"number-like strings"

NaN (Not a Number)

Number() method

null

not defined

string concatenation

"number-like strings"

A "number-like string" is a number enclosed in quotes. So, whereas 5 is an actual number, "5" is a "number-like string".

1. Declare a variable with a "number-like string" value, and try to do addition with it:

```
let bill = 50;
let tip = '10';
let total = bill + tip;
console.log(total); // 5010
```

Addition with number-like strings fails, because the plus-sign defaults to concatenation. But for other operations with number-like strings -- subtraction, multiplication, division -- the math works, because there is no plus-sign to confuse things.

2. Declare a number-like string and do division with it:

```
let pizzas = "4";
let people = 8;
let pizzasPP = pizzas / people;
console.log(pizzasPP); // 0.5
```

NaN

NaN (Not a Number) results from trying to do math with something that is neither a number nor a number-like string.

3. Try to do math with a price that includes a dollar sign:

```
let fullPrice = '$80';
let halfPrice = fullPrice * 0.5;
console.log('halfPrice', halfPrice); // NaN number
```

The string '$80' is in no way understood as the number 80, so attemptinh to do math with '$80' fails.

Number() method

The **Number()** method takes a variable as its argument, and where possible, returns a number. It is ideal for converting "number-like strings" into actual numbers.

4. A number in quotes such as "4"` is a "number-like string". Convert it to a number:

```
console.log('pizzas', pizzas, typeof(pizzas));
// pizzas 4 string
pizzas = Number(pizzas);
console.log('pizzas', pizzas, typeof(pizzas));
// pizzas 4 number
```

If the string passed to it cannot be converted to a number, the **Number()** method returns **NaN**.

5. Try to convert "banana" to a number:

```
let fruit = "banana";
let baNaNa = Number(fruit);
console.log('baNaNa', baNaNa, typeof(baNaNa));
// baNaNa NaN number
```

Despite its name, **NaN** has a data type of number.

"Number-like strings" can't be used for addition, but you can convert them with the Number() method.

6. Convert "15" to an actual number, so that it can be used for addition:

```
bill = 70;
tip = '15';
total = bill + tip;
console.log(total); // 7015

total = bill + Number(tip);
console.log(total); // 85
```

not defined vs undefined

not defined means the variable does not exist. It is an error that usually arises from typos. **undefined** means that the variable exists, but has no value.

7. To see the difference between *undefined* and *not defined*, declare a variable with no value, and then misspell it:

```
    let island;
    console.log(island, typeof(island)); // undefined undefined
    island = "Bali";
    console.log(island, typeof(island)); // Bali string
    // console.log(ixland); // error: ixland is not defined
```

null

null and **undefined** are both *falsey* (return false in a boolean context), but null is an actual value assigned to a variable. It has a data type of object, but it's just that null is an *empty* object.

8. Declare a variable, set it to null, and log it:

```
    let user = null;
    console.log('user', user, typeof(user));
    // user null object
```

string concatenation

Variables and substrings can be joined together with plus-signs (**+**) to make one bigger string. The procedure is known as **string concatenation**.

9. Concatenate the **topic** variable with substrings:

```
    let topic = "JavaScript";
    let intro = "Let's learn " + topic + "!";
    console.log(intro); // Let's learn JavaScript!
```

10. Concatenate with two variables:

```
    let firstName = 'Brian';
    let lastName = 'McClain';
    let greeting = 'Hello, class! My name is ' + fullName + ' ' + lastName
+ '.';
    console.log(greeting);
    // Hello, class! My name is Brian McClain.
```

Concatenation is often used for making multiple versions of similar strings, such as a set of image paths that differ only slightly by file name.

11. Concatenate an image file path consisting of two variables and three substrings:

```
let kind = 'Jack';
let suit = 'Hearts';
let imgPath = 'images/' + kind + '-of-' + suit + '.jpg';
console.log(imgPath); // images/Jack-of-Hearts.jpg
```

12. Change the variable values to get new image paths:

```
kind = 'Queen';
suit = 'Diamonds';
imgPath = 'images/' + kind + '-of-' + suit + '.jpg';
console.log(imgPath); // images/Queen-of-Diamonds.jpg

kind = 'King';
suit = 'Clubs';
imgPath = 'images/' + kind + '-of-' + suit + '.jpg';
console.log(imgPath); // images/King-of-Clubs.jpg

kind = 'Ace';
suit = 'Spades';
imgPath = 'images/' + kind + '-of-' + suit + '.jpg';
console.log(imgPath); // images/Ace-of-Spades.jpg
```

A plus-sign performs concatenation rather than addition if the expression includes a string.

13. Try adding numbers with a dollar-sign present. It reverts to string concatenation:

```
let food = 25;
let bev = 15;
let tip = 8;
let tot = '$' + food + bev + tip;
console.log(tot, typeof(tot)); // $25158 string
```

14. Remove the dollar sign, and run it again; this time the math works. *After* the math is done, put back the **$**:

```
tot = food + bev + tip;
console.log(tot); // 48
console.log(tot, typeof(tot)); // 48 number
tot = '$' + tot;
console.log(tot, typeof(tot)); // $48 string
```

changing a variables's data type
In the above example, the number **tot** is changed into a string. Changing the datatype of a variable *is* permitted, but it should be done sparingly.

16. Change a three digit number into a four-digit PIN by concatenating a leading zero. This result is a string, but the reason for changing datatypes does make sense:

```
let num = 582;
console.log(num, typeof(num)); // 582 number
let pin = "0" + num;
console.log(pin, typeof(pin)); // 0582 string
```

- **END Lesson 01.02**
- **NEXT: Lab 01.02**
- **Lesson 01.03**