Control Structures & Functions

User Interfaces

420-WC4-AB

Conditional Statements

```
if (condition is true) {
  ...
}
```

```
if (condition is true) {
  ...
} else {
  ...
}
```

```
if (condition is true) {
  ...
} elseif (condition is true) {
  ...
}
```

```
if (condition is true) {
  ...
} elseif (condition is true) {
  ...
} else {
  ...
}
```

Conditional Statements

```
if(userName != "John"){
    console.log("You are an imposter!");
} else {
    console.log("Welcome back John");
}

if(isNaN(age)) {
    // variable age is not a number!!
    console.log("You are not good with numbers");
}
```

Conditional Statements

```
if(!age || isNaN(age)) {
    msg = "You're too old to tell us your age?";
} else if (age <= 18 || age >= 65) {
    msg = "Not working I see!";
} else if (age > 45) {
    msg = "You should have lied about your age.";
} else if (age < 25) {
    msg = "You're too young to rent a car";
} else {
    msg = "There's nothing wrong with being average.";
}
```

Loops – While

- A while statement will execute a block of code as long as it’s expression is true. It is possible that the block of code is never executed based on the expression

```
while (condition is true) {
    ...
}
```

Loops – While

```
let number = 0;
while (number <= 10) {
   console.log(number);
   number = number + 2;
}
// 0, 2, 4, 6, 8, 10
// each value would appear on a new line
```

Loops –Do While

- A do while statement will execute a block of code as long as it’s expression is true. Unlike the white statement it will be execute at least once

```
do {
    ...
} while (condition is true);
```

Loops –Do While

```
let age;
do {
   age = prompt("How old are you?");
} while (!age || isNaN(age));

// keeps asking until user enters a number
// not ideal because "cancel" is ignored
```

Loops - for

- The for loop allows greater control for loop arrays arrays. You can specify an initial condition, a test condition (to end the loop), and a means of changing a counter variable for each pass through the loop, all in one statement.

```
for (startingValue; condition; iterationValue){
    ...
}
```

- The for in loop should used with objects, it automatically goes to the next element when the loop completes. The loop will occur once for every *element* of the *object*

```
for (element in object){
    ...
}
```

Loops - for

```
let count = 2
for(let i=0; i < count; i++) {
   // do something
  console.log("Current: " + i);
}
// Current: 0
// Current: 1
```

Work Out

Switch Statement

- A switch statement tests a value and can have many case statements which define various possible values.
- Statements are executed from the first matched case value until a break is encountered

```
switch (expression){
      case constant1:
            // stuff to do
            break;
      case constant2:
            // stuff to do
            break;
      case constant3:
            // stuff to do
            break;
      default:
            // stuff to do
}
```

Switch Statement

```
switch (dayOfTheWeek){
      case 2:
            console.log("Today is Monday!");
            break;
      case 3:
            console.log("Today is Tuesday!");
            break;
      case 4:
            console.log("Today is Wednesday!");
            break;
      case 5:
            console.log("Today is Thursday!");
            break;
      case 6:
            console.log("Today is Friday!");
            break;
      case 7:
            console.log("Today is Saturday!");
            break;
      default:
            console.log("Today is Sunday! Whoo-weee yo");
}
```

break vs. continue

- **Break**
  - Terminates the current loop, switch
- **Continue**
  - Terminates execution of the statements in the current iteration of the current loop, and continues execution of the loop with the next iteration.

Functions

Functions

- A function is a group of reusable code which can be called anywhere in your program. They eliminate the need of writing the same code over and over again.
- We use function so that we can divide up our code into reusable parts.

```
function myFunctionName() {
   console.log("Hello World");
}
```

- You can call or *invoke* this function by using its name followed by parentheses, like this:

```
myFunctionName();
```

- Each time the function is called it will execute all the code between the curly brackets

Function Documentation

```
/**
 * Logs "Hello World" to the console.
 */
function myFunctionName() {
   console.log("Hello World");
}
```

Function Scope

- Within the body of a function, a local variable takes precedence over a global variable with the same name.
- If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable.

```
let myVar = "global"; // Declare a global variable

function checkscope( ) {

    let myVar = "local";
    // let used to declare local variable
    // Without let, the global variable is used

    console.log(myVar);  // local

}
```

Parameters

- Parameters are variables that act as placeholders for the values.
- When a function is created, it can have one or more parameters separated by commas. It can also have no parameters.
- The actual values that are sent (or "passed") and we want the function to process.

```
function saySomething(param) {
   console.log("You said: " + param);
}
```

- To call this function we must include the 1 parameters required by the function

```
saySomething("JavaScript is not Java");
```

Parameter Documentation

```
/**
 * Log what the user has passed to the console.
 * @param  {*} words Text to be logged.
 */
function saySomething(words) {
   console.log("You said: " + words);
}
```

WORK IT OUT

Multiple Parameters

- Multiple parameters are separated by comas

```
/**
 * Adds 3 parameters provided together.
 * @param  {number} numA Number to add.
 * @param  {number} numB Number to add.
 * @param  {number} numC Number to add.
 */
function addNumbers(numA, numB, numC) {
  let tmp = numA + numB + numC;
}
```

Optional Parameters

- A default function parameter allows for a parameter to be initialized with a default values if no value (or *undefined*) is passed.

```
function addNumbers(numA, numB, numC = 0) {
  let tmp = numA + numB + numC;
}
```

- Default values should be set to the last parameters (one or more), so that you can omit setting the parameter if that parameter is the same as the default value

```
addNumbers(10,20,30); // 60
addNumbers(10,20);    // 30
```

Optional Parameter Documentation

- Multiple parameters are separated by comas

```
/**
 * Adds 3 parameters provided together.
 * @param  {number} numA - Number to add.
 * @param  {number} numB - Number to add.
 * @param  {number} [numC=0] - Number to add.
 */
function addNumbers(numA, numB, numC = 0) {
  let tmp = numA + numB + numC;
}
```

Return Values

- The **return** keyword returns a value to whoever calls the function
- A function can include the return statement but it doesn't have to!

```
function saySomething(param) {
    return ("You said: " + param);
}
let tmp = saySomething("JavaScript is not Java");
console.log( saySomething("JavaScript is not Java") );
```

- Note that as soon as a function encounters return statement the function stops and returns to where it was called.

Return Documentation

```
/**
 * Log what the user has passed to the console.
 * @param  {*} words - Text to be logged.
 * @return {string} String with what you said.
 */
function saySomething(words) {
    return "You said: " + words;
}
```

```
/**
 * Adds 3 parameters provided together.
 * @param  {number} numA - Number to add.
 * @param  {number} numB - Number to add.
 * @param  {number} [numC=0] - Number to add.
 * @return {number} The result of adding 3 numbers.
 */
function addNumbers(numA, numB, numC = 0) {
    let tmp = numA + numB + numC;
    return tmp;
    console.log("All done"); // this will never be executed
}
```

Work Out

Questions?

"No man really becomes a fool until he stops asking questions." – Charles Steinmetz