet is not interesting until you start to move things around on it. Just as trains make a railway interesting, applications make the internet interesting. One of the first “killer applications” for the internet was electronic mail. A user could connect to their “mail server” machine which was connected to the internet. The mail server accepts messages and stores them for the user to read. The mail server also sends mail messages to other mail servers. However, the application that did the most to get users onto the internet was the World Wide Web.

# Let

Programs use ***variables*** to hold values which the program wants to work on. A program can declare a variable by using the keyword let. The useful thing about variables declared using let is that they are discarded when program execution leaves the block in which they are declared.

```
{
    let personName = "Rob";
    // do things with the variable personName
}
// at this point in the code the variable personName no longer exists
```

Consider the code above. The variable personName is declared inside a block of statements. The variable personName can be used within that block, but when the program reaches the end of the statements in the block and leaves it, the variable personName is discarded. This means that there is no chance of the name variable being confused with other variables called personName which might be used in other parts of the program.

Note that if the enclosing block contains a variable called personName (it would be declared outside the block shown above) the outer personName variable would not be accessible inside the block. But the outer personName variable would become accessible when the program leaves the block above.

Once you have learned about let you should look at the glossary entries for ***var*** and ***global*** which are also used in JavaScript to manage where variables can be used.

# Local

# Loop

# Machine code

# Method

# Network

Once we had lots of computers, we started to link them together to form networks. Networks make two things possible. Firstly, a network connection gives you remote access to the processing power of the distant computer. You can run your programs on a distant machine. But secondly, networks let you move data between systems. A program running on one machine can access data stored on another. Data can be centralized and made available to connected clients. A service can be provided by several cooperating systems rather than by a process running on a single machine.

The first computer networks were “proprietary”. Machines made by company A could not talk to machines made by company B. This made it much harder for customers of company A to switch to company B (which the computer companies rather liked). But then someone made a network that was so compelling that everyone wanted to connect their machines to it. This network was “the Internet”.

# New

# Not a Number (NaN)

JavaScript uses the value Not a Number (NaN) to mean that the result of an operation has not generated a numeric result. Consider the following:

```
var x = 1/"fred";
```

This statement sets the value of x to the result of dividing the value 1 by the string “fred”. This is a meaningless calculation. In some programming language an attempt to divide a number by a string would be rejected when the program was compiled or produce an error when the program runs. In JavaScript when a numeric expression cannot be evaluated the program keeps running but the value of the result of the expression is set to NaN. You can check to see if a variable contains NaN, but not in the way you might think:

```
if (x==NaN) console.log("x is not a number");
```

The above code doesn’t work. NaN is not a value as such, so it isn’t really meaningful to compare it with anything. But we do know that the value of NaN is not equal to anything, *including itself.*

```
if (x!=x) console.log("x is not a number");
```

The statement above checks to see if x is equal to itself. If this test fails the value in x is not a number. Note that JavaScript expression evaluation is also aware of the concept of infinity. So perhaps you might like to read that entry in the glossary next.

# Null

# Object

A ***primitive*** data value such as a number or a string can only hold a single value. A JavaScript object is a container that can hold a collection of data values. A value held inside an object is called a property and each property has a name. Objects can be used to describe physical items with properties for each descriptive item. I could create an object to describe my car properties by using code like this:

```
let car = {color:"white", make:"Nissan"};
```

This statement creates an object that contains two properties. The first property is the color of the car, the second property is the make of the car. A variable with the identifier "car" is set to refer to the object that has been created. A JavaScript program can access a property as follows:

```
console.log(car.color);
```

This would display the message "white" on the console. A program can update the contents of a property by assigning a new value to the property:

```
car.color="blue";
```

This statement would set the color property of the car to the value "blue". A program can also add a new property to an object simply by assigning a value to a new property name:

```
car.model="Cube";
```

The object referred to by car now has a model property which is set to the string "Cube". Properties can be functions as well as values.

```
car.toString = function (){return this.color+" "+this.make+" "+this.model};
```

This statement adds a new property to the object referred to by car. The new property is a function called toString which returns a string describing the object contents. The string contains the color, the name and the model of the car. Note that the keyword this is used in the function to get a reference to the object that the function is part of. We can now call the toString function on the object referred to by car:

```
console.log(car.toString());
```

This would log the string “blue Nissan Cube” in the console (if the color property had been updated from the original white). You can create and manage an object by managing object properties like this, but you can make your objects in a more cohesive way by using JavaScript ***classes***. Objects are managed by ***reference***.

# Packet

# Parameter

A program can pass data into a call of a function or method by adding an argument to the call:

```
doDisplay("Hello");
```

Above you can see a call of a function called doDisplay. The function has a single argument which is the string “Hello”. Within the definition of the function the data item supplied is called a parameter.

```
function doDisplay(message){
    alert(message);
}
```

Above is the function doDisplay. The parameter to the function is called message. When the function call above runs the value of message is set to the string “Hello”. The function calls the alert function which displays the message for the user. Note that the parameter to a function is not given a type.

```
doDisplay(99);
```

The statement above calls doDisplay with an argument of 99 which is a number rather than a string. However, the function would work correctly because the value 99 would be converted into a string when the alert function displayed it.

```
function doDisplay(message="empty"){
    alert(message);
}
```

The above version of doDisplay sets a default value for the message parameter. If the argument is missed off the call of doDisplay the parameter will be set to the string “empty”.

```
doDisplay();
```

This would display the message “empty” in an alert.

# Physical

# Primitive

The JavaScript language provides eight different data types for holding different kinds of value. Seven of the types, including Number, Boolean and String are defined as “primitive”. You can’t add properties to a primitive type, it just holds a single value. If you want a variable that holds multiple values, for example a coordinate that contains values of x and y, you must use an ***object***. Objects are managed by ***reference***.

# Procedure

A procedure is a function that doesn’t return a value. If a program tries to use the value returned from a procedure call it will be given the value undefined.

# Program

A *program* is a sequence of instructions you give to a ***computer*** you tell it how to perform a specific task. You could write a program to add two numbers together and display the result.

# Property

A JavaScript object can contain property values. You can see an example of the creation of a JavaScript object and the addition of properties in the glossary description of Object. Have a read of that, and then come back here for more things about properties.

Hello again. Consider that we have an object that describes a car. It has color, model and make properties. You can delete a property from an object by using the delete operator:

```
delete car.model;
```

This statement would delete the model property from an object referred to by the variable car. Properties get a lot more interesting when we start to use the “braces (or square brackets)” method of accessing them in an object.

```
car["model"] = "Cube"
```

The statement above would restore the “model” property to the object referred to by car. The “dot” mechanism and the “braces” mechanism both generate exactly the same property. If you use the braces mechanism to add properties you can have property names that contain spaces, but you will have to use braces to read the property back again.

# Proprietary

A proprietary technology is one owned and promoted by an organization or company, usually so that they can maintain control of all or part of a market.

# Recursion

See Recursion.

# Reference

A reference is a variable that refers to an object. Variables which are objects are managed by reference, whereas all other types of variable are managed by value.

# Return

A JavaScript function can return use the return keyword to return a value or to just return early.

# Sandbox

# Scope

# Server

# Software

Software is a collection of program code, images, sound files and other data that has been created to provide a solution to a problem. It can be as simple as a single program, or it can be as complex as an operating system.

# Statement

# String

# System Software

System software is software that provides a service to other software. The most obvious piece of system software that you own is probably the operating system of your computer. Other pieces of system software include things like drivers for your graphics card or printer.

# This

# Var

# Variable

# Virtual

# Undeclared

An undeclared variable is one that you've not declared. It doesn't exist in your program. You could ask "How can I make one?" but of course the answer is that you don't. That's the whole point. Undeclared variables are a problem because they if JavaScript encounters one it will throw an exception which will stop that thread of execution. Consider the following completely legal JavaScript code:

```
if(age>70) console.log("too old");
```

This code displays a message in the console if the value in the variable age is greater than 70. However, if the variable age has been declared a ReferenceError exception will be thrown, stopping the program. Undeclared sounds a lot like Undefined (see below) but is actually quite different. An undeclared variable doesn't exist. An undefined variable exists, but it is set to the value undefined which means that it has not been given a value.

# Undefined

If you create a variable but don't put anything in it that value is set to "undefined".

```
var x;
console.log(x);
```

If you perform the above statements in the Developer Tools Console it will display the value "undefined". JavaScript regards undefined as a value that you can assign and test for.

```
x = undefined;
if(x==undefined) console.log("x has not been defined")
```

If you want to mark something as explicitly not set with a value, you can do this by assigning undefined to it. A function can test parameters to make sure that they are not undefined. If a program attempts to use an undefined value in a numeric expression the result of the expression is the special value "not a number" or NaN.

# Variable

# Web Server

# Web Browser

# World Wide Web