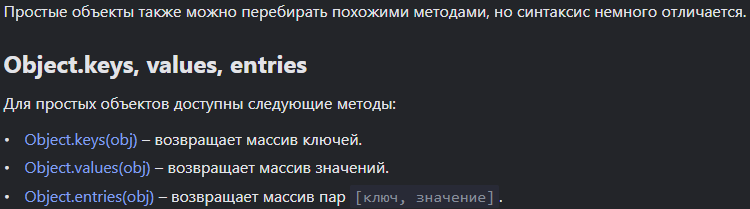
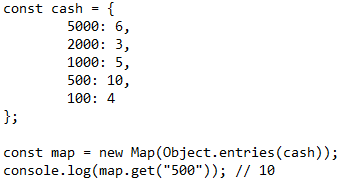
1. **JavaScript**……………………………………………………………………….25
   1. Object………………………………………………………………………………...25
      1. Перебор свойств и значений………………………………………………25
      2. Примечания………………………………………………………………...26
   2. Array…………………………………………………………………………………29
   3. String…………………………………………………………………………………29
   4. RegExp………………………………………………………………………………….
   5. Promise………………………………………………………………………………..
   6. Function……………………………………………………………………………...29
2. **JavaScript**
   1. **Object**
      1. Перебор свойств и значений:



* + 1. Преобразование Object to Map and Map to Object



**Object to Map**

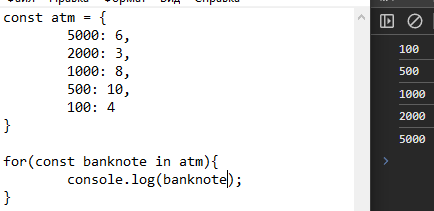


**Map to Object**

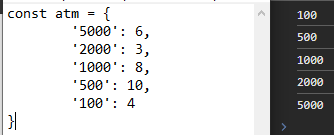
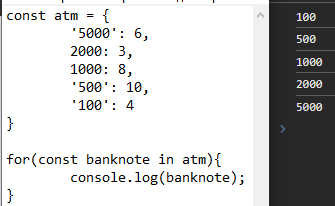
* + 1. Примечания

• Свойства объекта по умолчанию имеют тип String. Свойства объекта отсортированы особым образом:

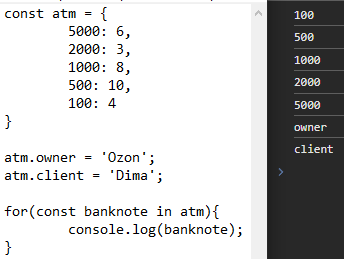
1. свойства с целочисленными ключами располагаются в порядке возрастания:



Примечание: интерпретатор не видит разницы между целочисленными свойствами с типом Number и целочисленными свойствами с типом String, вероятно под капотом осуществляется кастование типов из String в Number:

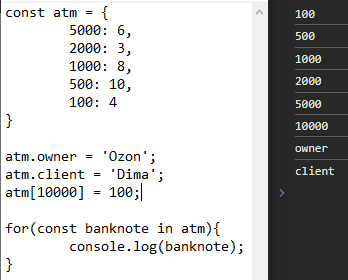
 

1. остальные свойства располагаются в порядке добавления по принципу FIFO:



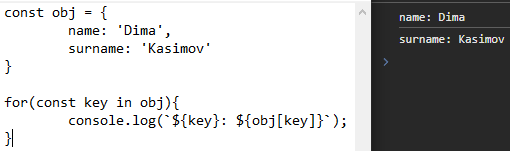
Примечание: **первыми** свойствами всегда идут **целочисленные свойства** объекта.

Примечание: если в объект, после его инициализации, добавить новое **целочисленное** свойство, то оно **переместится** в группу целочисленных свойств и подчинится правилу сортировки целочисленных свойств:

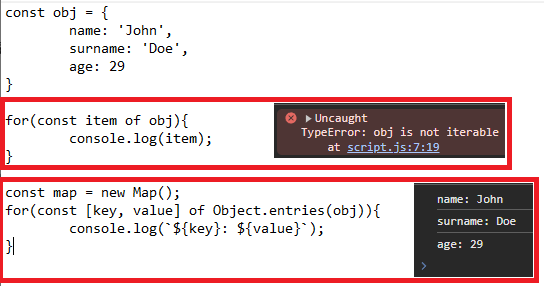


• Для перебора свойств объекта можно использовать следующие способы:

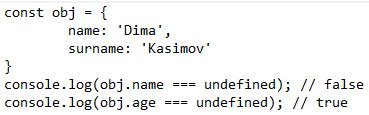
• проитерироваться через цикл «for in»:



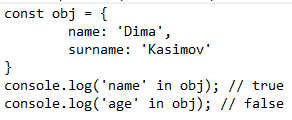
• продублировать пары свойство-значение в экземпляр класса Map:



• Проверить наличие свойства в объекте можно двумя способами:  
 • obj.key === undefined



• “key” in obj

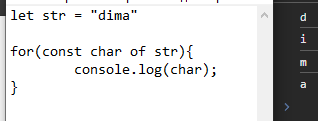


* 1. **Array**
  2. **String**
     1. Примечания

• Строки иммутабельны:



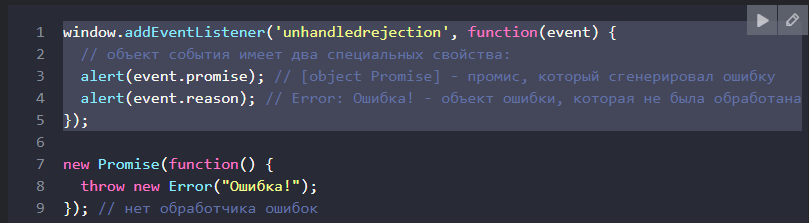
• По строкам можно итерироваться через цикл «for of»:



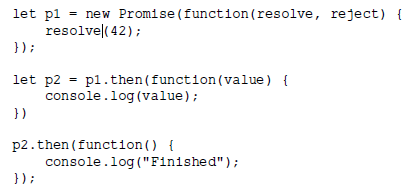
* 1. **RegExp**
  2. **Promise**
     1. Примечания

• В цепочке обработчиков промиса .catch можно расположить в единичном количестве в конце цепочки (обработать все ошибки), либо рассовать их по цепочке таким образом, чтобы последующие .then обработчики не отработали или отработал следующий .catch обработчик (пробросили исключение, которое будет обработано следующим обработчиком) и только после обработки ошибки будут отрабатывать последующие за .catch .then обработчики, если они есть.

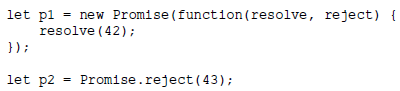
• В браузерной среде для обработки необработанных исключений (генерируется специальное событие при появлении необработанной ошибки) можно использовать unhanledrejection событие у глобального объекта «window».



• Обработка промиса .then также вернёт Promise.



• Промис может быть создан уже **установившимся** (p2), а не только как промис с запланированным выполнением (p1)



• Для получения результата работы сразу нескольких промисов можно воспользоваться Promise.all([p1,p2,…pn]) и Promise.race([p1,p2,…pn]);

Promise.all([p1,p2,…pn]) вернет промис, с соответствующим массивом результатов работы вложенных промисов. В случае отказа в одном из переданных промисов, вернется **НЕ МАССИВ**, а **результат отклонённого промиса**

Promise.race([p1,p2,…pn]) вернет промис с результатом первого успешно отработавшего промиса. **Важно**: если в Promise.race([p1,p2,…pn]) пробросить установившийся промис, то только он и будет обработан (даже если установившийся промис будет в состоянии rejected)

* 1. **Function**