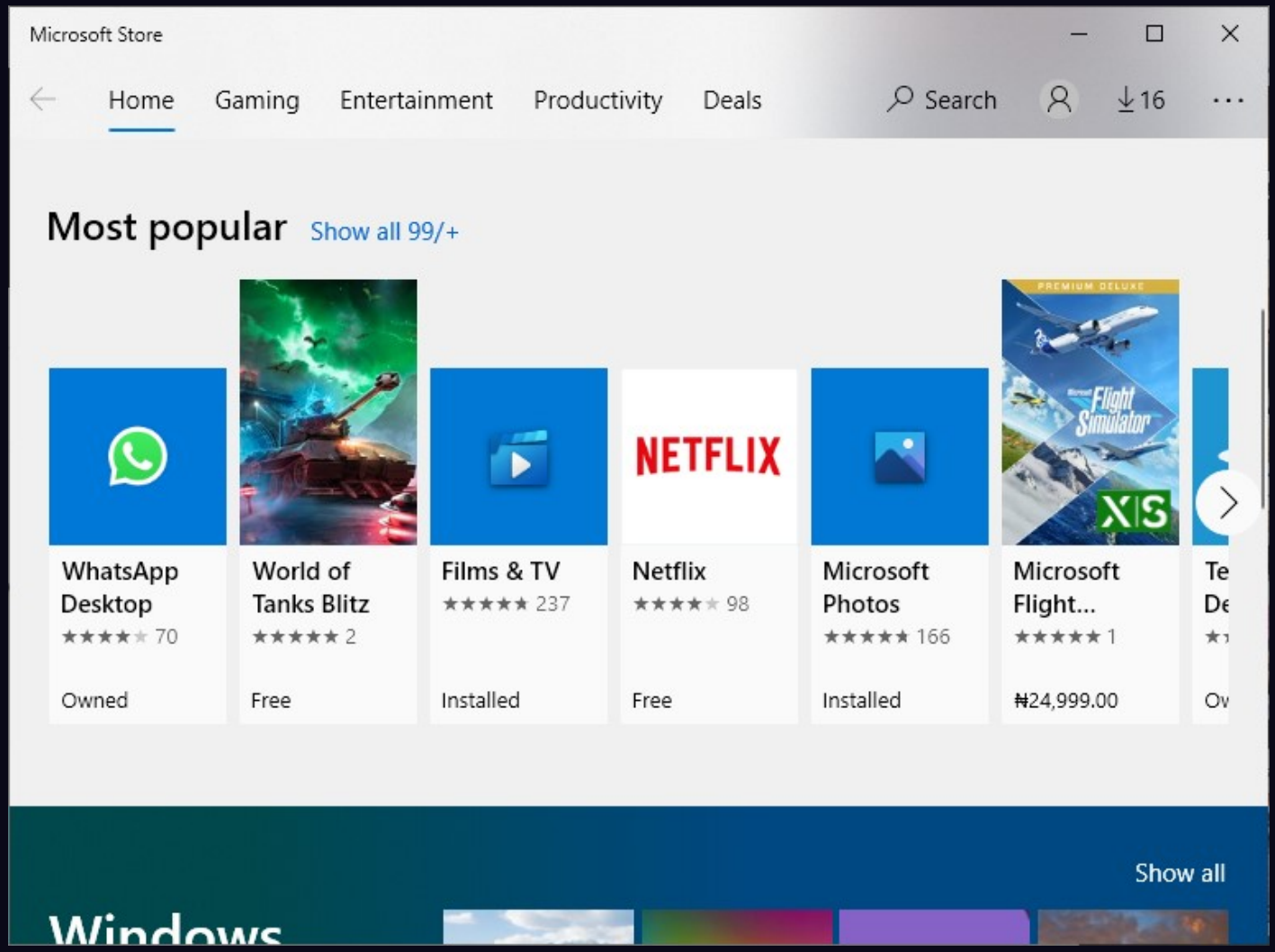
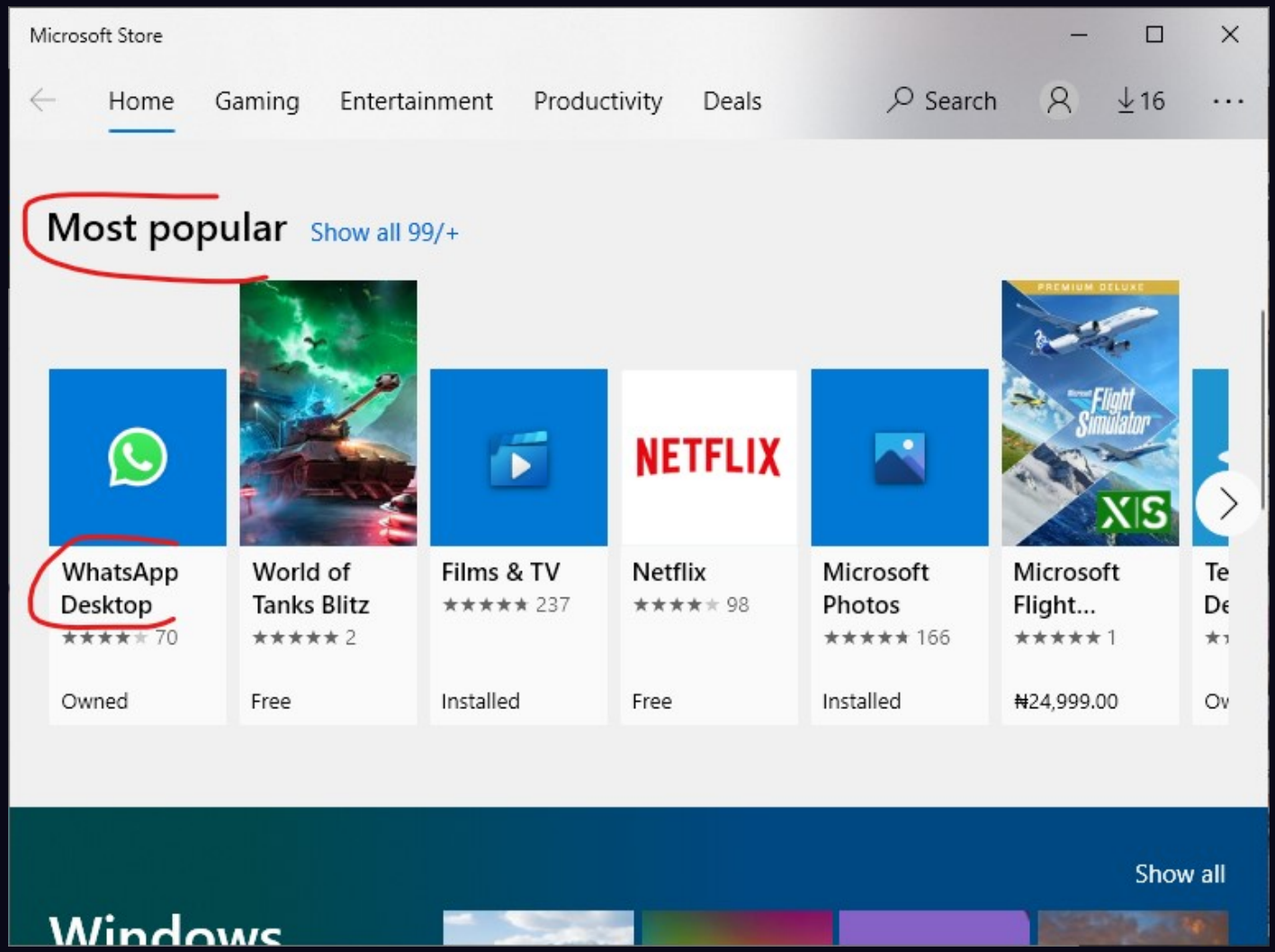
thinkdev #2

Values and Types

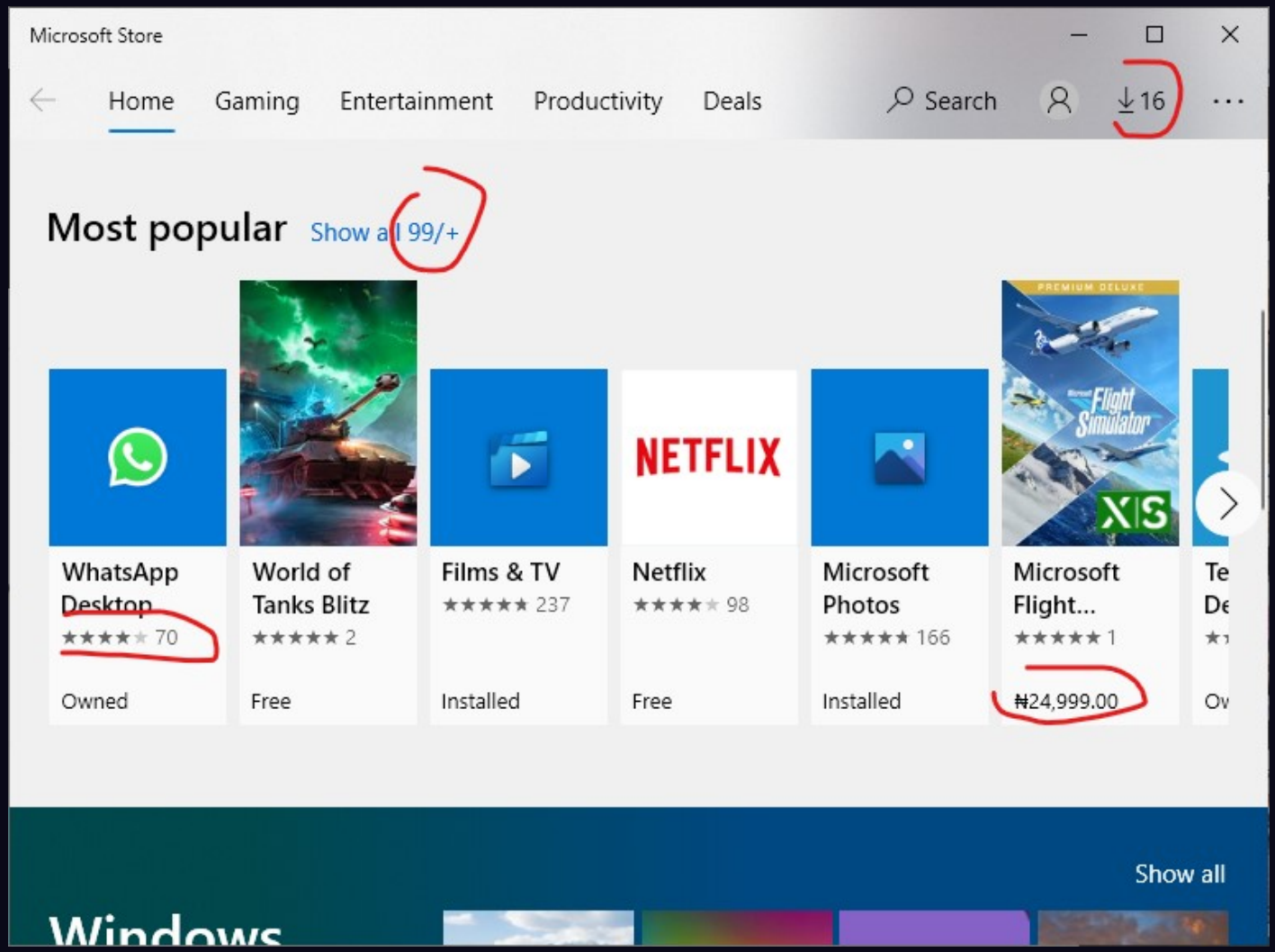


Screenshot of Microsoft Store

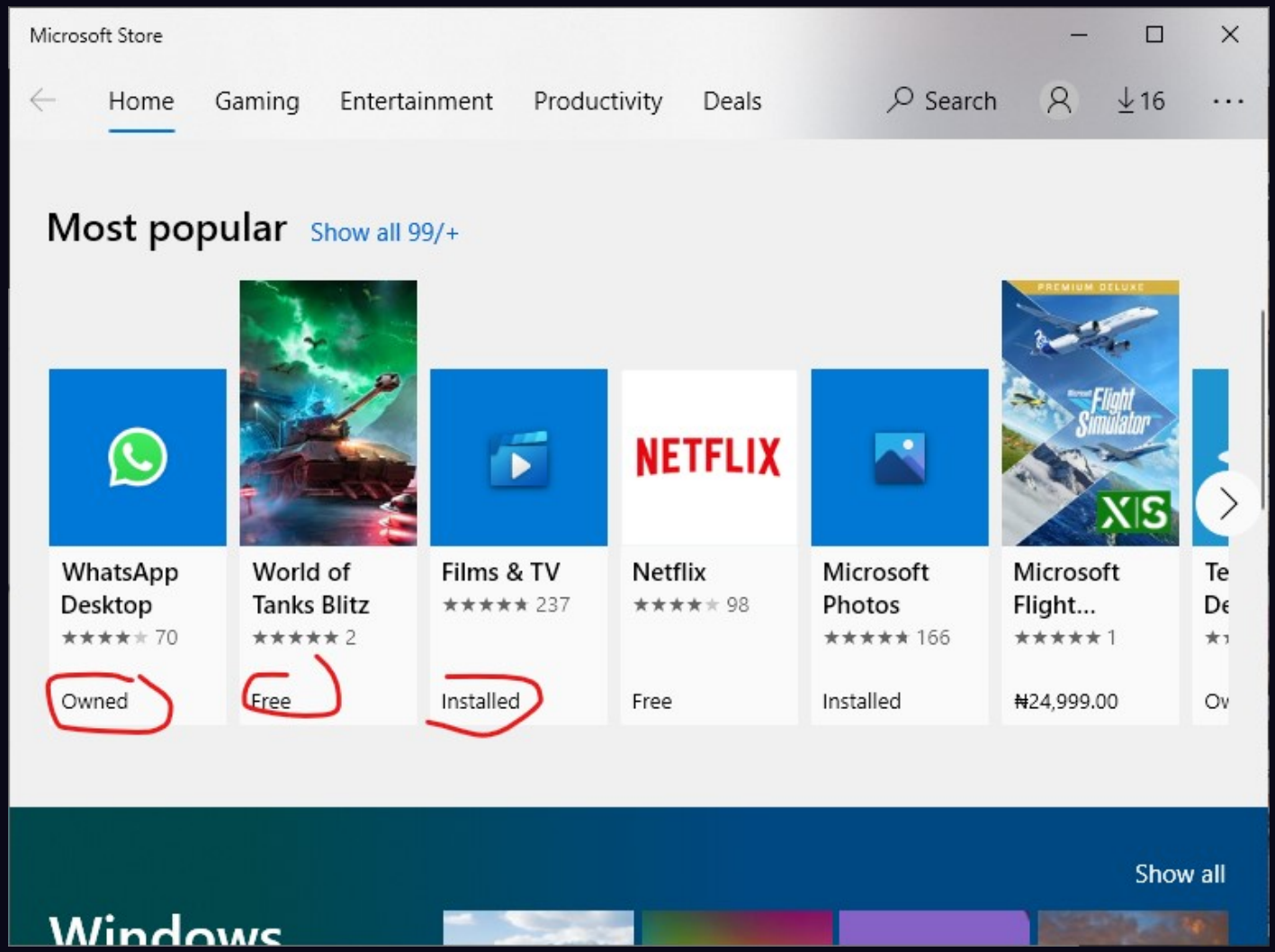
Text


Microsoft Store
Home
Gaming
Entertainment
Productivity
Deals
Search
16
Most popular
Show all 99/+
WhatsApp Desktop
70
Owned
World of Tanks Blitz
2
Free
Films & TV
237
Installed
NETFLIX
Netflix
98
Free
Microsoft Photos
166
Installed
PREMIUM DELUXE
Flight Simulator
XS
Microsoft Flight...
1
₦24,999.00
Show all

Numbers


Microsoft Store
Home
Gaming
Entertainment
Productivity
Deals
Search
16
Most popular
Show all 99/+
WhatsApp Desktop
70
Owned
World of Tanks Blitz
2
Free
Films & TV
237
Installed
NETFLIX
Netflix
98
Free
Microsoft Photos
166
Installed
PREMIUM DELUXE
Flight Simulator
XS
Microsoft Flight...
1
₦24,999.00
Show all
Windows

Alternatives



These have "formal" names

These have "formal" names

- We represent text with *strings*
  - `"Most popular"`
  - `'Installed'`
  - `'She\'s here'`

These have "formal" names

- We represent text with *strings*
  - `"Most popular"`
  - `'Installed'`
  - `'She\'s here'`
- We use *booleans* to choose between alternatives
  - `true`, `false`

What about numbers?

What about numbers?

- Many languages differentiate between types of numbers.

What about numbers?

- Many languages differentiate between types of numbers.
- *Integer* types (or *int*) for 1, 20, -7, ...

What about numbers?

- Many languages differentiate between types of numbers.
- *Integer* types (or *int*) for 1, 20, -7, ...
- *Floating point* types (or *float*) for 3.2, -0.789, ...

But in JavaScript ...

... a number is just a *number*.

typeof

Use `typeof` keyword to get the type of a value:

```
typeof "Hi" // "string"
typeof 12.34 // "number"
typeof 3_000_000 // "number"
typeof false // "boolean"
```

Comments

Messages ignored by the language, just for the developer

Comments

Messages ignored by the language, just for the developer

```
// Line comment

/* Block comment */

/*
Multiline
block
comment
*/
```

Operations on numbers

- Values aren't so useful alone

Operations on numbers

- Values aren't so useful alone
- We can do the usual arithmetics: (+, -, /, *)

Operations on numbers

- Values aren't so useful alone
- We can do the usual arithmetics: (+, -, /, *)
- Modulus operator % (more on this later)

Expressions

Things that have value.

```
"Hi 👋";
20.9;
50 * 70 / 67 + 9;
typeof true;
```

Expressions

You can wrap expressions in brackets.

```
("Hi 👋🏾");
(20.9);
((50 * 70) / (67 + 9));
(typeof true);
```

Expressions

Values and expressions can usually replace each other.

```
typeof ((50 * 70) / (67 + 9));
typeof typeof true
```

What if we wanted to store an expression?

Variables

```
// Declare a variable
const costPerItem = 3000
```

Variables

```
// Declare a variable
const costPerItem = 3000

// Use the variable
console.log(costPerItem * 10)
```

Variables

```
// Declare a variable
const costPerItem = 3000

// Use the variable
console.log(costPerItem * 10)

// The right-hand side is always an expression
const costPer10Items = costPerItem * 10
```

How to name variables?

How to name variables?

- First character must be a letter, underscore _, or dollar sign $.
  - Valid: x, $, _
  - Invalid: 0, 1

How to name variables?

- First character must be a letter, underscore _, or dollar sign $.
  - Valid: `x`, `$`, `_`
  - Invalid: `0`, `1`
- Following characters may include numbers
  - Valid: `y2`, `first_name`, `_LAST_NAME_`, `$10`
  - Invalid: `2a`

How to name variables?

- First character must be a letter, underscore _, or dollar sign $.
  - Valid: `x`, `$`, `_`
  - Invalid: `0`, `1`
- Following characters may include numbers
  - Valid: `y2`, `first_name`, `_LAST_NAME_`, `$10`
  - Invalid: `2a`
- Names are case-sensitive
  - `message`, `Message`, `MESSAGE` are different variables.

How to name variables?

The JavaScript convention is

`camelCase` 🐫

Variables that vary

- `const` variables are *constant*; they always refer to the same value.

Variables that vary

- `const` variables are *constant*; they always refer to the same value.
- That's fine in many cases, but sometimes we need to change that reference.

Variables that vary

Consider an e-commerce app

Screenshot of cart from Tarbiyah Books Plus

Variables that vary

```
const quantity = 0

// When the user clicks the plus button,
// we should increase the quantity
quantity = quantity + 1
// Error: Assignment to constant variable
```

Variables that vary

Use the `let` keyword instead

```
let quantity = 0

quantity = quantity + 1

console.log(quantity) // 1
```

Variables that vary

Addition assignment operator

```
let quantity = 0

quantity += 1

console.log(quantity) // 1
```

Variables that vary

Increment operator

```
let quantity = 0

quantity++

console.log(quantity) // 1
```

Variables that vary

You don't have to initialize a `let` variable with a value immediately.

```
let quantity; // value is undefined

// initialize after declaring
quantity = 0

quantity++

console.log(quantity) // 1
```

Empty values

- `undefined` and `null`.
- No hard rules on when to use which.
- Some use `null` for intentionally empty variables.

Statements

Not everything is an expression; some things don't produce value. E.g., variable declaration

```
// This is wrong! Declaration is not an expression
let a = const b = 10

// But this is valid; assignment is an expression
let a = 10
const b = a = 15
// a and b are now 15
```

Operations on strings

Joining strings

Also known as *concatenation*

```
const firstName = "Mubaraq"
const lastName = "Wahab"

const fullName = firstName + lastName
// "MubaraqWahab"
```

Joining strings

Also known as *concatenation*

```
const firstName = "Mubaraq"
const lastName = "Wahab"

// Better
const fullName = firstName + " " + lastName
// "Mubaraq Wahab"
```

Interpolation

You can use special strings called *template literals* to interpolate.

```
const firstName = "Mubaraq"
const lastName = "Wahab"

const fullName = ${firstName} ${lastName}`
// "Mubaraq Wahab"
```

Get a character from a string

Use square brackets to specify an *index* (starts from zero)

```
//                    0123456
const firstName = "Mubaraq"
const lastName = "Wahab"

const initial = firstName[0] // "M"
const second = firstName[1] // "u"
// and so on...
```

Get part of a string

Use the `slice` method

```
//                    0123456
const firstName = "Mubaraq"
const lastName = "Wahab"

const firstThreeLetters = firstName.slice(0, 3)
// "Mub"
const thirdToEnd = firstName.slice(2)
// "baraq"
```

Is this part of a string?

Use the `includes` method to check if a string includes another.

```
const firstName = "Mubaraq"
const lastName = "Wahab"

firstName.includes('ba')
// true
firstName.includes('ab')
// false
```

How long is a string?

Use the `length` property to get the length of a string.

```
const firstName = "Mubaraq"
const lastName = "Wahab"

firstName.length
// 7
```

String to number

You need to convert a string to a number sometimes, such as when working with user input.

```
// Assume this is from user input
const input = "20"

// Careful here! Result is "203"
input + 3
```

String to number

You need to convert a string to a number sometimes, such as when working with user input.

```
// Assume this is from user input
const input = "20"

// Convert to number first!
const inputAsNumber = Number(input)

// Result is 23
inputAsNumber + 3
```

String to number

You need to convert a string to a number sometimes, such as when working with user input.

```
// Assume this is from user input
const input = "20"

// An idiomatic way
const inputAsNumber = +input

// Result is 23
inputAsNumber + 3
```

Number to string

The other way round works too!

```
const num = 20

// Result is "20"
const numAsString = String(num)
```

Number to string

The other way round works too!

```
const num = 20

// Result is "20"
const numAsString = num.toString()
```

Number to string

The other way round works too!

```
const num = 20

// Result is "20"
const numAsString = "" + num
```

UPPERCASE, lowercase

```
const firstName = "Mubaraq"

firstName.toUpperCase()
// "MUBARAQ"

firstName.toLowerCase()
// "mubaraq"
```

See also

- The MDN Web Docs

Exercise

Check the website

Questions?