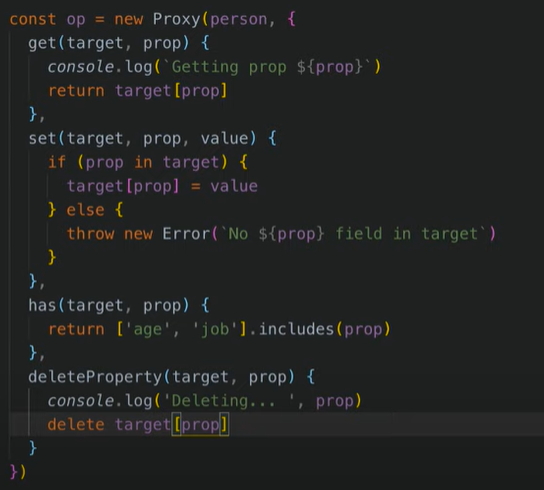
**Proxy**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Proxy

**Proxy** клас для задання логіки взаємодій із даними в функціях, об’єктах і класах

**Object =================================**

****

Для створення даного функціоналу – в конструктор класу **Proxy**(obj, validator) передаємо 2 параметри: **obj** – взаємодії із даними якого ми будемо відстежувати і **validator** – об’єкт, в якому ми налаштовуємо **get** I **set** методи для відстеження взаємодій із полями об’єкта

Для **get**(target, prop) – задаємо 2 параметри **target** – об’єкт із полями якого ми взаємодіємо і **prop** – ключ поля із яким ми збираємось взаємодіяти. В самому методі налаштовуємо логіку, що спрацьовуватиме при отримані даних із об’єкта який буде моніторити Proxy. Є ще **третій необов’язковий параметр** **receiver –** який є аналогом метода **ownKeys**, який ми також можемо перевизначити

**ownKeys**: function(obj) – метод для перебору власних ключів об’єкта. Дефолтна реалізація **Reflect**.**ownKeys**(obj)

Для **set**(target, prop, value) – target , value мають ті ж значення, що і в методі get, value значення, яке ми присвоюватимемо полю об’єкта

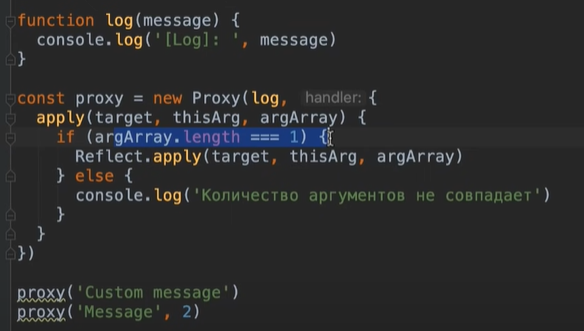
**has**(target, prop) – опрацьовує функціонал перевірки наявності даного поля в об’єкта. Дефолтна провірка **prop in obj**

**deleteProperty**(target, prop) – функція навішування логіки при видалені конкретного поля.

**ownKeys**(target) – метод для виведення ключів об’єкта

**Function =======================================**

****

****

Синтаксис налаштувань має дещо інший вигляд – в конструктор Proxy() ми передаємо функцію, яку потрібно моніторити, і метод, в якому ми описуємо логіку відстеження взаємодії із даною функцією

Сама ж функція для моніторингу містить в собі метод

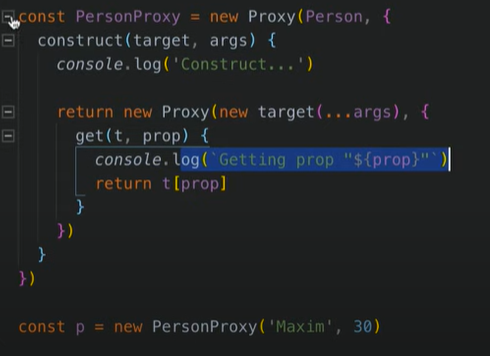
**apply**(**target, thisArg, arrayArg**) -> target – поточна функція thisArg –контекст, якщо ми його передавали через (call, apply, bind), arrayArg – параметри, що передаються в функцію

Логіка виклику поточної функції в Proxy

**Reflect**.**apply**(**target, thisArg, arrayArg**)

**target**.**apply**(**thisArg, arrayArg**)

**Classes =======================================**



Для ініціалізації класу в проксі ми передаємо його перший параметром.

**constract**(target, prop) – спрацює при ініціалізацї нового екземпляру класу. В даному методі, для імітації звичного функціоналу ініціалізації екземпляру класу достатньо прописати:

**return new target(…args)**

Існує можливість навішати логіку взаємодії із екземпляром класу, як ми робили із об’єктами, використовуючи ті ж методи (get, set, has, deleteProperty), для цього потрібно повернути Proxy в який передаємо першим параметром **new target(…args),** а другим конфігуратор