



# Strings

## "STRINGS OF CHARACTERS"

Strings are another primitive type in JavaScript. They represent text, and must be wrapped in quotes.

# STRINGS



```
let firstName = "Ziggy";      Double quotes work  
  
let msg = "Please do not feed the chimps!";  
  
let animal = 'Dumbo Octopus';  So do single quotes  
  
let bad = "this is wrong";    This DOES NOT work
```

It's fine to use either single or double quotes, just be consistent in your codebase.

# STRINGS ARE INDEXED

C	H	I	C	K	E	N
0	1	2	3	4	5	6

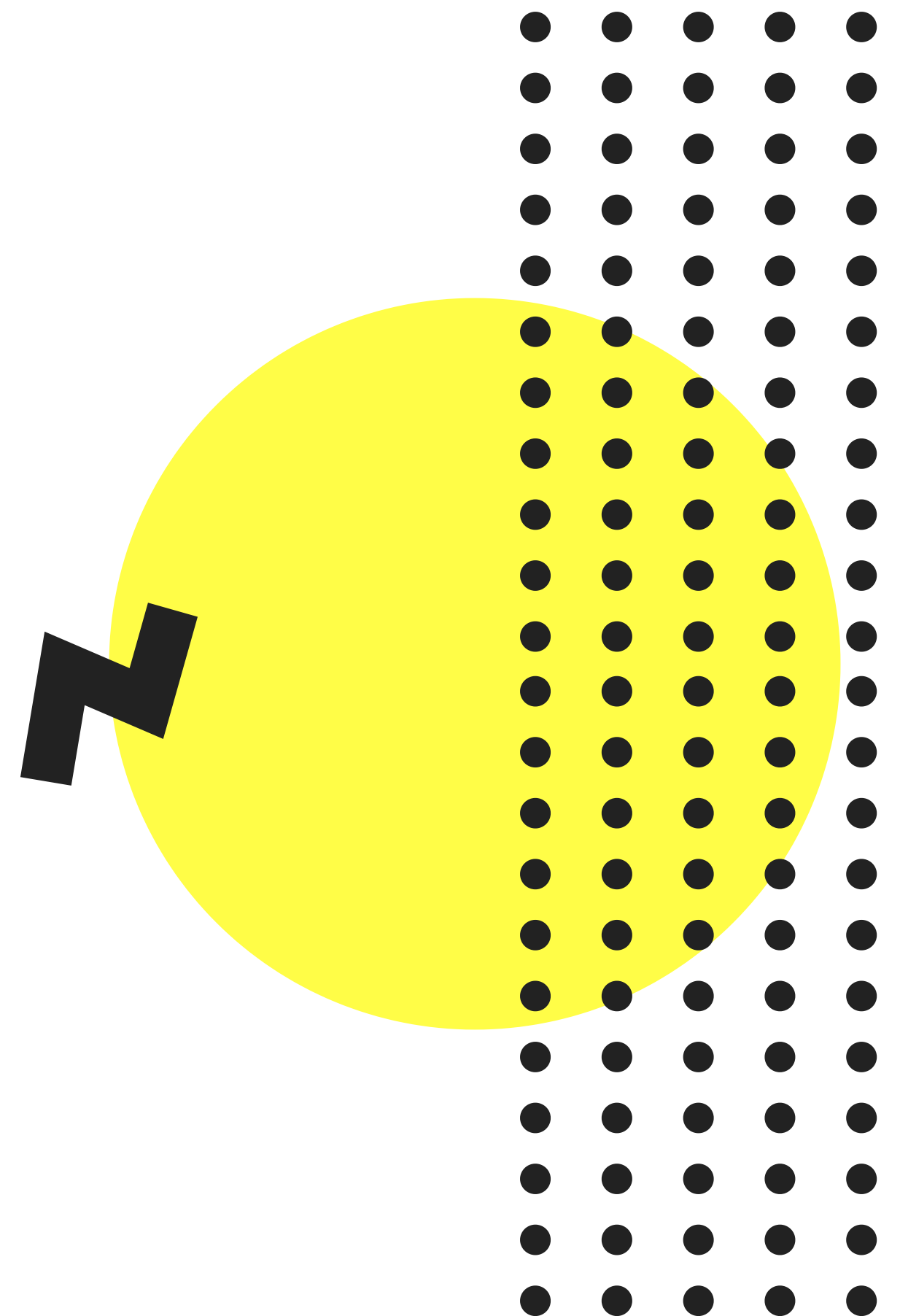
Each character has a corresponding index  
(a positional number)

# String Methods

**METHODS ARE BUILT-IN  
ACTIONS WE CAN PERFORM  
WITH INDIVIDUAL STRINGS**

They help us do things like:

- Searching within a string
- Replacing part of a string
- Changing the casing of a string



thing.method()

# Casing



```
let msg = 'I am king';  
let yellMsg = msg.toUpperCase(); // 'I AM KING'
```

```
let angry = 'LeAvE mE aLoNe!';  
angry.toLowerCase(); // 'leave me alone!'
```

```
//the value in angry is unchanged  
angry; // 'LeAvE mE aLoNe!'
```

# Trim



```
let greeting = '  leave me alone plz  ';  
greeting.trim() // 'leave me alone plz'
```

# thing.method(arg)

Some methods accept **arguments** that modify their behavior.

Think of them as inputs that we can pass in.

We pass these arguments inside of the parentheses.



# indexOf



```
let tvShow = 'catdog';  
  
tvShow.indexOf( 'cat' ); // 0  
tvShow.indexOf( 'dog' ); // 3  
tvShow.indexOf( 'z' ); // -1 (not found)
```

# slice



```
let str = 'supercalifragilisticexpialidocious'  
  
str.slice(0,5); //'super'  
  
str.slice(5); // 'califragilisticexpialidocious'
```

# replace

```
let annoyingLaugh = 'teehee so funny! teehee!';  
  
annoyingLaugh.replace('teehee', 'haha') // 'haha so funny! teehee!'  
//Notice that it only replaces the first instance
```



# WHAT IS THE VALUE OF *AGE*?



```
const age = "5" + "4";
```



# WHAT DOES THIS EVALUATE TO?



```
"pecan pie"[7]
```

# WHAT DOES THIS EVALUATE TO?



```
"PUP"[3];
```




# What is the value of *song*?



```
let song = "london calling";  
song.toUpperCase();
```



# What is the value of *cleanedInput*?




```
let userInput = "  TODD@gmail.com";  
let cleanedInput = userInput.trim().toLowerCase();
```





# What is the value of *index*?



```
let park = 'Yellowstone';  
const index = park.indexOf( 'Stone' );
```



# What is the value of *index*?



```
let yell = 'GO AWAY!!!';  
let index = yell.indexOf('!');
```



# WHAT DOES THIS EVALUATE TO?



```
'GARBAGE!'.slice(2).replace("B",'');
```

# STRING ESCAPES

- `\n` – newline
- `\'` – single quote
- `\"` – double quote
- `\\` – backslash

# Template Literals

**SUPER USEFUL!**

```
`I counted ${3 + 4} sheep`; // "I counted 7 sheep"
```

TEMPLATE LITERALS ARE STRINGS THAT ALLOW  
EMBEDDED EXPRESSIONS, WHICH WILL BE EVALUATED  
AND THEN TURNED INTO A RESULTING STRING

**WE USE BACK-TICKS  
NOT SINGLE QUOTES**

**`I am a template literal`**

\* The back-tick key is usually above the tab key

# TEMPLATE LITERALS



```
let item = 'cucumbers';  
let price = 1.99;  
let quantity = 4;
```

```
`You bought ${quantity} ${item}, total price: ${price*quantity}`;  
// "You bought 4 cucumbers, total price: $7.96"
```

# NULL & UNDEFINED

- Null
  - "Intentional absence of any value"
  - Must be assigned
- Undefined
  - Variables that do not have an assigned value are undefined



# NULL

```
1 // No one is logged in yet...
2 let loggedInUser = null; //value is explicitly nothing
3
4 // A user logs in...
5 loggedInUser = 'Alan Rickman';
```

# Undefined

```
1 let pickles; //We didn't assign a value
2 pickles; //undefined,
3 pickles = 'are very gross'
4
5 //Undefined also comes up in other situations:
6 let food = 'tacos';
7 food[7]; //undefined
```

# MATH OBJECT

Contains properties and  
methods for mathematical  
constants and functions



```
Math.PI // 3.141592653589793
```

```
//Rounding a number:
```

```
Math.round(4.9) //5
```

```
//Absolute value:
```

```
Math.abs(-456) //456
```

```
//Raises 2 to the 5th power:
```

```
Math.pow(2,5) //32
```

```
//Removes decimal:
```

```
Math.floor(3.9999) //3
```

# RANDOM NUMBERS

`Math.random()` gives us a random decimal between 0 and 1 (non-inclusive)



```
Math.random();  
//0.14502435424141957  
Math.random();  
//0.8937425043112937  
Math.random();  
//0.9759952148727442
```

# RANDOM INTEGERS

Let's generate random  
numbers between 1 and 10

```
const step1 = Math.random();  
//0.5961104892810127  
const step2 = step1 * 10  
//5.961104892810127  
const step3 = Math.floor(step2);  
//5  
const step4 = step3 + 1;  
//6  
  
Math.floor(Math.random() * 10) + 1;
```

# parseInt & parseFloat

Use to parse strings into numbers, but watch out for NaN!



```
parseInt('24') //24
parseInt('24.987') //24
parseInt('28dayslater') //28

parseFloat('24.987') //24.987
parseFloat('7') //7
parseFloat('i ate 3 shramp') //NaN
```