Day 2: Conditionals, Loops, Basic Functions

Andrew Luo

December 12, 2021

Variable Names

- names cannot start with a digit
  - ex. 95Andrew is illegal variable name
- may include dollar signs ($) and underscores (_) but no other special characters
- cannot include keywords
  - ex. let, var, const
- usually use camelCase by convention
  - first letter is lowercase and all others are uppercase
  - ex. andrewLuoIsCool

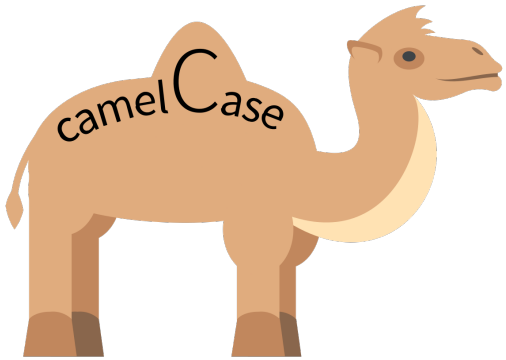

Figure: camelCase makes variable names look like the shape of a camel's back

String Escape Characters

- create a line break with \n
- ```
  console.log("Hello \n World");
  // outputs -> Hello
  //            World
  ```
- create a tab with \t
  ```

Template Literal

- use ` ` backticks, then you can insert expressions into strings using ${}
- ex.

```
let myName = "Bob";
console.log(`Hello ${myName}`);
// will output -> Hello Bob
```

Comparison Operators

- an *expression* is a statement that takes in multiple values and produces a new value
- comparison expressions produce either true or false
- $>$
- $<$
- $>=$
- $<=$
- $==$
  - only checks if values are the same
  - ex. $5 == "5"$ will return true
- $===$
  - checks if both value and types are the same
  - ex. $5 === "5"$ will return false

Comparing Strings

- strings can be compared
- the order is roughly alphabetic
- lowercase always comes before uppercase
- compares each letter one by one
- ex. happy comes before Happy, but after hapoy

Type Coercion

- when comparing two things of different type, Javascript converts them into the same type
- NaN
  - Not a Number
- you can also test if a variable actually holds a value
- ex.

```
let myVariable;
console.log(myVariable == undefined);
// returns true because myVariable has not
    been initialized
```

Logical Operators

- &&
  - "and"
  - evaluates to true if both operands are true
- ||
  - "or"
  - evaluates to true if one of the operands is true
- !
  - "not"
  - evaluates to true if the operand is false

Short Circuit Evaluation

- for ||,if the first operand is true, then the second operand will not be evaluated
- for $$,if the first operand is false, then the second operand will not be evaluated
- use this behaviour to create a fallback to default value
  - "console.log(null |"Hello")"
  - will fall back to "Hello"
  - 0, NaN,undefined,null and empty string "" will evaluate to false

Unary, Binary, Ternary Operators

- an *operand* is a variable which is taken by an operation
- unary operators only have 1 operand
  - ex. typeof operator
  - ```
    let myVarible = "Andrew";
    console.log(typeof myVariable);
    // will output -> string
    ```
- ternary operators have 3 operands
  - ```
    condition ? if true : if false
    ```

Conditional Execution

- if, else statements
- form:

```
if (expression) {
//executes if true
}
else {
//executes if false
}
```

- if, else if statements
- form:

```
if (expression1) {
//executes if expression is true
}
else if (expression2) {
//executes if expression 1 is false and
    expression 2 is true
}
else {executuetes if 1 and 2 are false}
```

While Loop

- continues executing until expression is false
- you need to put an updating statement in the body that eventually makes the expression false
- failure to do so leads to an *infinite loop*
- ex.

```
let i = 0;
while (i < 10)
{
  console.log("Hello World");
  i = i + 1;
}
// how many times does this output hello
    world?
```

Do While Loop

- same as while loop but the loop executes at least once, updating statement happens at the end of the loop body
- ex.

```
let i = 0;
do {
  console.log("Hello");
  i = i + 1;
}
while (i < 0);
// still prints Hello once
```

- notice how you want to indent your code for easy readability

For Loop

- useful when you know how many times you want to iterate (repeat)
- ```
  for (let i = 0; i < 10; i++){
    console.log("Hello World");
  }
  // prints "Hello World" 9 times
  ```
  ```

Incrementing and Decrementing

- i = i + 1 is known as *incrementing* the variable i
  - incrementing is increasing the numeric value by 1
  - decrementing is decreasing the numeric value by 1
- there are shortcuts for incrementing and decrementing
- i += 1
- i++
- i -= 1
- i - -

Functions

- the purpose of functions are to keep your code DRY ("Don't Repeat Yourself")
- code snippets that are reusable through code
- there are two ways of making functions
  - Old Way
  - ```
    function myFunction() {
    // function body
    }
    ```
  - New Way: ES6 Functions
  - ```
    let myFunction = () => {
    //function body
    }
    ```

Parameters and Arguments

- you can pass values to a function using*arguments* when you invoke the function
- the function receives the arguments as *parameters*
- you need to specify what the function receives between the parentheses of the declaration
- ex.

```
function hello(name) {
  console.log(`Hello ${name}`);
}

//later I invoke the function using ()
hello("Andrew");
// prints "Hello Andrew"
```

Return Value

- when the function is invoked, it can return a value
- ex.

```
function hello(name) {
  return `Hello ${name}`;
}
//later I invoke the function, but I then
    can store the return value into another
     variable

let returnValue = hello("Andrew");
console.log(returnValue);
// prints "Hello Andrew"
```