JavaScript

2014-03-27 박지민(differ)

개요

자바스크립트라는 게 대체 뭐지?

Javanese Script?

Java™Script?

JavaScript!

- 동적인 웹 사이트를 만들기 위해 만들어진 언어

JavaScript의 역사

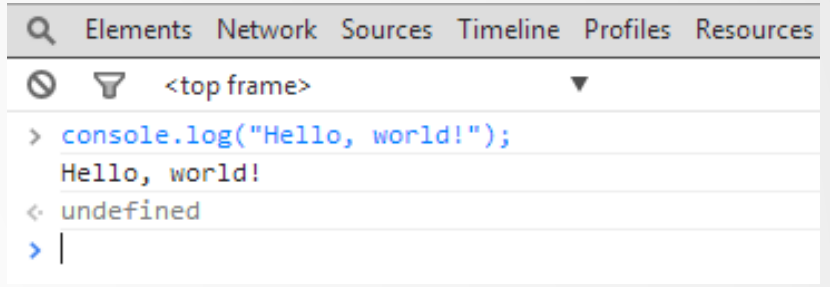
- “웹 브라우저 전쟁”
  - 인터넷 익스플로러와 넷스케이프 사이의 경쟁
  - “뭔가 멋진 새 기능이 있었으면 좋겠다.”
    - 인터넷 익스플로러: `<marquee>`
    - 넷스케이프: `<blink>`, `<frame>`
  - 넷스케이프에서 client-side 언어를 만들고 싶어했다.
    - 마이크로소프트의 Visual Basic같은 편리성
    - “웹 상에서 쓸 수 있는 자바 같은 녀석”
  - 이렇게 자바스크립트가 만들어졌다.
    - 원래 이름은 LiveScript (코드명 Mocha)
    - 넷스케이프에서 자바를 지원하면서 홍보하기 쉽게 이름을 바꿈.
      - “얘 자바랑 비슷한 인터프리트 언어인데 쓰기 편하고 뭐 더 안 깔아도 돼요!“

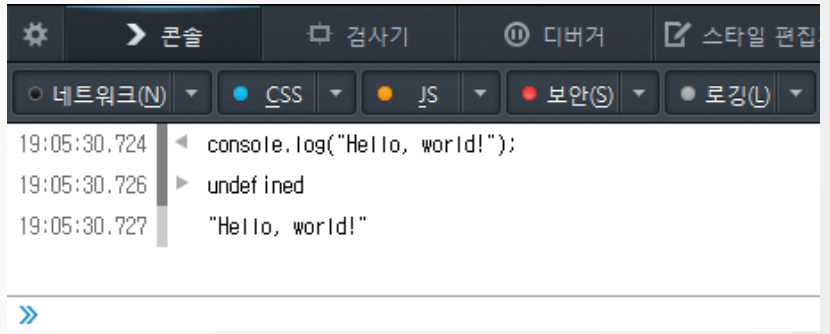
JavaScript의 역사

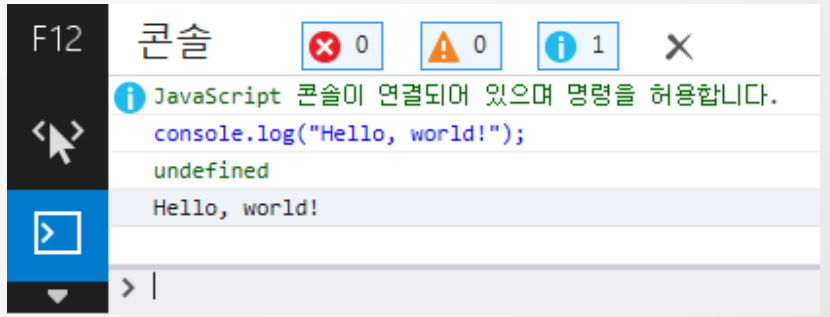
- 다양한 브라우저에서 지원
  - 인터넷 익스플로러: JScript (이름만 다르고 같은 놈이다.)
  - 이후 모든 웹 브라우저에서 자바스크립트를 지원하게 되었다.
- 표준 지정
  - ECMAScript: ECMA international에서 만든 자바스크립트 표준
    - 현재 5.1까지 나왔고 6이 만들어지고 있는 중.
  - 다양한 언어가 ECMAScript를 기반으로 제작되었다.
    - JavaScript, JScript, ActionScript 3, TypeScript, ExtendScript, QtScript, …
- 서버 사이드 자바스크립트
  - 서버에서도 자바스크립트를 쓸 수 있다.
    - 예: Node.js
  - 프론트엔드, 백엔드에서 같은 언어를 쓸 수 있다.
    - npm의 많은 패키지들은 웹 브라우저에서도 쓸 수 있다.

“자바스크립트 어떻게 깔죠?”

- 얜 이미 깔려있다
  - 웹 브라우저 콘솔
    - F12, Ctrl+I 또는 Ctrl+K
    - 여러 줄 입력: Shift+Enter
  - Node.js
    - > node
    - 아, 얘는 안 깔려 있을수도
- 모든 곳에서 모든 것을!
  - 데스크톱 (with HTML5)
    - 영상/오디오 처리, WebGL, 게임 컨트롤러, …
  - 모바일 (with HTML5)
    - 위치, 멀티터치, 카메라, 가속도 센서, …
  - 서버 (with Node.js)
    - 소켓, 멀티프로세스, 파일, …







참고: HTML에 스크립트 삽입

- cd public_html
  - mkdir ~/public_html
- vi blah.html
- <script>, </script>를 넣고 사이에 스크립트를 적는다.

기본적인 연산들

왜 그걸 손으로 계산하고 있지?

경고: 부정확한 설명

설명을 간단하게 하기 위해 슬라이드에서 자세히 설명하지 않고 넘어가는 게 많으니,
여기에 있는 걸 대강 보고 그게 전부라고 생각하지 말고, 노트를 참조할 것!

간단한 사칙연산

- 덧셈 / 뺄셈 / 곱셈 / 나눗셈 / 나머지
  - 정수와 소수 사이에 구별이 없다.
    - 정수 나누기 정수 = 소수
    - 사실 자바스크립트에는 정수가 없음.
  - 0으로 나누기를 하면?
    - 1/0 = Infinity
    - 1/-0 = -Infinity
    - 0/0 = NaN (Not a Number)
  - IEEE 부동 소수점 표준을 따름
    - SP때 배우게 될 거에요 ^_^
- 연산 순서: 곱셈과 나눗셈, 나머지, 덧셈과 뺄셈
  - 헷갈릴 때는 괄호로 잘 묶어주면 됩니다.

```
> 123+421
544
> 16-27.5
-11.5
> 42+56*12-3
711
> 5/123
0.04065040650406504
> 1e5 + 4.2e-5
100000.000042
> 11%4
3
```

변수

- 어떤 값을 가지고 있는 놈
  - 자바스크립트 변수는 다양한 값들을 들고 있을 수 있다.
    - 숫자, 문자열, 배열, 오브젝트, 함수, …
  - 변수마다 각자의 이름이 있다.
    - JS에서는 다양한 유니코드 문자를 변수 이름으로 쓸 수 있다.
      - 물론 한글 이름도 가능!
      - …하지만 웬만하면 ASCII 코드 영역 내에 있는 이름을 쓰자.
    - 이미 자바스크립트에서 쓰고 있는 단어들은 사용할 수 없다.
- 등호(=)를 써서 변수에 값을 대입할 수 있다.
  - 수학에서 쓰이는 “같다”의 뜻이 아님!
    - 이거 프로그래밍할 때 자주 실수함!
    - 나중에 나오겠지만 이 때는 ==를 쓴다.

```
> a=4
4
> b=7
7
> a*b-a-b
17
> a=a*b
28
> a+b
35
```

변수

- +=, -=, *=, /=, %=
  - a+=b는 a=a+b와 같다.
  - 왼쪽에 적은 값에 오른쪽에 적은 값과의 연산 값을 저장한다.
- a++, ++a
  - a++는 a+=1과 같다.
  - a++와 ++a의 차이는?
    - a++ : a의 값을 내뱉은 뒤 a를 1 증가시킴.
    - ++a : a를 1 증가시킨 뒤 a의 값을 내뱉음.
  - a--, --a도 있다.
    - 뭔지는 굳이 설명 안 해도 알 것 같은데…

```
> a=3
3
> a+=15
18
> a/=3
6
> a%=4
2
> a++
2
> ++a
4
```

문자열

- 자바스크립트로 할 수 있는 게 숫자놀이만은 아니다!
- 문자열은 “” 또는 ”로 감싸면 된다.
  - 두 따옴표 사이에는 아무런 차이가 없다.
  - “” 안에 또 “”를 넣고 싶을 때는 \”처럼 앞에 백슬래시를 붙이자!
  - \n: 개행 문자 (줄바꿈), \t: 탭
  - 그렇다면 백슬래시는 어떻게 넣지?
- +로 두 문자열을 합칠 수 있다.

```
> str="안녕하세요, 만나서 반갑습니다!"
"안녕하세요, 만나서 반갑습니다!"
> str+="으으"
"안녕하세요, 만나서 반갑습니다!으으"
> str="뭐 적어야 될지\n모르겠다"
"뭐 적어야 될지
모르겠다"
> str='"JavaScript, it\'s easy!", I said.'
""JavaScript, it's easy!", I said."
```

“원래 이런 거에요”

- 자바스크립트는 타입이 느슨한 언어!
  - 문자열과 숫자를 같이 다룰 때 조심해야 한다!
  - 특히 덧셈을 할 때는 조심!
- 수학 연산을 할 때 변수가 숫자인지 확인하자!
  - 단항 + 연산자를 사용해서 문자를 숫자로 바꿀 수 있다.
    - +“12”는 숫자 12다.
  - typeof 연산자를 이용해 타입을 확인할 수 있다.

```
> "12"+34
"1234"
> "12"-34
-22
> "12"-"34"
-22
> 3+1+"10"+5
"4105"
> "12"/"4"+"6"
"36"
```

진릿값

오오오 자바스크립트 그거슨 진리

- 어떤 말이 참인지 거짓인지를 나타내는 값.
  - 예: “여기 변수 a에 있는 값이 b에 있는 값보다 큰지 여부”
  - true, false 두 가지 값이 가능하다.
- 여러 비교 연산자들을 쓸 수 있다.
  - ==, <, >, <=, >=, !=
- 진릿값들끼리 연산을 할 수도 있다.
  - ==, !=, ||, &&, !
  - 연산 순서는… 직접 실험 해 보자.

```
> 42<123
true
> "Hello"=="Hello"
true
> "Hello"=="world"
false
> a=1+1
2
> a == 3-1
true
```

“원래 이런 거에요” x 2

- 2==“2”
  - ==와 !=는 타입을 그렇게 신경 쓰지 않는다.
    - `0 == “” && 0 == “0” && “” != “0”`
  - 타입까지 신경 쓰고 싶을 때는 ===와 !==를 쓰자.
    - 잘 모른다면 일단 그냥 저 등호 3개짜리를 쓰자.
- x===x는 항상 성립할까?
- &, |은 &&, ||와 다르다.
  - 뒤에 건 진릿값들 사이의 연산이다.
  - 앞에 건 비트 연산이다.
    - 3|4는 7이다.
    - 3||4는…?

```
> 2=="2"
true
> 2==="2"
false
> 3&10
2
```

그런데 이거 가지고 뭐하지?

조건문과 반복문

이거 없이 프로그래밍 할 수 있음?

Hello, world!

- console.log(출력할 내용)
  - 지금은 console.log라는 함수를 호출한다고만 알아 두자.
    - console.warn
    - console.error
    - console.info

```
> console.log("Hello, world!")
  Hello, world!
< undefined
> x = 2+2
  4
> console.log("2+2는 "+x)
  2+2는 4
< undefined
> console.log(42*123-x)
  5162
< undefined
```

조건문

- 주어진 조건의 진릿값에 따라 다른 행동을 하게 한다.
- if(조건){
  … 조건이 참이면 할 일 …
  }
- if(조건){
  … 조건이 참이면 할 일 …
  }else{
  … 조건이 거짓이면 할 일 …
  }

```
> a=12;
  if(a>15){
    b=10;
  }else{
  b=20;
  }
  20
> b
  20
```

```
> a=24;
  if(a>15){
    b=10;
  }else{
  b=20;
  }
  10
> b
  10
```

세미콜론

- 한 문장(?)을 쓸 때 뒤에 붙인다.
  - 문장의 끝을 알 수 있다.
  - if문 등에서는 뒤에 추가로 붙이지 않아도 된다.
- 자바스크립트에서 세미콜론은 꼭 붙이지 않아도 된다.
  - 줄바꿈이 있으면, 붙여야 될 경우 그 곳에 알아서 세미콜론을 붙여준다.
  - C랑 Java는 세미콜론이 꼭 있어야 된다.
- 하지만 이왕이면 붙이는 게 좋다.
  - 한 줄에 여러 문장을 적을 수 있다.
  - 가~끔 있는 일이지만 세미콜론이 안 붙는 경우도 있다.
    - “분명히 코드는 제대로 짰는데 왜 이러지?”
  - ~~C나 Java에 익숙한 사람에게 정신적 위안을 준다.~~

코드 블록

- 여러 개의 동작을 하나로 묶어준다.
- 파이썬과는 다르게, 들여쓰기는 별로 중요하지 않다.
  - 물론, 들여쓰기를 안 한다면… ㅎ_ㅎ
  - 코드를 쓸 때에는 같이 프로젝트에 참여하는 사람, **미래의 자기 자신**이 읽을 수 있게 쓰는 것이 중요하다.
- 많은 곳에서 한 문장 짜리 블록은 중괄호를 생략해도 된다.
  - `if(x>100) console.log("100 초과");`
  - `if(x>100){console.log("100 초과");}`
  - `if(x>100){console.log("100 초과")}`

if … else if … else

- else 대신 else if로 나머지 경우에 대한 처리를 할 수 있다.
  - 여러 개 붙일 수도 있다.
  - 마지막 else가 없어도 된다.
    - 하지만 else 뒤에 바로 else if가 올 수는 없다.
  - 3개 이상의 경우로 나눌 때 유용하다.
    - 한 경우를 처리했을 때 다른 경우는 처리하면 안 될 때.
  - if-else 개수를 확 줄일 수 있다.

```
> x = 10;
  if(x<50) console.log("50 미만");
  else if(x<100) console.log("100 미만");
  else console.log("100 이상");
  50 미만
< undefined
```

```
> x = 60;
  if(x<50) console.log("50 미만");
  else if(x<100) console.log("100 미만");
  else console.log("100 이상");
  100 미만
< undefined
```

```
> x = 140;
  if(x<50) console.log("50 미만");
  else if(x<100) console.log("100 미만");
  else console.log("100 이상");
  100 이상
< undefined
```

switch

- if … else if …보다 쓰기 편하다.
- switch(변수)
  - 변수 하나에 대해 여러 값과 비교한다.
- case 값:
  - 변수와 값이 같으면 아래 있는 코드를 실행한다.
    - 비교는 ===로 한다.
- break;
  - switch문을 빠져 나온다.
    - 안 쓰면 아래 case문이나 default문까지 실행된다!
- default:
  - 어느 case도 만족하지 못했을 때 실행된다.
  - 없어도 된다.

```
grade = 'A';
switch(grade){
  case 'A':
    console.log("A");
  case 'B':
    console.log("A나 B");
    console.log("break");
    break;
  case 'C': case 'D':
    console.log("C나 D");
    break;
  default:
    console.log("-_-");
}
```

다른 조건문

- 조건?참:거짓
  - `abs = x>0 ? x : -x;`
  - `if(x>0) abs=x; else abs=-x;`
  - 가독성을 너무 떨어뜨리지만 않는다면 유용하게 쓸 수 있다.

- **사족: ||, &&을 조건문처럼 쓸 수 있다.**
  - `value > 0 && foo = 1/value;`
  - 코드를 간략하게 만들 수 있지만 그다지 권장하지는 않음. ~~나는 좋아하지만~~

while 반복문

- **while**(조건){
  …조건이 참일 동안 할 일…
  }

- **if문과 비슷하지만 블록을 계속 실행한다.**
  - 똑같은 동작을 여러 번 할 수 있다.
  - 이제 컴퓨터가 할 수 있는 모든 연산을 할 수 있다!

```
> i=0;
  while(i<5){
    i++;
    console.log("i="+i);
  }
  i=1
  i=2
  i=3
  i=4
  i=5
< undefined
```

do…while 반복문

- do{
  …할 일…
  }while(조건);
- while문과 같지만 우선 블록을 한 번 실행한다.
  - 블록 안에 있는 내용을 최소한 한 번은 실행하고 싶을 때 유용!

```
> i=10;
  do{
    console.log("i="+i);
  }while(i<5);
  i=10
< undefined
```

for 반복문

- **for**(A;조건;B){
  …할 일…
  }
  - while문과 비슷하지만 좀 더 간략하게 쓸 수 있다.
  - A: for문이 시작되기 전에 실행됨
  - B: 블록을 한 번 실행한 다음 실행됨.
- A;
  while(조건){
  …할 일…
  B;
  }

```
> for(i=0;i<5;i++){
    console.log(i);
  }
  0
  1
  2
  3
  4
< undefined
```

첫 실습: 구구단

- 2*1부터 9*9까지의 곱셈 결과를 출력하는 프로그램을 만들어 보자.

```
for(i=2;i<=9;i++){
  for(j=1;j<=9;j++){
    mul=i*j
    console.log(i+"*"+j+" = "+mul);
```

(다른 실습이 더 나아 보여서 패스)

```
8*8 = 64
8*9 = 72
9*1 = 9
9*2 = 18
9*3 = 27
9*4 = 36
9*5 = 45
9*6 = 54
9*7 = 63
9*8 = 72
9*9 = 81
```

첫 실습: ProjectEuler

- ProjectEuler: 코딩+수학 퍼즐 사이트
  - 앞 몇 문제는 코딩 연습하기에도 적절하다.
- http://projecteuler.net/problem=1
  - 1000 미만의 3이나 5의 배수의 합을 구하는 프로그램을 짜 보자.
  - 배수 여부는 어떻게 확인하지?

```
> sum = 0;
  for(i=1; i<1000; i++){
    if(i%3==0 || i%5==0) sum += i;
  }
  console.log(sum)
  233168
< undefined
```

break와 continue

- break: 현재 루프를 빠져 나온다.
  - 루프를 더 돌 필요가 없을 때 쓴다.
- continue: 현재 블록을 건너 뛴다.
  - 처리하지 않고 넘어가는 경우가 생길 때 유용하다.
  - for문에서는 블록이 끝난 뒤에 실행 될 코드는 실행한다.

```
> for(i=1;i<=100;i++){
    if(i*i>70) break;
    console.log("i="+i);
    if(i%2==0) continue;
    console.log(i*i);
  }
i=1
1
i=2
i=3
9
i=4
i=5
25
i=6
i=7
49
i=8
< undefined
```

라벨 있는 break와 continue

- break와 continue는 루프 한 단계만을 빠져 나온다.
  - 여러 단계를 다 빠져 나오려면 플래그 변수를 하나 만들어서….
- 반복문과 switch문 앞에 라벨을 붙일 수 있다.
  - `outer_loop: while(){…}`
- 이 라벨을 지정하면 루프 여러 단계를 빠져나올 수 있다.
  - `break outer_loop;`
- 많이 쓰이지는 않는다.
  - 참고로 Java에서도 쓸 수 있다.

```
> for(var i=0,j;i<5;i++){
    switch(i){
      case 3: break;
    }
    console.log(i);
  }
  0
  1
  2
  3
  4
< undefined
```

```
> asdf: for(var i=0,j;i<5;i++){
    switch(i){
      case 3: break asdf;
    }
    console.log(i);
  }
  0
  1
  2
< undefined
```

배열과 객체

메모리를 최대한 간단하게 소비하는 법

타일 여러 개

- 게임 “2048”
  - 자바스크립트로 짜여졌다.
  - 타일은 어떻게 관리할까?
  - ```
    tile_11 = …;
    tile_12 = …;
    tile_13 = …;
    …
    ```
  - 아무리 생각해도 이건 아닌 것 같은데….
- 비슷한 것 여러 개를 묶을 수 있으면…


    ```

배열

- 비슷한 것 여러 개를 한꺼번에 다룰 때 쓸 수 있다.
  - `a = [1,2,3,4];`
  - 자바나 C와는 다르게, 타입이 같을 필요는 없다.
    - `b = [true, "LOL", 42, [false, -1]];`
    - 배열 안에 배열이 올 수도 있다.
- 배열의 (i+1)번째 원소: `a[i]`
  - 인덱스는 0부터 시작한다!
    - 거의 모든 프로그래밍 언어가 이런다.
  - 문자열에도 쓰일 수 있다.
- 배열의 길이: `a.length`
  - 반복문과 같이 사용하면 좋다!
  - 이것도 역시 문자열에 있다.

```
> a=[1,4,1,5,9,2];
  i=0;
  while(i<a.length){
    console.log("원주율의 "+(i+1)+
      "번째 자리: "+a[i]);
    i++;
  }
  원주율의 1번째 자리: 1
  원주율의 2번째 자리: 4
  원주율의 3번째 자리: 1
  원주율의 4번째 자리: 5
  원주율의 5번째 자리: 9
  원주율의 6번째 자리: 2
< 5
```

팁: 코드를 n번 실행시키고 싶을 때

- `for(i=0;i<n;i++)…`
  - 보통 쓰이는 형태.
  - n에 배열의 길이를 넣으면 각 배열의 원소에 접근할 수 있다.
- `for(i=1;i<=n;i++)…`
  - 잘 쓰이지는 않는다.
  - 0부터 시작하지 않고 1부터 i를 시작하고 싶을 때 쓸 수 있다.
- `for(i=n;i-->0;)…`
  - 한눈에 i가 0으로 간다는 게 보이잖아!
  - 특정 상황에서는 이게 더 나을 수 있다.
- ~~`for(i=0;++i<=n;)…`~~

Object

- 파이썬의 dict, 자바의 Map과 비슷한 존재.
  - 문자열 하나당 값 하나가 할당된다.
  - 배열처럼 아무거나 넣을 수 있다.
    - 객체 안에 배열을, 배열 안에 객체를 넣을 수도 있다.
    - 심지어 자기 자신을 넣을 수도 있다(!!)
- 객체를 표시할 때
  - {} 사이에 키:값과 같은 식으로 항목을 넣고, 항목 사이는 쉼표로 둔다.
  - JSON은 자바스크립트에서 배열과 객체 표기법을 따른다.
    - 다만, 키 값은 항상 쌍따옴표로 감싸야만 하는 등의 몇몇 차이가 있다.

```
> o={
    hello:12,
    'foo bar':34
  };
  Object {hello: 12, foo bar: 34}
> o.hello
  12
> o['foo bar']
  34
> o['asdf'] = 42;
  42
> o.asdf
  42
```

“원래 이런 거에요” x 3

- 자바스크립트에서 배열은 사실 객체의 일종이다.
  - a.b는 a['b']와 같다.
  - 배열은 a['0'], a['1'], …같은 속성값들이 있는 객체다.
    - a[0], a[1]에서 숫자는 (명세상으로) 문자열로 변환된다.
  - 배열에서 양의 정수가 아닌 속성 이름은 무시된다.
    - 아, 물론 a.length === a['length']지만.
- 그렇다고 배열과 객체가 완전히 같다는 건 아니다.
  - length 속성에 값을 설정하면 배열의 길이가 “조절”된다.

```
> a = [1,2,3];
[1, 2, 3]
> a[NaN] = 42
42
> a.NaN
42
> a[-1] = 15
15
> a[-1]
15
> a.hello = 'world'
"world"
> a.hello
"world"
```

“원래 이런 거에요” x 4

- 다음 식의 값들을 예측해 보자.
  - `[]+[]`
  - `[]+{}`
  - `{}+[]`
  - `{}+{}`
  - `[]+[,,]`

- 타입이 막 섞이면 정말 무섭다.

아직도 뭔가 재미있는 걸 만들 수 있을 것 같지는 않은데…

Function

문과와 이과를 가르는 말

함수

- 입력을 받아 특정 동작을 수행하고 값을 반환한다.
  - 입력: 인자 (argument)
  - 동작: 블록 안에 적는다.
  - 반환: return으로 한다.

- 수학에서의 함수와는 다르다!
  - 입력값이 같아도 출력값이 다를 수 있다.
  - 특정한 동작을 수행하는 일을 주로 한다.
  - ~~이게 싫으면 하스켈 써.~~

```
> i=27;
function f(){
  if(i%2==0) i/=2;
  else i=3*i+1;
}
while(i!=1){
  console.log(i);
  f();
}
27
82
41
124
62
31
94
47
```

return

- 함수에서 값을 반환해 외부에서 쓸 수 있게 한다.
- 함수에서 42를 반환: return 42;
  - 함수에서 빠져 나오고, 함수를 호출한 쪽에 값을 반환한다.
- 이미 존재하는 몇몇 함수들
  - Math.random()
    - 0 이상 1 이하 난수를 반환한다.
  - Date.now()
    - 1970년 1월 1일 자정부터 흐른 시각을 ms단위로 반환한다.
  - prompt()

```
> i=1;
function foo(){
  return i*5+1;
}
while(i<5){
  console.log(foo()*2);
  i++;
}
12
22
32
42
```

null과 undefined

- undefined와 null은 빈 값을 뜻한다.
  - undefined는 변수에 값이 할당되지 않은 것을 뜻한다.
    - return이 없는 함수의 반환 값
    - 정의만 하고 값을 대입하지 않은 변수
    - 객체의 없는 속성에 접근할 때
  - null은 변수가 “빈 값”이라는 걸 명확하게 표현할 때 쓴다.
    - 객체를 찾아 반환하는 함수에서 객체를 찾지 못했을 때
    - 비어도 되는 값에 빈 값을 넣을 때
  - undefined == null이지만 undefined !== null이다.

함수 인자

- 함수에 인자를 넣을 수 있다.
  - 값을 다르게 하면서 같은 일을 여러 번 할 수 있게 한다.
  - 호출할 때 값을 넣어주지 않으면?
- 인자를 받는 몇몇 함수들
  - console.log
    - console 객체의 log 속성이 함수다.
    - 인자로 무언가를 넣으면 그걸 출력 해 준다.
  - parseInt
    - 문자열을 정수로 바꿔준다.
  - Math.sqrt
    - 주어진 수의 제곱근을 구해준다.
  - alert, prompt

```
> function fibonacci(n){
    if(n<2) return n;
    return fibonacci(n-1)+fibonacci(n-2);
  }
  for(i=4;i<10;i++){
    console.log("F("+i+") = "+fibonacci(i));
  }
  F(4) = 3
  F(5) = 5
  F(6) = 8
  F(7) = 13
  F(8) = 21
  F(9) = 34
< undefined
```

스코프

- 함수 인자로 쓰인 변수들은 바깥쪽과는 다른 변수이다.
  - 안쪽에서 아무리 바꿔봤자 바깥쪽에 영향을 주지는 않는다.
    - (객체 제외)
- 스코프는 함수 블록 단위로 생긴다.
  - C와 Java와는 다르게 그냥 블록 단위가 아니다.
- 제일 바깥쪽은 글로벌 스코프다.
  - 지금까지 쓴 모든 변수들은 글로벌 스코프에 존재한다.
  - 글로벌 스코프는 최대한 안 사용하는 것이 좋다.
    - 변수 이름이 겹치면 상당히 짜증나는 일이 일어난다.
  - 그러면 함수 스코프는 어떻게 쓸 수 있지?

```
> i=10;
  j=20;
  function f(i){
    i++; j++;
  }
  f(i);
  console.log(i,j);
  f(j);
  console.log(i,j);
  10 21
  10 22
<- undefined
```

var

- `var x=1, y=2, z;`
  - x, y, z를 이 var문이 있는 함수의 스코프에 선언한다.
    - x, y는 각각 1, 2로, z는 undefined로
  - var를 사용하지 않으면 바깥 스코프의 x, y, z를 쓰게 된다.
- 함수의 로컬 변수를 쓸 때에는 var를 쓰는 습관을 기르자!
  - 자주 쓰는 변수 이름이 충돌되면 상당히 짜증나는 일이 벌어진다.
  - 외부 스코프에 값을 쓰지 않는 게 더 깔끔하다.
    - 예전 웹 브라우저에서는 로컬 변수에 접근하는 시간보다 외부 변수에 접근하는 시간이 더 걸린다. (〈자바스크립트 성능 최적화〉, 한빛미디어)
  - ~~C나 Java에 익숙한 사람에게 정신적 위안을 준다.~~

두 번째 실습: 이차방정식의 근

- 세 인자 a, b, c를 받는 함수 root를 만들자.
  - $ax^2+bx+c=0$의 근을 출력한다.
    - 중근이면 하나만, 아니면 두 개 다.
    - 판별식은 로컬 변수를 만들어서 그 곳에 저장하자.
  - 근이 없으면 (실근이 아니면) 없다고 출력해준다.
  - 근이 무수히 많으면 무수히 많다고 출력해준다.
  - 근이 하나 또는 두 개일 때에는 true를,
    아니면 false를 반환하자.

```
> root(1,0,0)
  -0
< true
> root(1,2,1)
  -1
< true
> root(3,4,5)
  실근이 존재하지 않음!
< false
> root(3,-10,2)
  0.21370035215310867 3.119632981180225
< true
> root(0,5,2)
  -0.4
< true
```

```
function root(a, b, c){
    if(a == 0){
        // 일차방정식
        if(b == 0){
            if(c == 0) console.log("근이 무수히 많음!");
            else console.log("근이 존재하지 않음!");
            return false;
        }
        console.log(-c/b);
        return true;
    }

    var D = b*b - 4*a*c;
    var x = -b/(2*a);
    if(D > 0){
        // 근이 두 개
        D = Math.sqrt(D)/(2*a);
        console.log(x-D, x+D);
        return true;
    }else if(D == 0){
        // 중근
        console.log(x);
        return true;
    }else{
        console.log("실근이 존재하지 않음!");
        return false;
    }
}
```

Call by value / reference

- Call-by-value
  - 함수 안에서 인자 값을 바꿔도 밖에 영향을 주지 않는다.
- Call-by-reference
  - 함수 안에서 인자 값을 바꾸면 밖에도 영향을 준다.
- 자바스크립트와 자바에서 함수는 call-by-value이다.
  - 함수 안에서 인자의 값을 바꿔도 함수 밖에서는 변화가 없다.
    - number, string, boolean, null, undefined
  - 오브젝트 값을 바꾸면 밖에서도 바뀌는 것 같은데…?
    - 함수를 호출할 때 레퍼런스를 call-by-value로 넘겨주기 때문이다.
    - 인자 자체에 다른 값을 대입해보면 알 수 있다.
    - 인자 자체의 “값”은 아니지만, 그래도 내부 값을 바꾸면 외부에 있는 값도 바뀌므로 조심!

```
> function f(a,b,c,d){
    a = 42;
    b = a+b;
    c.foo = 'bar';
    d = c;
  }

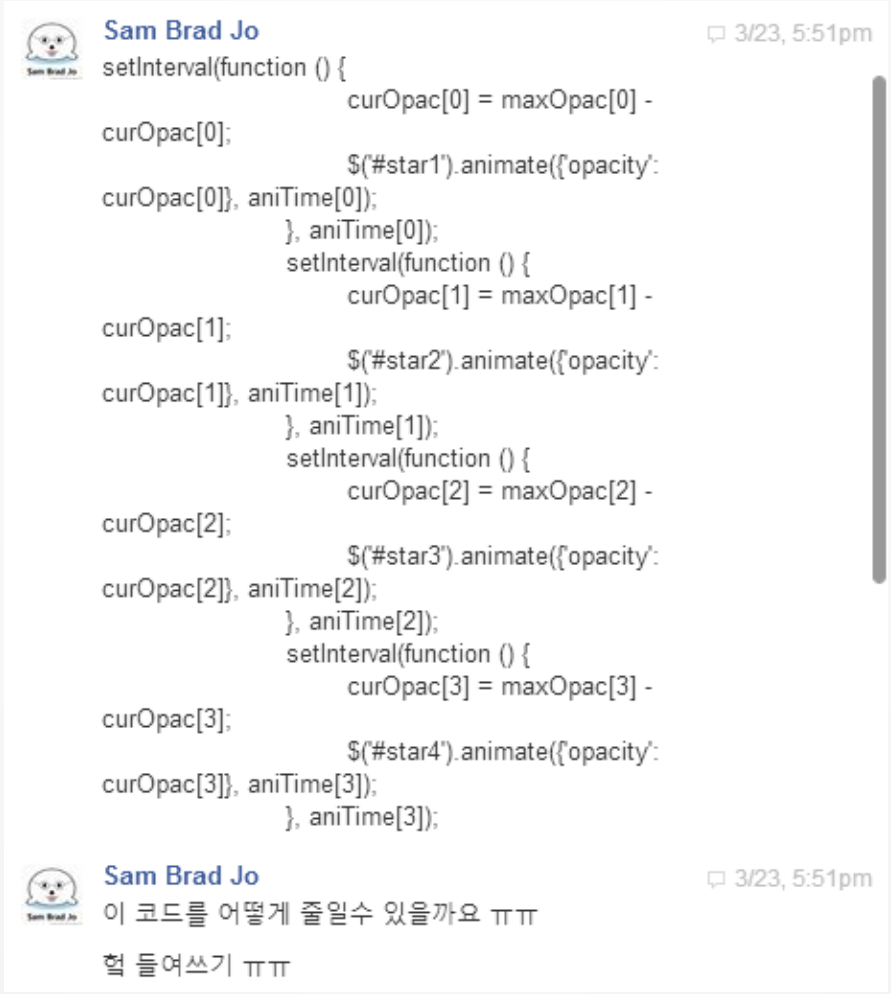
  var test1 = 10,
      test2 = 20,
      test3 = {'blah': 'a', 'foo': 'b'},
      test4 = {'hello': 'world'};

  f(test1, test2, test3, test4);
  console.log("Test1:", test1);
  console.log("Test2:", test2);
  console.log("Test3:", test3);
  console.log("Test4:", test4);
  Test1: 10
  Test2: 20
  Test3: Object {blah: "a", foo: "bar"}
  Test4: Object {hello: "world"}
< undefined
```

자바스크립트에 내장된 객체와 함수

- Math
  - 수학 연산과 관련된 함수들이 한가득 들어 있다.
    - `Math.sin, Math.cos, Math.exp, Math.sqrt, Math.floor, …`
- Number
  - 숫자 자체와 관련된 상수들이 있다.
    - `Number.MAX_VALUE, Number.MIN_VALUE, Number.NaN, …`
  - 숫자 자체에도 각종 메소드가 있다.
    - `x.toString, x.toFixed, x.toExponential`
- String
  - `String.fromCharCode`
  - 역시 문자열 자체에도 각종 메소드가 있다.

일급 함수

함수와 함께 춤을

일급 함수

- 일급 시민
  - 인자로 넘기거나, 변수에 할당되거나, 함수에서 리턴될 수 있는 놈들.
- 자바스크립트의 함수는 일급 시민이다!
  - 함수를 변수에 할당할 수 있다.
  - 함수에 함수를 인자로 넣을 수 있다.
  - 함수에서 함수를 리턴할 수 있다.
    - 함수에서 함수를 리턴하는 함수를 리턴할 수도 있다.
      - 함수를 인자로 받는 함수에서 함수를 인자로 받아 함수를 반환하는 함수를 반환할 수도…

으아아아

일급 함수: 예시

- twice
  - 함수 인자와 값을 받는다.
  - 인자로 받은 함수를 두 번 실행시킨다.
- twice(duplicate, 'hi')
  - duplicate(duplicate('hi'))랑 같다.
  - duplicate('hihi')
  - 'hihihihi'

```
> function twice(f,x){
    return f(f(x));
  }
  function duplicate(x){
    return x+x;
  }
  console.log(typeof duplicate);
  console.log(twice(duplicate, 'hi'));
  function
  hihihihi
<- undefined
```

함수 선언과 함수식

- 함수식: `var x = function(a,b,c){…};`
  - `function x(a,b,c){…}`랑 유사하다.
  - 변수 x에 함수 `function(a,b,c){…}`을 대입한다.

- 거추장스럽게 함수를 각각 정의하지 않아도 된다.
  - 매번 함수 선언을 하고 그 이름을 쓰는 것 보다 낫다.
  - 함수를 인자로 넘겨줄 때 많이 쓴다.
    - `[8,2,1,4,8].map(function(x){return x*2;});`
  - 새로 함수를 만들어 반환할 때에도 많이 쓴다.

일급 함수: 다른 예시

- twice
  - 함수 인자를 하나 받는다.
  - 함수를 자기자신과 합성한 함수를 반환한다.
- twice(twice)(duplicate)
  - twice(twice)는 twice를 두 번 적용하는 함수를 반환한다.
    - twice(twice)(x)는 twice(twice(x))
  - twice(twice)(duplicate)
    - twice(twice)(duplicate)(x)는 duplicate를 4번 실행한 것과 같다.

```
> function twice(f){
    return function(x){
      return f(f(x));
    };
  }
  function duplicate(x){
    return x+x;
  }
  console.log(typeof duplicate);

  var quadruple = twice(duplicate);
  console.log(quadruple('hi'));
  console.log(twice(twice)(duplicate)('hi'));
function
hihihihi
hihihihihihihihihihihihihihihihi
<- undefined
```

함수를 인자로 받는 함수들의 예

- 배열의 forEach, map, filter
  - 인자 하나를 받는 함수를 인자로 받는다.
  - forEach: 함수 각각의 원소에 대해 주어진 함수를 실행한다.
  - map: forEach랑 비슷하지만, 함수 반환값들로 이루어진 새 배열을 만든다.
  - filter: 함수가 true를 반환하는 원소들로만 이루어진 새 배열을 만든다.
- setInterval, setTimeout
  - 함수 하나와 숫자 하나를 인자로 받는다.
  - setInterval(f,x): 함수 f를 x 밀리초마다 실행한다.
  - setTimeout(f,x): x 밀리초 뒤에 함수 f를 실행한다.
- ...등등 상당히 많다.

“원래 이런 거에요” x 5

- 다음 코드를 실행하면 어떻게 될까?
  - ```
    hello1("world");
    hello2("world");

    function hello1(s){
      console.log("Hello, "+s+"!");
    }
    var hello2 = function(s){
      console.log("Hello, "+s+"!");
    };
    ```

- 함수 선언은 순서에 상관 없이 먼저 정의된다.
    ```

“for문이 안 먹혀요 ㅠㅠ”



Sam Brad Jo 3/23, 5:51pm

```
setInterval(function () {
                curOpac[0] = maxOpac[0] -
curOpac[0];
                $('#star1').animate({'opacity':
curOpac[0]}, aniTime[0]);
            }, aniTime[0]);
            setInterval(function () {
                curOpac[1] = maxOpac[1] -
curOpac[1];
                $('#star2').animate({'opacity':
curOpac[1]}, aniTime[1]);
            }, aniTime[1]);
            setInterval(function () {
                curOpac[2] = maxOpac[2] -
curOpac[2];
                $('#star3').animate({'opacity':
curOpac[2]}, aniTime[2]);
            }, aniTime[2]);
            setInterval(function () {
                curOpac[3] = maxOpac[3] -
curOpac[3];
                $('#star4').animate({'opacity':
curOpac[3]}, aniTime[3]);
            }, aniTime[3]);
```

Sam Brad Jo 3/23, 5:51pm

이 코드를 어떻게 줄일수 있을까요 ㅠㅠ

헐 들여쓰기 ㅠㅠ

```
setTimeout(function(){
    console.log("i = 0");
}, 1000);
setTimeout(function(){
    console.log("i = 1");
}, 2000);
setTimeout(function(){
    console.log("i = 2");
}, 3000);
setTimeout(function(){
    console.log("i = 3");
}, 4000);
```

```
for(var i=0;i<4;i++){
    setTimeout(function(){
        console.log("i = "+i);
    }, (i+1)*1000);
}
```

스코프를 신경 써야 할 때

- “어 이거 반복문을 쓰면 간단히 만들 수 있을 것 같은데...”
  - 함수 내부의 i는 그 함수 자체의 로컬 변수가 아니다.
    - 반복문의 i값을 참조하게 된다.
  - setTimeout을 쓰면 함수가 바로 실행되지 않는다.
    - 딜레이로 주어진 값만큼의 시간이 지났을 때야 비로소 실행된다.
    - 또한, 자바스크립트는 기본적으로 단일 스레드 상에서 실행되기 때문에,
      아래에 시간이 오래 걸리는 코드라 할 지라도 코드 실행이 한 번 다 끝나기 전까지
      setTimeout으로 실행 할 코드는 실행되지 않는다.
  - 반복문을 빠져 나왔을 때 i의 값은 4다.
    - setTimeout때에는 딜레이 값이 제대로 계산되어 제 시간에 메시지가 뜨기는 한다.
    - 하지만 이 때 i의 값은 4이므로...
- 어떻게 해야 될까?

클로져

- 함수(A)가 함수(B)를 리턴하는 상황을 생각 해 보자.
  - A의 변수는 B에서 접근할 수 있다.
  - A의 변수는 외부에서 접근하지 못한다.
  - B를 여러 번 실행하면 A의 변수가 바뀐다.
    - A의 실행이 이미 끝났다 해도!
- 클로져는 이런 상황을 뜻한다.
  - 함수 A에서 함수 B를 반환.
  - B에서는 로컬 변수가 아니면서 A의 스코프에 속한 변수를 가지고 있음.
    - 외부에서 접근하지 못 하는 변수

```
> function getCounter(){
    var i=0;
    return function(){
      return i++;
    }
  };

  var counter = getCounter();
  console.log(counter());
  console.log(counter());
  console.log(counter());
  0
  1
  2
< undefined
```

```
setTimeout(function(){
    console.log("i = 0");
}, 1000);
setTimeout(function(){
    console.log("i = 1");
}, 2000);
setTimeout(function(){
    console.log("i = 2");
}, 3000);
setTimeout(function(){
    console.log("i = 3");
}, 4000);
```

```
for(var i=0;i<4;i++){
    var f = function(x){
        return function(){
            console.log("i = "+x);
        };
    };
    setTimeout(f(i), (i+1)*1000);
}
```

(제작중)

이 뒤의 내용은 만들다 말았습니다.

프로토타입

객체 지향적으로 놀아 보자!

new Object

- 함수로 새 객체를 만들 수 있다.
  - 이때 이 함수는 생성자(constructor)가 된다.
  - this: 객체 자기 자신을 가리키는 키워드.
  - new Foo()를 하면 Foo에서 return this를 한 것마냥 새 객체가 나온다.
- 여러 객체를 한꺼번에 다루기 편하다.
  - `this.foo = function(){…}`
  - 메소드를 한 번 정의하면 여러 번 쓸 수 있다.
  - 관리하기도 편하다.

```
> function Foo(x){
      this.bar = x;
      this.baz = "42";
      return 123;
  }

  var x = Foo(1);
  var y = new Foo(2);

  console.log(x);
  console.log(y.bar);
  console.log(y.baz);
  123
  2
  42
< undefined
```

prototype

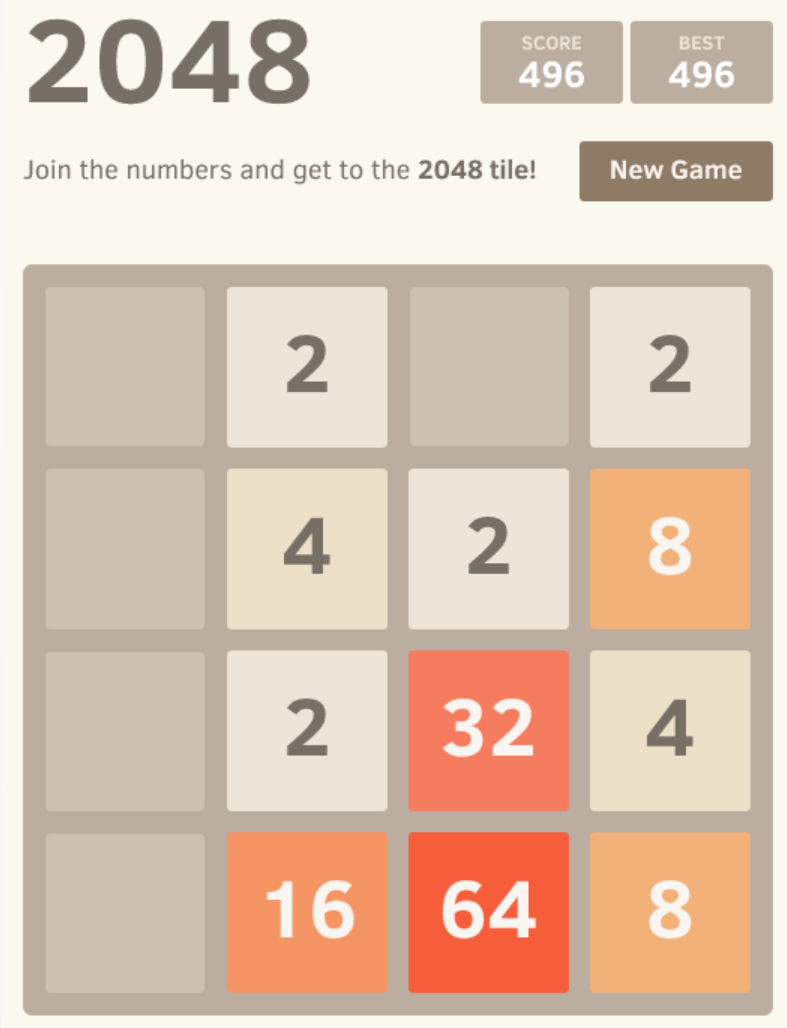
- 객체의 “틀”의 역할을 한다.
  - Java의 class와 비슷한 면이 있음.
  - String.prototype, Array.prototype, …
- Prototype에 있는 함수는 객체에 직접 정의한 함수처럼 쓸 수 있다.

bind, call, apply

- 함수에 쓸 수 있는 메소드
  - bind: 함수에 this나 인자를 bind한다.
  - call: 함수를 호출한다.
  - apply: 함수를 호출한다. (call과 달리 인자를 배열로)

- 여러 군데에서 유용하게 쓸 수 있다.
  - this를 다른 것으로 바꾸고 싶을 때 (예: Array.prototype.slice)
    - this를 "잊어버리게" 하고 싶지 않을 때
  - 앞에 인자를 몇 개 지정해줄 때
  - 동적인 길이의 인자를 넘겨주고 싶을 때

실습 (A): 2048


2048
SCORE
496
BEST
496
Join the numbers and get to the 2048 tile!
New Game
2
2
4
2
8
2
32
4
16
64
8

2048 구조

- polyfill
  - 모든 함수를 모든 웹 브라우저에서 지원하지 않는다.
    - 예: bind, requestAnimationFrame, …
  - 자바스크립트로 비슷하게나마 지원하게 해 주는 게 polyfill이다.
- 모듈
  - tile.js, grid.js
    - tile.js에는 타일 클래스가, grid.js에는 그리드를 관리하는 그리드 클래스가 있다.
  - keyboard_input_manager.js
  - local_storage_manager.js
  - html_actuator.js
  - game_manager.js
    - 실제 게임 처리가 이루어 지는 곳이다.

게임 로직 바꾸기

- GameManager.prototype.move
  - 타일을 움직일 수 있으면 움직인다.
    - buildTraversals로 반복 방향을 간편하게 정한다.
    - grid.cellContent({x:?, y:?})로 그리드의 한 칸을 가져온다.
    - tile.mergedFrom: 어느 타일들이 합쳐져 만들어 졌는지 저장된다.
  - 로직을 다양하게 바꿔 보자.
    - tile.value * 2 대신 tile.value + 1을 쓰면?
    - 타일이 합쳐지는 규칙을 바꿔 보면?

실습 (B): 캔버스

- 미리 정의된 canvas 함수와 속성들
  - canvas.moveTo(x,y)
    - 펜을 x,y로 이동한다.
  - canvas.lineTo(x,y)
    - 펜을 x,y까지 긋는다.
  - canvas.lineWidth, canvas.strokeStyle
    - 각각 선 너비와 선 스타일(색)이다.
  - canvas.stroke()
    - 선을 긋는다.

레퍼런스 / 참고

- Wikipedia
  - JavaScript: https://en.wikipedia.org/wiki/JavaScript
  - ECMAScript: https://en.wikipedia.org/wiki/ECMAScript
- ECMAScript® Language Specification
  - http://www.ecma-international.org/ecma-262/5.1/
- ProjectEuler
  - Problem 1: https://projecteuler.net/problem=1
- WAT
  - https://www.destroyallsoftware.com/talks/wat
- 2048
  - http://gabrielecirulli.github.io/2048/
  - https://github.com/gabrielecirulli/2048