Reflect API

It is a built-in object that provides methods for interceptable JavaScript operations.

Reflect is not a function object, so it's not constructible.

==========

Reflect.apply()

- It calls a target function with arguments as specified.

Reflect.apply(target, thisArgument, argumentsList)

target: The target function to call.

thisArgument: The value of this provided for the call to target.

argumentsList: An array-like object specifying the arguments with which target should be called.

The result of calling the given target function with the specified this value and arguments.

```
console.log(Reflect.apply("".charAt, "Egypt", [3])); //p
```

==========

Reflect.construct()

- It acts like the *new* operator, but as a function, It is equivalent to calling new target(...args)

Reflect.construct(target, argumentsList)

Reflect.construct(target, argumentsList, newTarget)

target: The target function to call.

argumentsList: An array-like object specifying the arguments with which target should be called.

newTarget Optional: The constructor whose prototype should be used. See also the new.target operator. If newTarget is not present, its value defaults to target.

```
function fun1(a, b, c, d){
    console.log(a+b+c+d)
}
Reflect.construct(fun1, ["We ", "should ", "love ", "ourselves "]);//We should love ourselves
```

==========

Reflect.defineProperty()

- It is like *Object.defineProperty()* but returns a Boolean

Reflect.defineProperty(target, propertyKey, attributes)

target: The target object on which to define the property.

propertyKey: The name of the property to be defined or modified.

attributes:The attributes for the property being defined or modified.

A Boolean indicating whether or not the property was successfully defined.

```
myObj = {Name:"Noura", Address:"Giza"}
Reflect.defineProperty(myObj, "Age", {value:23})
console.log(myObj)// Object { Name: "Noura", Address: "Giza", Age: 23}
```

==========

Reflect.deleteProperty()

- It allows to delete properties. It is like the *delete* operator as a function.

Reflect.deleteProperty(target, propertyKey)

target: The target object on which to delete the property.

propertyKey: The name of the property to be deleted.

A Boolean indicating whether or not the property was successfully deleted.

```
console.log(Reflect.deleteProperty(myObj, "hamada")) //true
console.log(myObj) // Object { Name: "Noura", Address: "Giza", Age: 23}
console.log(Reflect.deleteProperty(myObj, "Address")) //true
console.log(myObj) // Object { Name: "Noura", Age: 23}
```

Reflect.get()

- It works like getting a property from an object *(target[propertyKey])* as a function.

Reflect.get(target, propertyKey)

Reflect.get(target, propertyKey, receiver)

target: The target object on which to get the property.

propertyKey: The name of the property to get.

receiver Optional: The value of this provided for the call to target if a getter is encountered. When used with Proxy, it can be an object that inherits from target.

Returns the value of the property.

```
console.log(Reflect.get(myObj, "Name")) // Noura
let obj = new Proxy(myObj, {
    get(t, prop, receiver) {
      return receiver[prop] + "ada";
    },
  });
console.log(Reflect.get(obj, "Name", myObj)) // Nouraada
```

==========

Reflect.getOwnPropertyDescriptor()

- It is similar to *Object.getOwnPropertyDescriptor()*. It returns a property descriptor of the given property if it exists on the object, undefined otherwise.

Reflect.getOwnPropertyDescriptor(target, propertyKey)

target: The target object in which to look for the property.

propertyKey: The name of the property to get an own property descriptor for.

Returns a property descriptor object if the property exists in target object; otherwise, undefined.

```
//Reflect.getOwnPropertyDescriptor()
console.log(Reflect.getOwnPropertyDescriptor(myObj, "Age").value) //23
```

==========

Reflect.getPrototypeOf()

- It is almost the same method as *Object.getPrototypeOf()*. It returns the prototype (i.e. the value of the internal *[[Prototype]]* property) of the specified object.

Reflect.getPrototypeOf(target)

target: The target object of which to get the prototype.

Returns the prototype of the given object. If there are no inherited properties, null is returned.

```
console.log(Reflect.getPrototypeOf(Object.prototype)); // null
console.log(Reflect.getPrototypeOf({n:"nosa", a:44})); // Object.prototype
```

==========

Reflect.has()

- It works like the *in* operator as a function.

Reflect.has(target, propertyKey)

target: The target object in which to look for the property.

propertyKey: The name of the property to check.

Returns a Boolean indicating whether or not the target has the property.

```
console.log(Reflect.has(myObj, "Age")) // true
```

==========

Reflect.isExtensible()

- It determines if an object is extensible (whether it can have new properties added to it). It is similar to *Object.isExtensible()*, but with some differences.

Reflect.isExtensible(target)

target: The target object which to check if it is extensible.

Returns a Boolean indicating whether or not the target is extensible.

```
console.log(Reflect.isExtensible(myObj)) //true
let x = Object.seal({})
console.log(Reflect.isExtensible(x)) //false
var y = Object.freeze({})
console.log(Reflect.isExtensible(y)) //false
```

==========

Reflect.ownKeys()

- It returns an array of the target object's own property keys.

Reflect.ownKeys(target)

target: The target object which to get the own keys.

Returns an Array of the target object's own property keys.

```
console.log(Reflect.ownKeys(myObj)) //Array [ "Name", "Age" ]
```

==========

Reflect.preventExtensions()

It prevents new properties from ever being added to an object (i.e., prevents future extensions to the object). It is similar to *Object.preventExtensions()*, but with some differences.

Reflect.preventExtensions(target)

target: The target object object on which to prevent extensions.

Returns a Boolean indicating whether or not the target was successfully set to prevent extensions.

```
Reflect.preventExtensions(myObj)
console.log(Reflect.isExtensible(myObj)) //false
```

==========

Reflect.set()

- It works like setting a property on an object.

Reflect.set(target, propertyKey, value)

Reflect.set(target, propertyKey, value, receiver)

target: The target object on which to set the property.

propertyKey: The name of the property to set.

receiver Optional: The value of *this* provided for the call to the setter for *propertyKey* on *target*. If provided and target does not have a setter for *propertyKey*, the property will be set on *receiver* instead.

Returns a Boolean indicating whether or not setting the property was successful.

```
Reflect.set(myObj, "Name", "Mikasa")
console.log(myObj.Name) //Mikasa
Reflect.set(obj, "Name","7amada", myObj)
console.log(Reflect.get(obj, "Name", myObj)) //7amadaada
```

==========

Reflect.setPrototypeOf()

- It is the same method as *Object.setPrototypeOf()*, except for its return type. It sets the prototype (i.e., the internal *[[Prototype]]* property) of a specified object to another object or to null, and returns true if the operation was successful, or false otherwise.

Reflect.setPrototypeOf(target, prototype)

target: The target object of which to set the prototype.

prototype: The object's new prototype (an object or null).

Returns a Boolean indicating whether or not the prototype was successfully set.

```
console.log(Reflect.setPrototypeOf({}, Object.prototype)); // true
console.log(Reflect.setPrototypeOf({}, null)); //true
console.log(Reflect.setPrototypeOf(Object.freeze({}),null )) //false
```

==========

Name: **Noura Mahmoud**

Track: **Mobile Cross-platform**