Control Flow + More Data Types

Review: Terminology

In the following code, name the data types used, describe the scope for the variables, and spot

the: `statement`, `variable`,`expression`, `operator`, `function`, `argument`, and `return value`.

```
function calculateTip(total) {
  var tipPercent = 0.15;
  return (total * tipPercent);
}

var billTotal = 10.00;
var billTip   = calculateTip(billTotal);
var receipt   = 'Total: ' + billTotal + ' Tip: ' + billTip;
console.log(receipt);
```

The if statement

Use `if` to tell JS which statements to execute, based on a condition.

```
if (condition) {
  // statements to execute
}
```

```
var x = 5;


if (x > 0) {
  console.log('x is a positive number!');
}
```

Comparison Operators

Use these operators to compare two values for equality, inequality, or difference.

```
  var myAge = 28;
```

Operator	Meaning	True expressions

==	Equality	`myAge == 28`
		`myAge == '28'`
		`28 == '28'`

===	Strict equality	`myAge === 28`
!=	Inequality	`myAge != 29`

Operator	Meaning	True expressions
!==	Strict inequality	`myAge != '28'` `28 != '28'`
>	Greater than	`myAge > 25` `'28' > 25`
>=	Greater than or equal	`myAge >= 28` `'28' >= 25`
<	Less than	`myAge < 30` `'28' < 30`
<=	Less than or equal	`myAge <= 28` `'28' <= 28`

Common mistake: Do not confuse = (the assignment operator) with==.

Logical Operators

These are typically used in combination with the comparison operators:

```
var posNum = 4;

var negNum = -2;
```

Operator	Meaning	True expressions

Operator	Meaning	True expressions
&&	AND	`posNum > 0 && negNum < 0` `4 > 0 && -2 < 0`
||	OR	`posNum > 0 || negNum > 0` `4 > 0 || -2 > 0`
!	NOT	`!(posNum === negNum)` `!(posNum < 0)`

When combining together multiple conditions, use parantheses to group:

```
var myAge = 28;
if ((myAge >= 0 && myAge < 3)  || myAge > 90) {
  console.log('You\'re not quite in your peak.');
}
```

Truthy vs Falsey

If you don't use a comparison or logical operator, JS tries to figure out if the value is "truth-y".

```
var catsRule = true;
if (catsRule) {
```

```
  console.log('Yay cats!');
}
```

Values that are "false-y": `false`, the empty string (`""`), the number`0`, `undefined`, `null`.

```
var name = '';
if (name) {
  console.log('Hello, ' + name);
}
var points = 0;
if (points) {
  console.log('You have ' + points + ' points');
}
var firstName;
if (firstName) {
  console.log('Your name is ' + firstName);
}
```

Short-Circuit Evaluation

JS evaluates logical operators from left to right and stops evaluating as soon as it knows the answer.

`(false &&` anything) => false

`(true ||` anything) => true

```
var nominator = 5;
var denominator = 0;
if (denominator != 0 && (nominator/denominator > 0)) {
  console.log('Thats a valid, positive fraction');
}
```

The if/else statement

Use `else` to give JS an alternative statement to execute.

```
var age = 28;
if (age > 16) {
  console.log('Yay, you can drive!');
} else {
  console.log('Sorry, but you have ' + (16 - age) + ' years til you can drive.');
}
```

The if/else if/else statement

You can use `else if` if you have multiple exclusive conditions to check:

```
var age = 20;

if (age >= 35) {
  console.log('You can vote AND hold any place in government!');
} else if (age >= 25) {
  console.log('You can vote AND run for the Senate!');
} else if (age >= 18) {
  console.log('You can vote!');
} else {
  console.log('You have no voice in government!');
}
```

Exercise Time!

The while loop

The `while` loop tells JS to repeat statements until a condition is true:

```
while (expression) {
  // statements to repeat
}
```

```
var x = 0;
while (x < 5) {
  console.log(x);
  x = x + 1;
}
```

Beware: It's easy to accidentally cause an "infinite loop."

The for loop

The `for` loop is another way of repeating statements, more specialized than `while`:

```
for (initialize; condition; update) {
  // statements to repeat
}
```

```
for (var i = 0; i < 5; i = i + 1) {
  console.log(i);
}
```

The break statement

To prematurely exit a loop, use the `break` statement:

```
for (var current = 100; current < 200; current++) {
  console.log('Testing ' + current);
  if (current % 7 == 0) {
    console.log('Found it! ' + current);
    break;
  }
}
```

Exercise Time!

The array data type

An array is a type of data-type that holds an ordered list of values, of any type:

```
var arrayName = [element0, element1, ...];
```

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet'];
var raceWinners = [33, 72, 64];
var myFavoriteThings = ['Broccoli', 60481, 'Love Actually'];
```

The `length` property reports the size of the array:

```
console.log(rainbowColors.length);
```

Array access

You can access items with "bracket notation". The index starts at 0.

```
var arrayItem = arrayName[indexNum];
```

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet'];

var firstColor = rainbowColors[0];

var lastColor = rainbowColors[6];
```

Changing arrays

You can also use bracket notation to change the item in an array:

```
var myFavoriteThings = ['Broccoli', 60481, 'Love Actually'];

myFavoriteThings[0] = 'Celery Root';
```

Or to add to an array:

```
rainbowColors[4] = 'Playgrounds';
```

You can also use the `push` method:

```
rainbowColors.push('Dancing');
```

Iterating through arrays

Use a for loop to easily process each item in an array:

```
var rainbowColors = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet'];

for (var i = 0; i < rainbowColors.length; i++) {

  console.log(rainbowColors[i]);

}
```

Exercise Time!