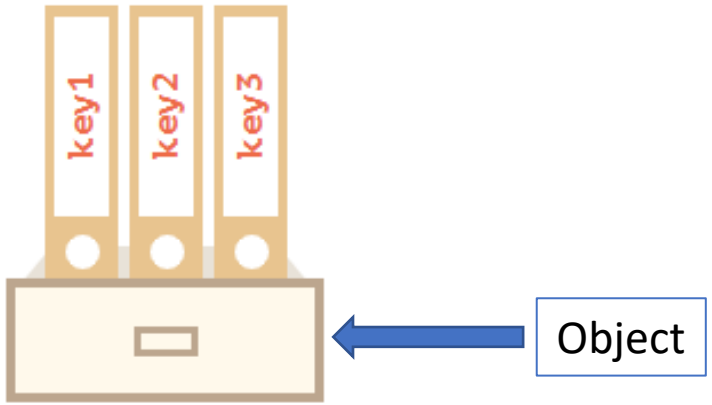
Objects in JavaScript

Objects

- In JavaScript, objects are used to store keyed collections of various data and more complex entities.
- We can imagine an object as a cabinet with labeled files. Every piece of data is stored in its file by the key.
  - It’s easy to find a file by its name or add/remove a file.



Creating an object

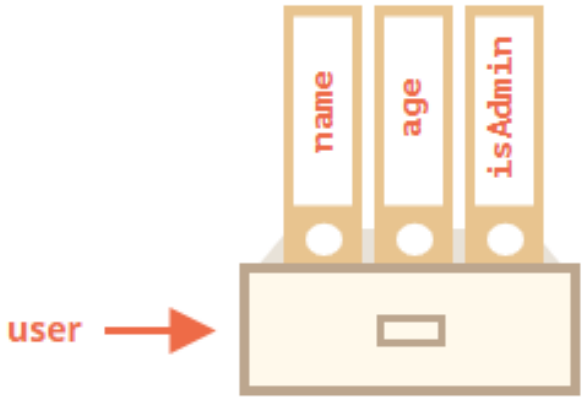
- Simplest way and the usual way to create an object in JavaScript is using the *object literal* syntax { }, a set of curly braces with an optional list of *properties*.

```
let user = {};  // "object literal" syntax
```

Literals and properties

- We can immediately put some properties into `{...}` as “key: value” pairs:

```
let user = {           // an object
    name: "John",      // key "name", value "John"
    age: 30,           // key "age",  value 30
    isAdmin: true,     // key "isAdmin", true
};
```



Multiword property names

- We can also use multiword property names, but then they must be in quotes

```
let user = {
    name: "John",
    age: 30,
    isAdmin: true,
    "loves music": true  // multiword property name must be quoted
};
```

Accessing/ modifying properties

- Property values are accessible using the dot notation.

```
user.name="Bob"; // set property name with value “Bob”
user.age = 50;   // set property age with value 50

alert( user.name ); // John
alert( user.age ); // 50
```

- For multiword properties, the dot access doesn’t work.
  - There’s an alternative “square bracket notation”

```
user["loves music"] = false; // set
alert(user["loves music"]);  // get
```

Add and Remove properties

- In JavaScript, properties of an object can also be added and removed at the runtime.
  - Add syntax is like set syntax, except we use a new property name

```
user.id = 123; // adding a new property id in an exiting user object
```

  - To remove a property, we can use `delete` operator

```
delete user.age;
```

Computed properties

- We can use square brackets in an object literal, when creating an object. That's called *computed properties*.

```
let fruit = prompt("Which fruit to buy?", "apple");

let bag = {
  [fruit]: 5, // the name of the property is taken from the variable fruit
};

alert( bag.apple ); // 5 if fruit="apple"
```

Property value shorthand

- When the property has the same names as variables, there's a special *property value shorthand* to make it shorter.

```
function makeUser(name, age) {
    return {
      name, // same as name: name
      age,  // same as age: age
    };
  }
```

- We can use both normal properties and shorthand's in the same object

```
let age = 30;
let user = {
    name: "Tom",
    age, // same as age: age
};
```

- Always favor clarity over brevity

Property names limitations

- no limitations on property names
  - They can be any strings (even the reserved keywords) or symbols (a special type of identifiers)

- Best to follow same variable naming best practices for property names as well.

Property existence test, “in” operator

- A notable feature of objects in JavaScript, compared to many other languages, is that it’s possible to access any property. There will be no error if the property doesn’t exist!
  - Reading a non-existing property just returns `undefined`. So, we can easily test whether the property exists:

```
let user = {};
alert( user.noSuchProperty === undefined ); // true
```

  - There’s also a special operator "`in`" for that.

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

The “for…in” loop

- To walk over all keys of an object, there exists a special form of the loop: `for..in`.

```
let user = { name: "John", age: 30, isAdmin: true };

for (let key in user) {
    alert(key);  // name, age, isAdmin
    alert(user[key]); // John, 30, true
}
```

- We could use any variable name here instead of `key`.
  - `for(let prop in obj)` is also widely used.

Value type vs Reference type

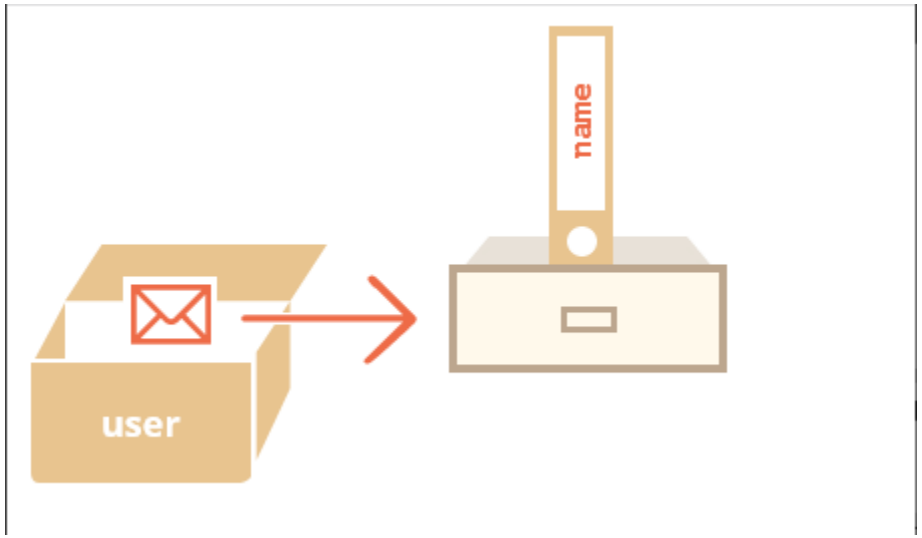
- One of the fundamental differences of objects versus primitives is that objects are stored and copied “by reference”, whereas primitive values: strings, numbers, booleans, etc. – are always copied “as a whole value”.

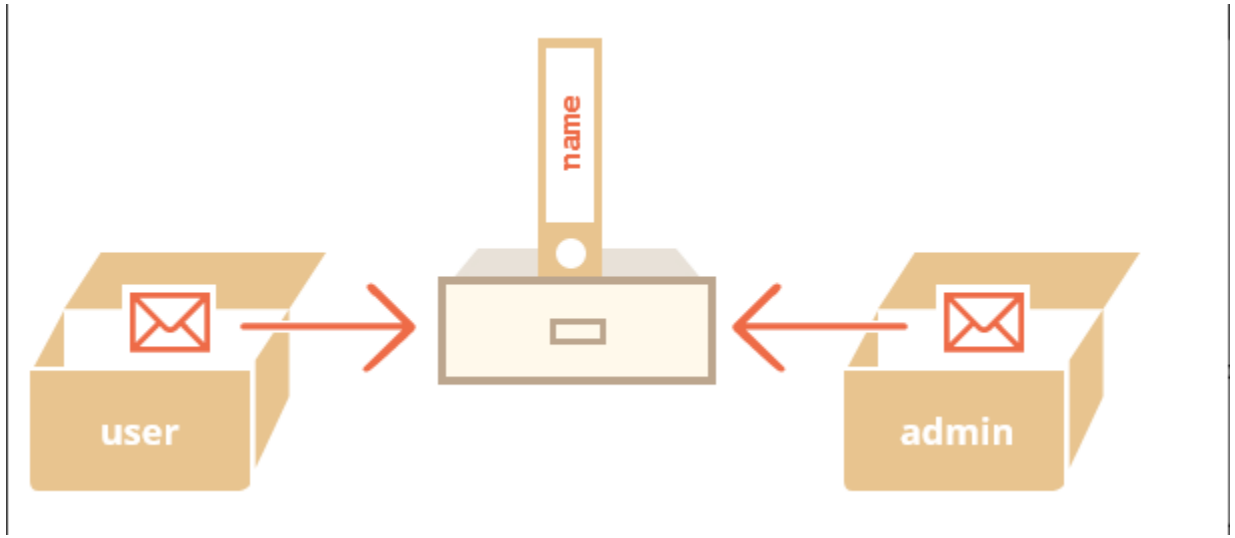
```
let message = "Hello!";
let phrase = message; // second copy of "Hello!";
message = "Hi!";
console.log(phrase); // Hello!

let user={id:123, name: "user"};
let admin = user; // there is still single copy of the object
admin.name= "admin";
console.log(user.name); // admin
```

Value type vs Reference type


```
let message = "Hello!";
let phrase = message;
```

```
let user = { name: 'John' }; let admin = user;
```





Comparison by reference

- Two objects are equal only if they are the same object.

```
let a = {};
let b = a; // copy the reference

alert( a == b ); // true, both variables reference the same object
alert( a === b ); // true
```

- Two independent objects are not equal, even though they may look identical.

```
let a = {};
let b = {}; // two independent objects

console.log(a == b); // false
console.log(a === b); //false
```

Call by value (“call by sharing”)

```
function changeVal(msg1) {
    msg1 = 'changed';
}

let msg2 = 'orginal';
changeVal(msg2);
console.log(msg2); // original
```

```
function changeObjRef(obj2) {
    obj2 = { msg: 'changed' };
}

let obj1 = { msg: 'original' };
changeObjRef(obj1);
console.log(obj1.msg); // original
```

```
function changeObjProp(obj1) {
    obj1.msg = 'changed';
}

let obj2 = { msg: 'original' };
changeObjRef(obj2);
console.log(obj2.msg); // changed
```

Exercise

- Write code that creates three different person objects, sam1, sam2, john
  - Assume for this domain that all person objects have name and age properties
  - Use object literals to create the objects
  - sam1 and sam2 both have values “Sam” and 10
  - john has values “John” and 10

- Write a function, isPersonEqual(obj1, obj2) that checks equality for person objects
  - assume that the only properties it needs to check are name and age
  - Call it with sam1 and sam2 and verify it returns true
  - Call with sam1 and john and verify false

Strings

Continuum of pure consciousness

Strings

- textual data stored as strings
  - no separate type for a single character
- can be created using single quotes, double quotes or backticks:

```
let single = 'single-quoted';
let double = "double-quoted";
let backticks = `backticks`;
```

- Single and double quotes are technically the same.
  - Our coding convention is to use double quotes for visual clarity
  - Also can use apostrophes in strings without escape
- Backticks allow us to embed any expression into the string
  - Wrap the expression in `${…}`
  - backticks also allow a string to span multiple lines.

Backticks

- Backticks allow us to embed any expression into the string
  - Wrap the expression in `${…}`
  - backticks also allow a string to span multiple lines.

```
let a = 5;
let b = 10;
console.log("Fifteen is " + (a + b) + " and\nnot " + (2 * a + b) + ".");
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
```

```
Fifteen is 15 and
not 20.
```

Escape Sequences

- Some characters have special meanings (', ", `, \ and more).
  - escape them by placing a backslash (\) immediately before

- pre-defined escape sequences l
  - \n is for new line, \t for tab …
  - https://www.w3schools.com/js/js_strings.asp

```
console.log("Hi I\'m Jack.\nI am a JS programmer.")
```

String length

- Length of a string can be accessed by using its `length` property.

```
console.log(`str\n`.length ); //4, Note that \n is a single “special” character
```

- `str.length` is a numeric property, not a method.

Accessing characters

- To get a character at position pos, use square brackets `[pos]` or call the method `str.charAt(pos)`.
  - The first character starts from the zero position:

```
let str = `Hello`;

// the first character
alert( str[0] ); // H
alert( str.charAt(0) ); // H

// the last character
alert( str[str.length - 1] ); // o
```

  - The square brackets are a modern way of getting a character.

Looping over iterables

- We can also iterate over characters using `for..of`:

```
for (let char of "Hello") {
    alert(char); // H,e,l,l,o (char becomes "H", then "e", then "l" etc)
}
```

- This construct can’t be used with objects.

- Exercise: refactor the for loop to use an index instead of for..of

Strings are immutable

- Strings can’t be changed in JavaScript.
  - It is impossible to change a character.

```
let str = 'Hi';
str[0] = 'h'; // doesn't work
console.log( str[0] ); // H
```

  - The usual workaround is to create a whole new string and assign it to the original variable.

```
let str = 'Hi';
str = 'h' + str[1]; // replace the string
alert( str ); // hi
```

String methods (APIs)

- JavaScript includes several useful string methods

```
let str = "Hello";

console.log(str.indexOf("l")); // 2
console.log(str.indexOf("Hell")); // 0
console.log(str.toUpperCase()); // HELLO
console.log(str.toLowerCase()); // hello
console.log(str.startsWith("H")); // true
console.log(str.substr(1,3)); //ell
```

- https://www.w3schools.com/jsref/jsref_obj_string.asp

Exercise

- Write a program that keeps on asking for user input and prints it, until user types the word "stop" (without quotes). "Stop" word can be in any case (small, capital or mixed)
- Write a function that takes a comma separated string of words and converts it into an array of words and prints in reverse order.
- Write a function to replace the first occurrence of "for" in an input string with 4.

input	output
Thanks for joining us.	Thanks 4 joining us.

References

- Objects (javascript.info)
- Object references and copying (javascript.info)
- Strings (javascript.info)