Object.create()

The Object.create() method is used to create a new object and link it to the prototype of an existing object. We can create a job object instance, and extend it to a more specific object.

```
const job = {
  position: 'cashier',
  type: 'hourly',
  isAvailable: true,
  showDetails() {
    const accepting = this.isAvailable ? 'is open' : 'is
closed'
    console.log(
      `The ${this.position} position is ${this.type} and
${accepting}.`
    )
  },
}

// Use Object.create to pass properties
const barista = Object.create(job)

barista.position = 'barista'
barista.showDetails()

// Output
// The barista position is hourly and is open.
```

swipe ⟶

Object.keys()

Object.keys() creates an array containing the keys of an object. We can create an object and print the array of keys.

```
// Initialize an object
const employees = {
    boss: 'Michael',
    secretary: 'Pam',
    sales: 'Jim',
    accountant: 'Oscar'
};

// Get the keys of the object
const keys = Object.keys(employees);
console.log(keys);

// Output
["boss", "secretary", "sales", "accountant"]
```

swipe ⟶

Object.values()

Object.values() creates an array containing the values of an object.

```
// Initialize an object
const session = {
    id: 1,
    time: `26-July-2018`,
    device: 'mobile',
    browser: 'Chrome'
};

// Get all values of the object
const values = Object.values(session);
console.log(values);

// Output
[1, "26-July-2018", "mobile", "Chrome"]
```

swipe ⟶

Object.entries()

Object.entries() creates a nested array of the key/value pairs of an object.

```
// Initialize an object
const operatingSystem = {
    name: 'Ubuntu',
    version: 18.04,
    license: 'Open Source'
};

// Get the object key/value pairs
const entries =
Object.entries(operatingSystem);
console.log(entries);

// Output
[
    ["name", "Ubuntu"]
    ["version", 18.04]
    ["license", "Open Source"]
]
```

swipe ⟶

Object.assign()

Object.assign() is used to copy values from one object to another.We can create two objects, and merge them with Object.assign().

```
// Initialize an object
const name = {
    firstName: 'Philip',
    lastName: 'Fry'
};

// Initialize another object
const details = {
    job: 'Delivery Boy',
    employer: 'Planet Express'
};

// Merge the objects
const character = Object.assign(name, details);
console.log(character);

// Output
{firstName: "Philip", lastName: "Fry", job:
"Delivery Boy", employer: "Planet Express"}
```

swipe ⟶

Object.freeze()

Object.freeze() prevents modification to properties and values of an object, and prevents properties from being added or removed from an object.

```
// Initialize an object
const user = {
    username: 'AzureDiamond',
    password: 'hunter2'
};

// Freeze the object
const newUser = Object.freeze(user);

newUser.password = '*******';
newUser.active = true;
console.log(newUser);

// Output
{username: "AzureDiamond", password: "hunter2"}
```

swipe ⟶

Object.seal()

Object.seal() prevents new properties from being added to an object, but allows the modification of existing properties. This method is similar to Object.freeze(). Refresh your console before implementing the code below to avoid an error.

```
// Initialize an object
const user = {
    username: 'AzureDiamond',
    password: 'hunter2'
};

// Seal the object
const newUser = Object.seal(user);

newUser.password = '*******';
newUser.active = true;
console.log(newUser);

// Output
{username: "AzureDiamond", password: "*******"}
```

swipe ⟶

Object.getPrototypeOf()

Object.getPrototypeOf() is used to get the internal hidden [[Prototype]] of an object, also accessible through the __proto__ property.

In this example, we can create an array, which has access to the Array prototype.

```
const employees = ['Ron', 'April', 'Andy', 'Leslie'];

Object.getPrototypeOf(employees);

// Output
[constructor: ƒ, concat: ƒ, find: ƒ, findIndex: ƒ, pop: ƒ, …]
```

swipe ⟶

Object.setPrototypeOf()

The Object.setPrototypeOf() static method sets the prototype (i.e., the internal [[Prototype]] property) of a specified object to another object or null.

```
const obj = {};
const parent = { foo: 'bar' };

console.log(obj.foo);
// Expected output: undefined

Object.setPrototypeOf(obj, parent);

console.log(obj.foo);
// Expected output: "bar"
```

swipe ⟶