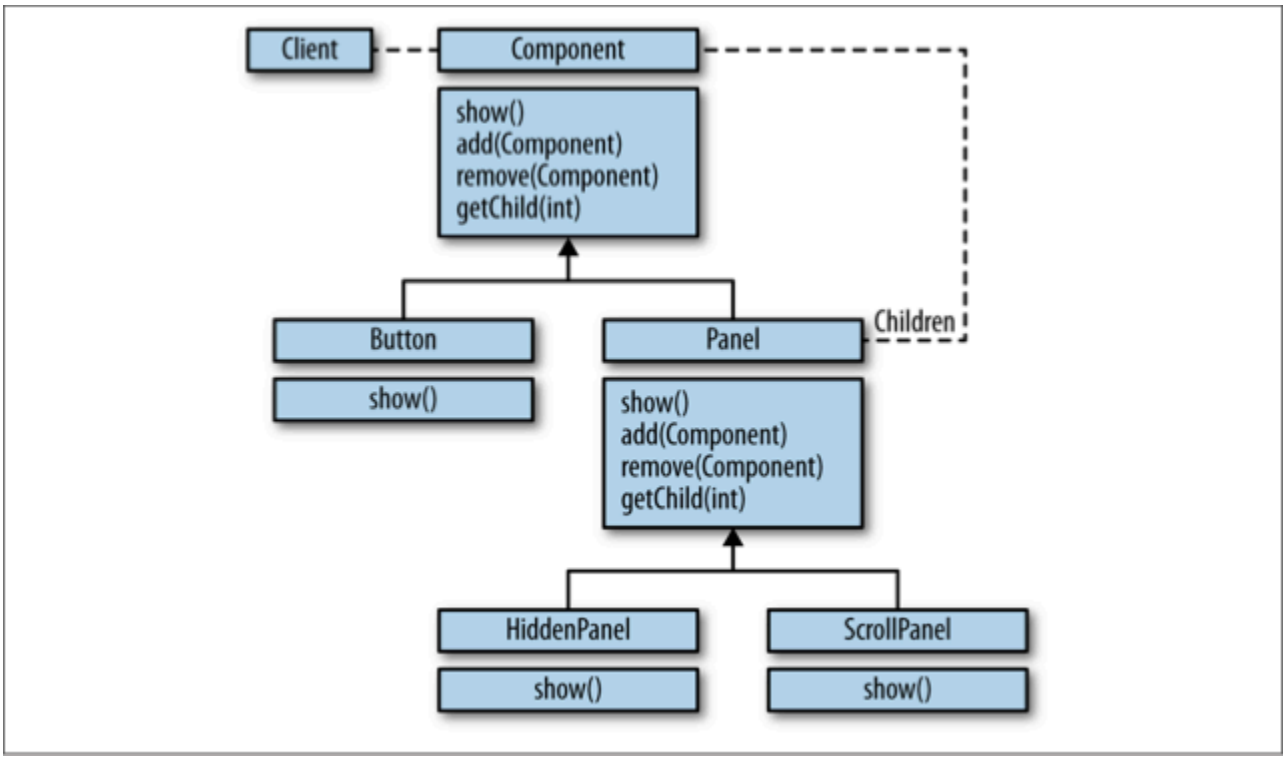
"함수형 자바스크립트" 겉핥기

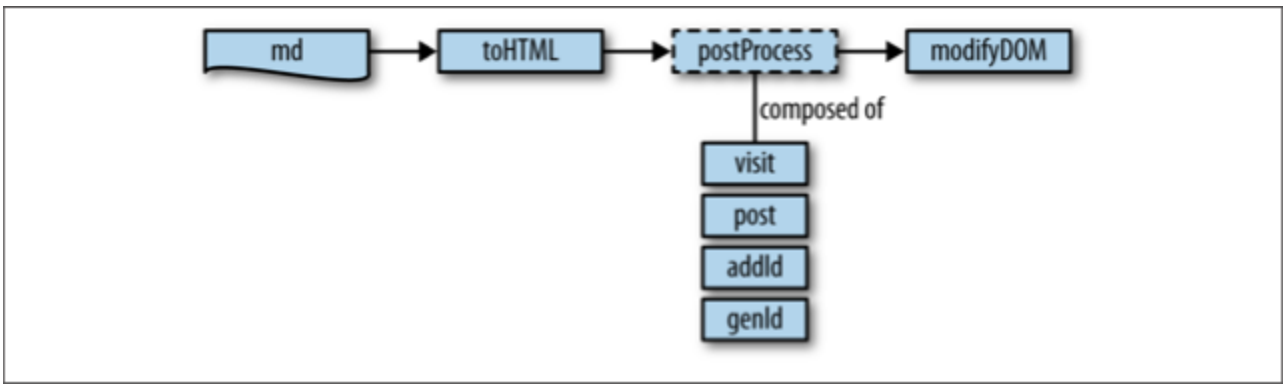
개봉 영화 예고편 보는 느낌으로 봐주세요.

문제를 다루는 방법

객체 지향은 문제를 명사나 객체의 집합으로 나눈다.



함수형은 문제를 동사나 함수의 그룹으로 나눈다



일급 함수

함수형 프로그래밍의 정의를 간단히 '일급 함수'와 '편의성'이라는 두 용어로 요약할 수 있다.
'일급'이라는 용어에는 모든 것을 값으로 취급한다는 의미가 내포되어 있다.

```
/* 숫자를 변수에 저장하듯이 함수를 변수에 저장할 수 있다. */
var fortytwo = function() { return 42 };
```

```
/* 숫자를 배열에 저장하듯이 함수를 배열에 저장할 수 있다. */
var fortytwos = [42, function() { return 42 }];
```

```
/* 숫자를 객체에 저장하듯이 함수를 객체에 저장할 수 있다. */
var fortytwos = {number: 42, fun: function() { return 42 }};
```

```
/* 언제든 숫자를 만들 수 있듯이 필요할 때 함수를 만들 수 있다. */
42 + (function() { return 42 })(); //=> 84
```

```
/* 함수에 숫자를 전달할 수 있듯이 함수에 함수를 전달할 수 있다. */
function weirdAdd(n, f) { return n + f() } weirdAdd(42, function() { return 42 }); //=> 84
```

```
/* 함수가 숫자를 반환할 수 있듯이 함수가 함수를 반환할 수 있다. */
return 42;
return function() { return 42 };
```

클로저를 이용한 추상화

클로저를 이용해서 생성 시에 어떤 '설정'에 따라 다른 함수를 만들 수 있다.

```
function plucker(FIELD) {
  return function(obj) {
    return (obj && obj[FIELD]);
  };
}
```

```
var best = {title: "Infinite Jest", author: "DFW"};

var getTitle = plucker('title');

getTitle(best);
//=> "Infinite Jest"
```

```
var books = [{title: "Chthon"}, {stars: 5}, {title: "Botchan"}];

var third = plucker(2);

third(books);
//=> {title: "Botchan"}
```

```
_.filter(books, getTitle);
//=> [{title: "Chthon"}, {title: "Botchan"}]
```

~~꽃보다 남자~~ 값보다 함수

함수에 값을 넘겨줌으로써 원하는 동작을 구현할 수 있다. 하지만 값 대신 함수를 전달함으로써 기능을 더 일반화시킬 수 있다.

```
function repeat(times, VALUE) {
  return _.map(_.range(times), function() { return VALUE; });
}

repeat(4, "Major");
//=> ["Major", "Major", "Major", "Major"]
```

```
function repeatedly(times, fun) {
  return _.map(_.range(times), fun);
}

repeatedly(3, function() {
  return Math.floor((Math.random()*10)+1);
});
//=> [1, 3, 8]
```

```
repeatedly(3, function() { return "Odelay!"; });
//=> ["Odelay!", "Odelay!", "Odelay!"]
```

```
repeatedly(3, function(n) {
  var id = 'id' + n;
  $('body').append($("<p>Odelay!</p>").attr('id', id));
  return id;
});
```

```
function iterateUntil(fun, check, init) {
  var ret = [];
  var result = fun(init);

  while (check(result)) {
    ret.push(result);
    result = fun(result);
  }

  return ret;
};
```

```
iterateUntil(
  function(n) { return n+n },
  function(n) { return n <= 1024 },
  1
);
//=> [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

```
repeatedly(10, function(exp) { return Math.pow(2,exp+1) });
//=> [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

함수 조립

curry

```
function curry(fun) {
  return function(arg) {
    return fun(arg);
  };
}
```

```
['11','11','11','11'].map(parseInt) //=> [11, NaN, 3, 4]
['11','11','11','11'].map(curry(parseInt)); //=> [11, 11, 11, 11]
```

```
function curry2(fun) {
  return function(secondArg) {
    return function(firstArg) {
      return fun(firstArg, secondArg);
    };
  };
}
```

```
var parseBinaryString = curry2(parseInt)(2);

parseBinaryString("111");
//=> 7

parseBinaryString("10");
//=> 2
```

partial

```
/**
 * _.partial(function, *arguments)
 */


var add = function(a, b) { return a + b; };
add5 = _.partial(add, 5);
add5(10);
=> 15
```

compose

```
/**
 * _.compose(*functions)
 */

var greet = function(name){ return "hi: " + name; };
var exclaim = function(statement){ return statement.toUpperCase() + "!"; };
var welcome = _.compose(greet, exclaim);
welcome('moe');
=> 'hi: MOE!'
```

순수성과 불변성

함수형 프로그래밍은 단지 함수를 다루는 기법이 아니다.
함수형 프로그래밍은 소프트웨어 개발의 복잡성을 최소화하는 개발 방식을 추구한다.
프로그램에서 발생하는 상태 변화를 최소화하거나 아예 없애는 것은 복잡성을 줄일 수 있는 방법 중 하나이다.

순수성

```
var rand = _.partial(_.random, 1);
```

```
rand(10);
//=> 7

repeatedly(10, partial1(rand, 10));
//=> [2, 6, 6, 7, 7, 4, 4, 10, 8, 5]

_.take(repeatedly(100, _.partial(rand, 10)), 5);
//=> [9, 6, 6, 4, 6]
```

```
function randString(len) {
  var ascii = repeatedly(len, _.partial(rand, 36));

  return _.map(ascii, function(n) {
    return n.toString(36);
  }).join('');
}
```

```
randString(0);
//=> ""

randString(1);
//=> "f"

randString(10);
//=> "k52k7bae8p"
```

```
describe("randString", function() {
  it("builds a string of lowercase ASCII letters/digits", function() {
    expect(randString()).to???(???);
  });
});


describe("_.map", function() {
  it(“should return an array made from...”, function(){
    expect(.map([1,2,3], sqr)).toEqual([1,4,9]);
  });
});
```

ⓘ 순수한 함수란?

- 오직 인자만을 이용해서 계산 결과가 만들어진다.
- 외적 요소에 영향을 받는 데이터에 의존하지 않는다.
- 자신의 바디 외부의 상태를 변화시킬 수 없는 구조다.

```
function generateRandomCharacter() {
  return rand(26).toString(36);
}

function generateString(charGen, len) {
  return repeatedly(len, charGen).join('');
}
```

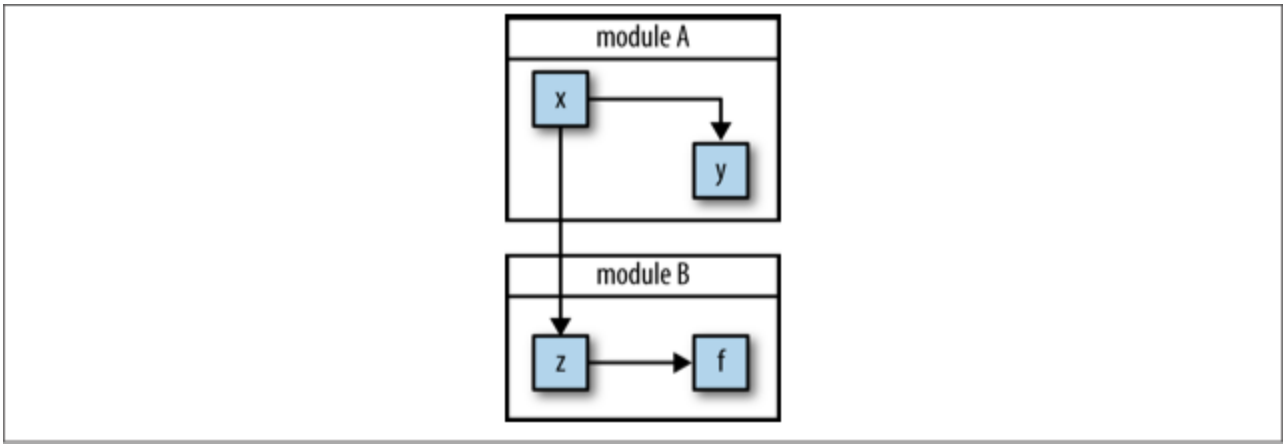
```
generateString(generateRandomCharacter, 20);
//=> "2lfhjo45n2nfnpbf7m7e"
```

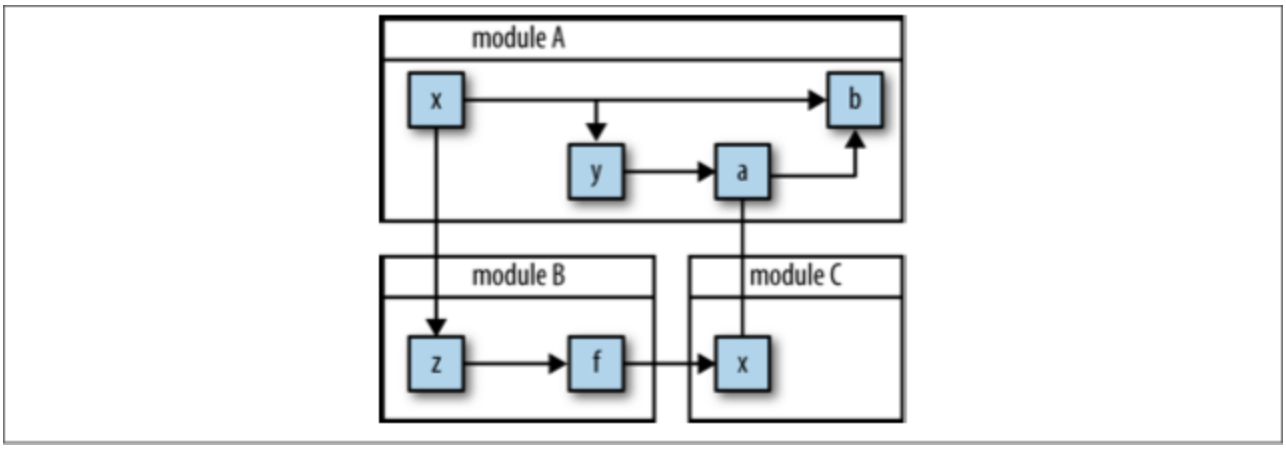
```
var composedRandomString = _.partial(generateString, generateRandomCharacter);

composedRandomString(10);
//=> "j18obij1jc"
```

```
describe("generateString", function() {

  var result = generateString(always("a"), 10);

  it("should return a string of a specific length", function() {
    expect(result.constructor).toBe(String);
    expect(result.length).toBe(10);
  });

  it("should return a string congruent with its char generator", function() {
    expect(result).toEqual("aaaaaaaaaa");
  });

});
```

불변성

함수형 프로그래밍에서는 변이가 일어나지 않는 상황이 가장 이상적이다.
자바스크립트에서는 배열과 객체를 레퍼런스로 전달하므로 진정한 불변성을 갖지 못한다.
마찬가지로 언제든 자바스크립트 객체 필드에 접근할 수 있으므로 객체 필드를 불변으로 만들기도 쉽지 않다.





Object#freeze
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze

```
var a = [1, 2, 3];

a[1] = 42;

a;
//=> [1, 42, 3]

Object.freeze(a);

a[1] = 108;

a;
//=> [1, 42, 3]

Object.isFrozen(a);
//=> true
```

```
var x = [{a: [1, 2, 3], b: 42}, {c: {d: []}}];

Object.freeze(x);

x[0] = "";

x;
//=> [{a: [1, 2, 3], b: 42}, {c: {d: []}}];
```

```
x[1]['c']['d'] = 100000;

x;
//=> [{a: [1, 2, 3], b: 42}, {c: {d: 100000}}];
```

```
function deepFreeze(obj) {
  if (!Object.isFrozen(obj))
    Object.freeze(obj);

  for (var key in obj) {
    if (!obj.hasOwnProperty(key) || !_.isObject(obj[key]))
      continue;
    deepFreeze(obj[key]);
  }
}
```

```
var x = [{a: [1, 2, 3], b: 42}, {c: {d: []}}];

deepFreeze(x);

x[0] = null;

x;
//=> [{a: [1, 2, 3], b: 42}, {c: {d: []}}];

x[1]['c']['d'] = 42;

x;
//=> [{a: [1, 2, 3], b: 42}, {c: {d: []}}];
```

기타

재귀

- 자바스크립트 재귀의 한계와 극복하는 방법
- 꼬리 재귀, 상호 재귀, 게으른 재귀

흐름 기반 프로그래밍

- 체이닝
- 파이프라이닝

클래스를 이용하지 않는 프로그래밍

- 믹스인(Mixin)

아 그거 내가 해봐서 아는데...

```
/*
modules example =>
{
  "a": {},
  "b": {
    "dependency": ["a"]
  },
  "c": {
    "dependency": ["a, b"]
  },
  "d": {
    "dependency": ["x"]
  }
}
*/

function createDependencyMap(modules) {
  return _.object(
    _.keys(modules),
    _.map(modules, function (module) {
      return _.clone(module.dependency) || []
    })
  )
}

/*
map example =>
{
  "a": [],
  "b": ["a"],
  "c": ["a", "b"],
  "d": ["x"]
}
*/

function isBroken(map) {
  return !_.every(
    _.uniq(_.flatten(_.toArray(map))),
    _.partial(_.contains, _.keys(map))
  )
}
```

함수형 "반응" 프로그래밍

Functional reactive programming
Functional reactive programming (FRP) is a programming paradigm for reactive programming using the building blocks of functional programming. FRP has been used for programming GUIs, robotics, and music, aiming to simplify these problems by explicitly modeling time.

- From Wikipedia, the free encyclopedia

Reactive programming
In computing, reactive programming is a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.

- From Wikipedia, the free encyclopedia

- Elm
  - http://elm-lang.org/
  - http://elm-lang.org/learn/What-is-FRP.elm
- RxJS
  - https://github.com/Reactive-Extensions/RxJS
  - http://reactive-extensions.github.io/RxJS/
- Bacon.js
  - https://github.com/baconjs/bacon.js
  - http://baconjs.github.io/
  - http://raimohanska.github.io/bacon.js-slides/index.html
  - http://blog.flowdock.com/2013/01/22/functional-reactive-programming-with-bacon-js/
- Kefir.js
  - https://github.com/pozadi/kefir
  - http://pozadi.github.io/kefir/