Name: _____

JavaScript to Swift – Transition Guide

This booklet is designed to help you manage the transition between the syntax for JavaScript, used for Year 9 IST, and Swift 4, used for Year 10 IST.

Declaring variables

Here are declarations for some common data/object types in **JavaScript:**

```
var myInteger;
var myFloat;
var myString;
var myArrayOfStrings;
```

Here are declarations for some common data/object types in **Swift:**

```
var myInteger: Int?
var myFloat: Float?
var myString: String?
var myArrayOfStrings: Array<Int?>
```

Don't worry about the ? yet. They are called optionals and will come to that later.

Differences to observe:

- JavaScript is a *weakly* typed language, meaning that you do not need to specify the type of variable at the time of declaration. On the other hand, Swift is a *strongly* typed language so you do need to tell each variable it's data type – unless you are providing it will an initial value, in which case you do not need to specify the type:

```
var myInteger = 1
var myFloat = 1.5
var myString = "Hello World"
var myArray = ["String 1", "String 2", "String 3"];
```

- In JavaScript, all variables are variables declared with the `var` keyword. In Swift, if you have a variable but its value never changes in the program it should instead be declared as a constant with the `let` keyword. For example, if you specify the GST rate for Australia – that should probably be declared as a constant since it shouldn't be changed within the program:

```
let gstRate = 0.1
```

`let` constants can be used exactly the same as variables. The only difference is the program will not compile if you try to change a `let` constant within your code. This is a good safety measure.

Using variables

Here is an example of using the aforementioned variables in **JavaScript:**

```
myInteger = 5;
myFloat = 5.8;
myString = "Hello World";
myArray = ["String 1", "String 2", "String 3"];
```

The equivalent in **Swift:**

```
myInteger = 5
myFloat = 5.8
myString = "Hello World"
myArray = ["String 1", "String 2", "String 3"]
```

Differences to observe:

- Not much – they are pretty much identical, except in JavaScript you need to terminate each line with a semicolon, but you do not in Swift.

Selection (if) statements – binary selection

Binary (if-else) selection in **JavaScript:**

```
var myInteger = 5;

if (myInteger < 5) {
   console.log("My integer is less than 5");
}

else {
   console.log ("My integer is greater than 5");
}
```

The equivalent in **Swift:**

```
var myInteger = 5

if myInteger < 5 {
   print("My integer is less than 5")
}

else {
   print("My integer is greater than 5")
}
```

The two languages are syntactically very similar, with a few slight differences:

- In Swift, you do not put parenthesis around the conditions – unless you have multiple conditions which you are ANDing && or ORing || together.

- To output debug messages in Swift you use the in-built `print()` function. This will print the output to the console window in Xcode. Just like in JavaScript, you cannot use the print() function for user-facing output, it is purely for use by you as the developer for debugging purposes.

Selection (if) statement – multiway selection

Multiway (if-elseif-else) selection in **JavaScript:**

```
var myInteger = 5;

if (myInteger < 5) {
   console.log("My integer is less than 5");
}

else if (myInteger < 10) {
   console.log("My integer is less than 10");
}

else {
   console.log("My integer is greater than 10");
}
```

The equivalent in **Swift:**

```
var myInteger = 5

if myInteger < 5 {
   print("My integer is less than 5")
}

else if myInteger < 10 {
   print("My integer is less than 10")
}

else {
   print("My integer is greater than 10")
}
```

Repetition statements – counted loops

Counted loop in **JavaScript:**

```
var myInteger = 5;

for (i = 0; i <= myInteger; i++) {
   console.log(i);
}
```

The equivalent in **Swift:**

```
var myInteger = 5

for i in 0...myInteger {
   print(i)
}
```

You should always use a counted loop when possible. Many students have had bugs in their programs due to using a while loop and then forgetting to do something like increment the loop counter – if you use a counted loop then you won't make this mistake, as everything is built into the loop construct and it auto-increments in Swift.

Repetition statements – pre-test loops

Pre-test loop in **JavaScript:**

```
var i = 0;

while (i <= 5) {
   console.log(i);
   i = i + 1;
}
```

The equivalent in **Swift:**

```
var i = 0

while i <= 5 {
   print(i)
   i = i + 1
}
```

Repetition statements – post-test loops

Post-test loop in **JavaScript:**

```
var i = 0;

do {
   console.log(i);
   i = i + 1;
} while (i <= 5)
```

The equivalent in **Swift:**

```
var i = 0;

repeat {
   print(i)
   i = i + 1
} while i <= 5
```

Custom Functions – no parameters, no return

Custom function in **JavaScript:**

```
function myCustomFunction() {

   // Do stuff here

}
```

Calling the function (from the same file) in **JavaScript:**

```
myCustomFunction();
```

The equivalent in **Swift:**

```
func myCustomFunction() {

   // Do stuff here

}
```

Calling the function (from the same file/class) in **Swift:**

```
myCustomFunction()
```

Custom Functions – with parameters, no return

Custom function in **JavaScript:**

```
function myCustomFunction(paramOne, paramTwo) {

   // Do stuff here

}
```

Calling the function (from the same file) in **JavaScript:**

```
myCustomFunction("Hello", "World");
```

The equivalent in **Swift:**

```
func myCustomFunction(paramOne, paramTwo) {

   // Do stuff here

}
```

Calling the function (from the same file/class) in **Swift:**

```
myCustomFunction("Hello", "World")
```

Custom Functions – with parameters, with return

Custom function in **JavaScript:**

```
function myCustomFunction(paramOne) {

   return 1;

}
```

Calling the method (from the same file) in **JavaScript:**

```
var returnValue = myCustomFunction("Hello");
```

The equivalent in **Swift:**

```
func myCustomFunction(_ paramOne: String) -> Int {

   return 1;

}
```

Calling the method (from the same file/class) in **Swift:**

```
var returnValue = myCustomFunction("Hello")
```

Swift also has the concept of named/labelled parameters in functions, so that you can give each parameter a label. This helps identify the purpose of the arguments when they are passed into the function call. For example, here is myCustomFunction rewritten with a parameter name/label.

```
func myCustomFunction(inputText paramOne: String) -> Int {

   return 1;

}
```

Calling the method (from the same file/class) in **Swift:**

```
var returnValue = myCustomFunction(inputText: "Hello")
```

If you do not want to use parameter names/labels, use an underscore _ character (as per the first example above) instead of the name/label to indicate that you explicitly do not want to use a name or label.