## Прототипное наследование

В JavaScript объекты имеют специальное скрытое свойство **[[Prototype]]**

[[Prototype]] либо **null**, либо **ссылка** на другой объект.

**__proto__** — исторически обусловленный геттер/сеттер для [[Prototype]]

Современный JavaScript предполагает
**Object.getPrototypeOf/Object.setPrototypeOf**

Прототип используется только для чтения свойств. Операции записи/удаления работают напрямую с объектом. Свойства-аксессоры – исключение.

Цикл **for..in** проходит не только по собственным, но и по унаследованным свойствам объекта. отфильтровать их можно при помощи метода obj.hasOwnProperty(key)

Почти все остальные методы, получающие ключи/значения, такие как **Object.keys, Object.values** и другие – игнорируют унаследованные свойства.

### Методы прототипов, объекты без свойства __proto__

- Object.create(proto[, descriptors]) – создаёт пустой объект со свойством **[[Prototype]]**, указанным как **proto**, и необязательными дескрипторами свойств **descriptors**.
- Object.getPrototypeOf(obj) – возвращает свойство **[[Prototype]]** объекта **obj**.
- Object.setPrototypeOf(obj, proto) – устанавливает свойство **[[Prototype]]** объекта **obj** как **proto**.
- obj.isPrototypeOff(obj): возвращает **true**, если объект входит в цепочку прототипов другого объекта.
- obj.hasOwnProperty(key): возвращает **true**, если у **obj** есть собственное (не унаследованное) свойство с именем **key**.

**Object.create** даёт нам лёгкий способ создать поверхностную копию объекта со всеми дескрипторами:

```
let clone = Object.create(Object.getPrototypeOf(obj),
Object.getOwnPropertyDescriptors(obj));
```

Создать объекты без прототипов с помощью **Object.create(null)**

Ещё методы:

## F.prototype

**F.prototype** свойство функции содержащий объект прототип

Если в **F.prototype** содержится объект, оператор **new** устанавливает его в качестве **[[Prototype]]** для нового объекта.

В старые времена, прямого доступа к прототипу объекта не было. Надёжно работало только свойство **"prototype"** функции-конструктора.

Если после создания свойство **F.prototype** изменится (**F.prototype = <другой объект>**), то новые объекты, созданные с помощью **new F**, будут иметь в качестве **[[Prototype]]** другой объект, а уже существующие объекты сохранят старый.

F.prototype по умолчанию имеет свойство constructor

У каждой функции (за исключением стрелочных) по умолчанию уже есть свойство **"prototype"**

По умолчанию **"prototype"** – объект с единственным свойством **constructor**, которое ссылается на функцию-конструктор.

Чтобы сохранить верное свойство **"constructor"**, мы должны добавлять/удалять/изменять свойства у прототипа по умолчанию вместо того, чтобы перезаписывать его целиком:

## Встроенные прототипы

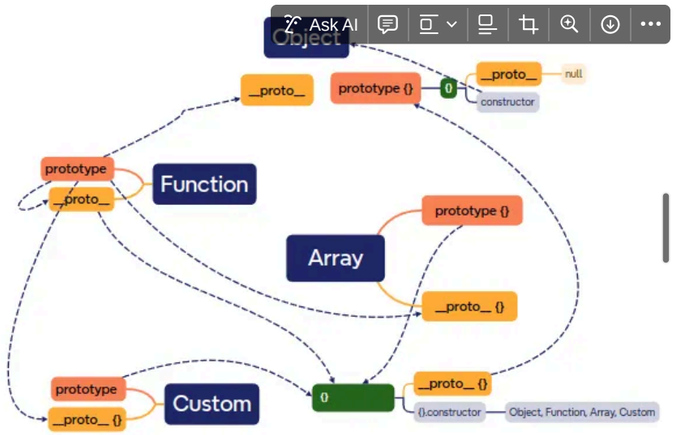
### Object.prototype

Все встроенные функции-конструкторы используют свойство **prototype**

Object - функция конструктор (new Object) у которой есть свойство **prototype**

### Другие встроенные прототипы

**Array, Date, Function, Number**




https://xmind.ai/3lQr1oJ5

- Function - функция конструктор для других функций конструкторов: Object, Array, Number
- любая функция - экземпляр new Function
- Function.prototype === Function.__proto__
- Object.prototype.__proto__ === null

### Примитивы

Для примитивных типов данных String, Number, Boolean при обращении к методам будет создан временный объект обертка. null и undefined не имеют оберток.

**Изменение встроенных прототипов** не рекомендуется, кроме как для полифилов

### Заимствование у прототипов