**Value Type & Reference Type in Javascript**a simple look at computer memory explains what’s happening

Primitive Types: Boolean, null, undefined, string, number

Reference: Array, Function, Object

* Primitives

텍스트이(가) 표시된 사진

자동 생성된 설명

variable에 primitive type이 할당된다면 이것을 variable as containing the primitive value라 생각할 수 있음.

테이블이(가) 표시된 사진

자동 생성된 설명

그리고 memory에 위와 같이 올라갈 것임

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 위와 같이 x, y에 primitive type을 할당한 후, 새로운 variables에 이 변수를 ‘다시’ =을 사용해 할당해준다 -> ‘새로운 변수에 기존 변수를 COPY한다’

테이블이(가) 표시된 사진

자동 생성된 설명

-> copy의 개념이기 때문에 새로운 변수는 새로운 메모리를 할당 받기 때문에 copy한 주체와 할당 받은 객체는 메모리 관점에서 서로 관련성이 없다.

텍스트이(가) 표시된 사진

자동 생성된 설명

따라서 변수 값을 할당 받은 새로운 변수에 값을 수정해도 기존 변수에 아무런 영향을 끼치지 않는다

* Objects

non-primitive value를 할당 받은 변수들은 해당 value에 대한 reference가 주어진다.  
Objects는 computer’s memory에 생성되는데, arr=[]를 쓰면 메모리에 배열이 생성된다. 변수 arr가 수신하는 것은 array의 주소와 위치이다.

address가 number, string처럼 값으로 전달되는 new data type이라 가정했을 때, 이를 출력하면 어떤 값이 나올까? 🡪 address가 reference에 의해 전달되는 값의 위치를 메모리 내에서 가리키기 때문에 value는 <>로 표시된다. (string은 ‘’, “”인 것처럼)

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 코드 실행 결과 memory에 모습은 아래와 같다

테이블이(가) 표시된 사진

자동 생성된 설명

또 주목할 점은 variable의 address와 value는 static하다는 것이다. 그저 메모리에 array가 변할 뿐이다. 우리가 위와 같은 reference variable을 사용하여 무언가를 하면 js engine은 arr의 실제 메모리 위치로 가서 작동한다.

* Assigning by Reference

primitive value를 가진 variable에 의해 할당되면 value값이 copy되었다. non primitive value, object또한 마찬가지이다. 그러나 그래 보이지 않을 뿐, 예시를 살펴보자

텍스트이(가) 표시된 사진

자동 생성된 설명

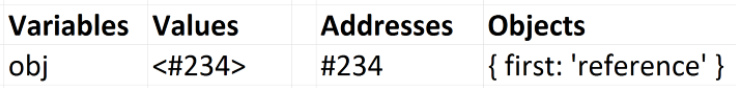
테이블이(가) 표시된 사진

자동 생성된 설명

기존 variable copy와 같이 똑같이 Value가 copy되었다. 그러나 복사된 value가 가리키는 것은 Address의 위치, 주소이며 이는 reference변수의 Objects를 가리키고 있다.  
🡪 따라서 reference 변수가 가리키는 object의 값이 변하면 refCopy변수가 가리키는 object의 값도 변하는 것 = 같은 Objects를 주시한다.

* Reassigning a Reference





위와 같이 json object를 obj 변수에 할당하였다. 그리고 다시 위 변수에 재할당하려 한다.



텍스트이(가) 표시된 사진

자동 생성된 설명

기존에 가리킴 받던 #234에 objects는 재할당과 동시에 사라지는 것이 아니라 메모리상에 여전히 존재한다. 그리고 새롭게 할당된 object의 address 주소를 value값으로 바뀌게 된다. 그러면 #234는 계속 존재하는가? 아니다! javascript engine이 garbage collection을 수행하게 되어 나중엔 메모리에서도 사라지게 된다.

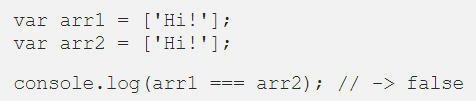
* == and ===

reference-type variables에 사용된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

여러 개의 variables가 동일한 item에 한 reference를 가진다면 비교 result 값은 true이다.



그러나 여러 변수가 별개의 objects라면 동일한 properties를 가질지라도 비교 결과는 false가 된다.

따라서 distinct objects들 사이의 properties가 동일한지 비교하기 위해 가장 쉬운 방법은 모두 string으로 type을 바꿔서 비교하는 것이고 혹은 recursively loop를 이용해 비교하는 것이다.

* Passing Parameters through Functions

function의 parameter로 primitive values를 전달하는 것은 ‘=’을 쓰는 것과 동일하다. 매개변수가 parameter로 pass된 값을 복사하는 방식이기 때문.

텍스트이(가) 표시된 사진

자동 생성된 설명

위와 같은 코드를 실행시켜 ‘// PAUSE’ 까지의 메모리 상태를 보면 아래와 같다

테이블이(가) 표시된 사진

자동 생성된 설명

* Pure Functions

scope 밖에 어떠한 영향도 끼치지 못하는 function을 pure function이라 하며 parameter로 primitive value를 가지면 자동으로 함수 내에서만 영향을 미치게 된다. 따라서 함수가 return되고 가비지 콜렉터 된다.

그러나 Object를 함수내에서 사용했을 때는 함수 밖의 영역에서도 영향을 미친다. ex) array,   
옛날에 읽었던 MIT에서 배웠던 내용! -> mutable한 array를 func parameter로 받아서 처리하는 경우, 해당 scope 밖에 변화가 일어나기 때문에 지양해야 함. (모듈 단위 테스트를 한다고 쳤을 때, 굉장히 위험한 행위 동작임)

pure function으로 쓰여진 native array function중 Array.map, Array.filter가 있다. 얘들은 원본을 건드리지 않고 내부 값을 복사하여 처리하기 때문에 immutable한 처리가 가능하다

* + impure function 예시

텍스트이(가) 표시된 사진

자동 생성된 설명

* + pure function 예시

텍스트이(가) 표시된 사진

자동 생성된 설명

* Test Yourself

텍스트이(가) 표시된 사진

자동 생성된 설명  
🡪 출력되는 값은 무엇일까!

시계이(가) 표시된 사진

자동 생성된 설명

Remember that assignment through function parameters is essentially the same as assignment with =