

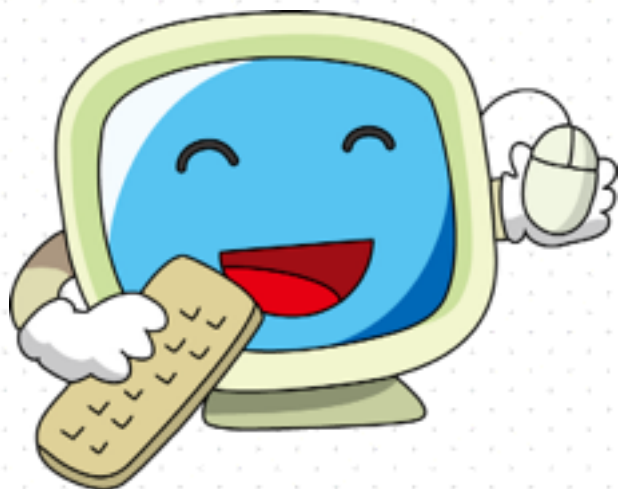
いわーく勉強会

第2回

前回の復習

コンピュータの3大原則

1. **コンピュータ**は、**入力**、**演算**、**出力**を行う装置である。
2. **プログラム**は、**データ**と**命令**の集合体である。
3. コンピュータの都合は、人間の感覚と異なる場合がある。



数字なら任せてよ！

コンピュータの3大原則

2. プログラムは、データと命令の集合体である。

変数と
メソッド

起動したよ！

ボタンが押されたよ！

イベントドリブン

Viewが読み込まれたよ！

綺麗なコード

誰かが読むことを意識する。

4つの原則

1. 簡潔性
2. 明瞭性
3. 一般性
4. 自動化

変数の名前の付け方

スコープの広い変数ほど名前から伝わる情報量を増やす。

- グローバル変数はわかりやすい名前。

```
NSString *navTitle; // ナビゲーションバーに表示するタイトル
```

- ローカル変数は短い名前。

```
MGLLine *line = section.topLines[i];
```

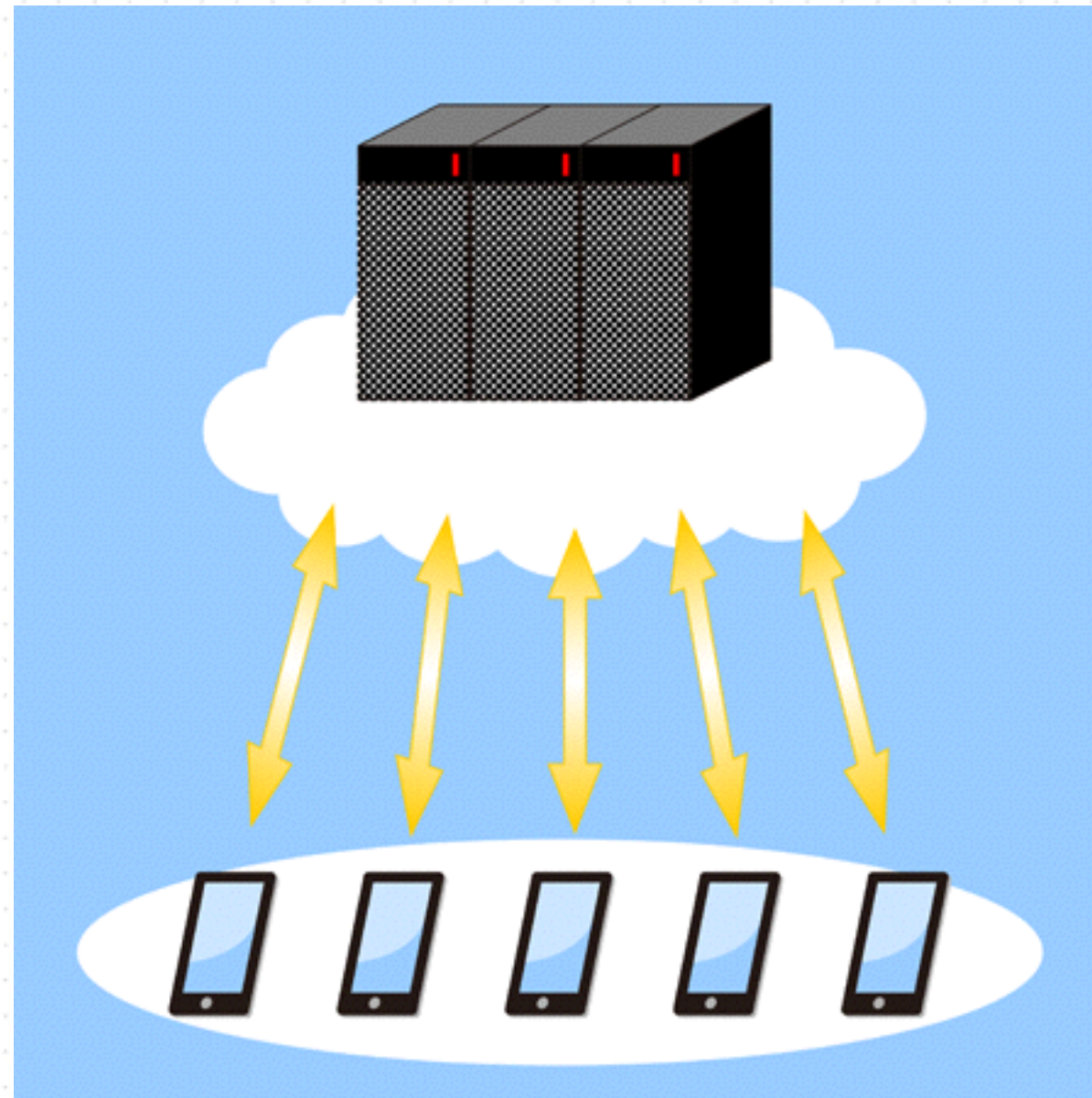
コメントをつけると
なお良い

その他

- インデントを整える
control + i (Xcodeの場合)
- マジックナンバー
- コメント

本日の内容

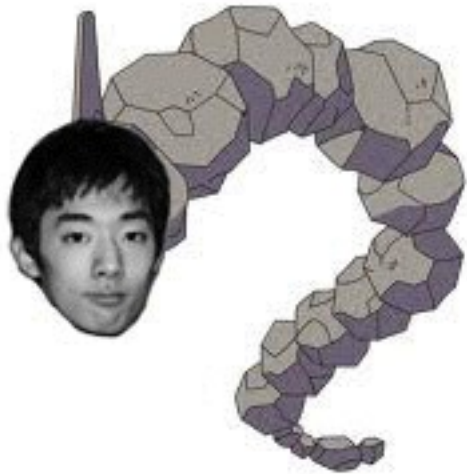
ネット通信



Countアプリも オンライン化の時代



サーバーはどうするんですか？



もちろん作るぞ。

Node.jsで サーバーサイド プログラミング

なぜNode.jsなのか

- ネットワークの基礎を学びやすい。
- 非同期処理が得意で、チャット等のiPhoneアプリと相性が良い。
- セキュリティも自分で考える必要がある。
- Webサーバとアプリケーションの間に垣根が無く、シンプル。

とりあえず作ってみる

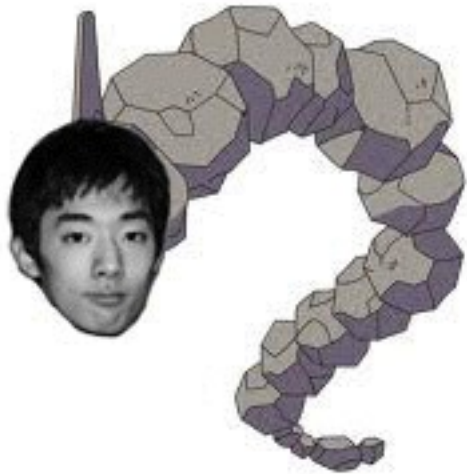
- javascript(以下js)でコードを書く。

```
server.js
1  var http = require('http');
2  http.createServer(function(request, response){
3      response.writeHead(200, {'content-type': 'text/html'});
4      response.end('Hello Node.js!');
5  }).listen(8080);
6  |
```

- 実行する `node server.js`
- ブラウザで localhost:8080にアクセス



できたぜウキキー！



んじゃ説明するよ。

なんか用語とかの説明

```
server.js
1  var http = require('http');
2  http.createServer(function(request, response){
3    response.writeHead(200, {'content-Type': 'text/html'});
4    response.end('Hello Node.js!');
5  }).listen(8080);
6
```

httpモジュール

200 OK

ヘッダ

ポート番号

Content-Type

jsは書きにくい

- jsはシンプルで使いやすいが、
長いコードになると
読みにくい
書きにくい
拡張しづらい
などの問題に出会う。

そこでCoffeeScript



ようやく出番だな！

```
npm install -g coffee-script
```


coffeeとjsの比較

- coffee hello.coffee
- coffee -c hello.coffee

coffee

js

```
server.js  x  hello.coffee  x
1  hello = ->
2  console.log "Hello World!"
3
4  hello()
```

```
server.js  x  hello.coffee  x  hello.js  x
1  // Generated by CoffeeScript 1.6.3
2  (function() {
3    var hello;
4
5    hello = function() {
6      return console.log("Hello World!");
7    };
8
9    hello();
10
11  }).call(this);
12
```

CoffeeScriptの文法①

- 宣言は要らない
- 行末のセミコロンが要らない
- 関数定義は `->` で行う
- インデントでブロックを表現する
- 関数の最後の値が自動的にreturn

宣言が要らない

セミコロンが要らない

```
1 a = 30
2 b = 50
3 console.log a+b
4
```

「->」で関数定義

```
5  # 関数定義
6  hello = -> console.log "hello"
7  # 関数の呼び出し
8  hello()
```


インデントでブロックを表現

```
10 # helloWorld関数の定義
11 ▼ helloWorld = ->
12     console.log "Hello"
13     console.log "World"
14
15 # helloWorld関数の実行
16 helloWorld()
```

最後の値が自動的にreturn

引数をとる関数

```
18  # 2つの数を足す関数
19  add = (a,b) -> a+b
20
21  # add関数の実行
22  sum = add 20,30
23  console.log sum
24
```

実行時、引数があれば
() の省略が可能。

CoffeeScriptの文法②

- 文字列に変数を含むのは #{変数}
- ヒアドキュメント
- 引数に関数を渡す
- 配列、レンジ、ループ
- 条件分岐

文字列に変数を含む

```
25 tommy = "トミー"  
26 console.log "Hello, #{tommy}"  
27 |
```


引数に関数を渡す

```
28  # 引数に(無名)関数を渡す
29  setTimeout ->
30      console.log "1秒後にトミーが面白いことをする"
31      , 1000
32
```

配列、レンジ、ループ

```
33 week = ['日', '月', '火', '水', '木', '金', '土']
34 weekdays = week[1..5]
35 console.log weekdays
36
37 for i in [0..3]
38   console.log "#{week[i]}曜日はわたぬきが面白いことをする"
39
40 weekdays.forEach (wday) ->
41   console.log "#{wday}曜日はトミーが面白いことをする"
42
```

条件分岐

```
43 numbers = [1..20]
44
45 # numbersの中で、3の倍数ならアホになる。
46 # 3の倍数ではなく、4の倍数なら、天才になる。
47 for num in numbers
48   if num % 3 == 0
49     console.log "#{num} アホー。"
50   else if num % 4 == 0
51     console.log "#{num} てんさーい！"
52   else
53     console.log "#{num}"
54
```

CoffeeScriptの文法③

- オブジェクト
- クラス

オブジェクト

```
55 ▼ persons =  
56   tommy:  
57     name: "トミー"  
58     age: 20  
59   watanuki:  
60     name: "ぬきわた"  
61     age: 20  
62  
63 console.log persons.watanuki.name
```

クラス

```
65 ▼ class Person
66 ▼   constructor: (name) ->
67     @age = 0
68     @name = name
69   speak: ->
70     console.log "#{@name} speaks"
71
72   tommy = new Person "トミー"
73   nukiwata = new Person "わたぬき"
74   console.log tommy.name
75   nukiwata.speak()
```

問題

カードゲームを作ろう

- 2人で戦えるカードゲーム。
- それぞれ異なる攻撃力、防御力、速度がある。
両方とも合計は15ポイント、HPは200。
- スピードの早い方が先攻（同じ場合は毎ターンランダムで決まる）。
先攻はそのターン中、攻撃力が0 - 4増える。
- 毎ターン、それぞれ自分の攻撃力 - 相手の防御力のダメージを与え、HPが無くなったら負け。

ヒント①

- 乱数は `Math.random()` で生成できる。
0～1のランダムな数値を取得。
ただし1は含まない。
- なので、0 - 4の整数を得たい場合は、
これを5倍して切り捨てれば良い。
`Math.floor Math.random()*5`

ヒント②

- 以下の2つのクラスを作ると良い。
- ゲームを管理するクラス
- プレイヤーを管理するクラス

さてこれで準備は完了



オンライン Countアプリを作る

MVCカウントアプリ
を作ってください

cocoapod

```
gem install cocoapods
```

```
pod setup
```

Podfileの作成

```
platform :ios, '5.0'
```

```
pod 'AFNetworking', '~> 1.3.0'
```

接続処理

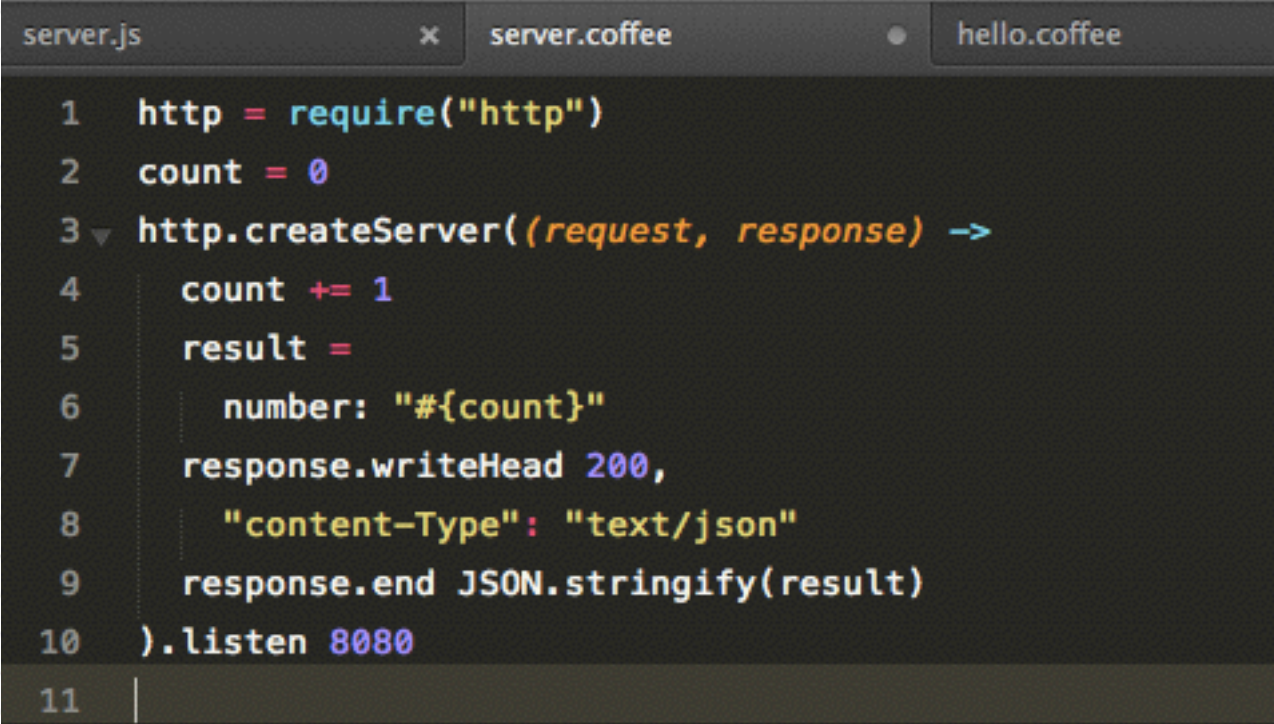
```
AFHTTPClient *client = [AFHTTPClient clientWithBaseURL:[NSURL URLWithString:@"http://localhost:8080"]];
[client registerHTTPOperationClass:[AFJSONRequestOperation class]];
[client setDefaultHeader:@"Accept" value:@"text/json"];
[client getPath:@"" parameters:nil success:^(AFHTTPRequestOperation *operation, id responseObject) {

    label.text = responseObject[@"number"];

} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    NSLog(@"NetworkError:%@",error);
}];
```

server.coffee

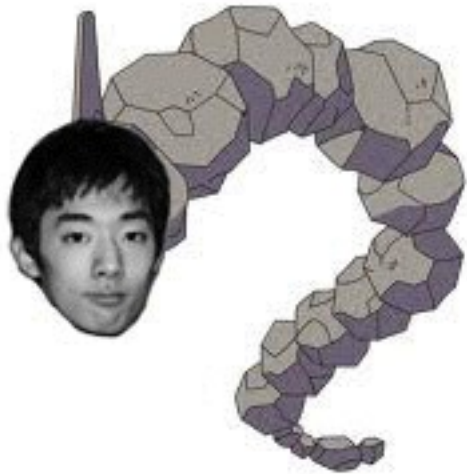
- `npm -g install hotnode`
`hotcoffee server.coffee`

A screenshot of a code editor with three tabs: 'server.js', 'server.coffee' (active), and 'hello.coffee'. The 'server.coffee' tab contains the following code:

```
1 http = require("http")
2 count = 0
3 http.createServer((request, response) ->
4   count += 1
5   result =
6     number: "#{count}"
7   response.writeHead 200,
8     "content-type": "text/json"
9   response.end JSON.stringify(result)
10 ).listen 8080
11
```



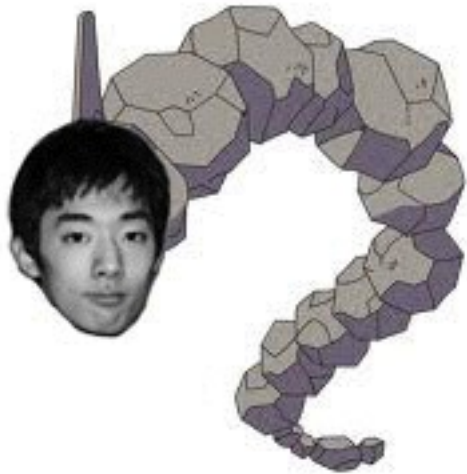

できたぜウキキー！



(資料作り) 疲れた。



けどマイナスとかどうするの？



ルーティングするよ。

server2.coffee

```
1 http = require("http")
2 path = require("path")
3 count = 0
4 pages = [
5   route: 'plus'
6   output: -> count += 1
7   ,
8   route: 'minus'
9   output: -> count -= 1
10  ,
11  route: 'kakeru'
12  output: -> count *= 2
13  ,
14  route: 'waru'
15  output: -> count /= 2
16 ]
17 http.createServer((request, response) ->
18   lookup = path.basename decodeURI(request.url)
19   pages.forEach (page) ->
20     if page.route == lookup
21       page.output()
22       result =
23         number: "#{count}"
24         response.writeHead 200,
25           "content-Type": "text/json"
26         response.end JSON.stringify(result)
27 ).listen 8080
```

ちなみにjsだと



```
// Generated by CoffeeScript 1.6.3
```

```
(function() {  
  var count, http, pages, path;  
  http = require("http");  
  path = require("path");  
  count = 0;  
  pages = [  
    {  
      route: 'plus',  
      output: function() {  
        return count += 1;  
      }  
    }, {  
      route: 'minus',  
      output: function() {  
        return count -= 1;  
      }  
    }, {  
      route: 'kakeru',  
      output: function() {  
        return count *= 2;  
      }  
    }, {  
      route: 'waru',  
      output: function() {  
        return count /= 2;  
      }  
    }  
  ];  
};
```

```
http.createServer(function(request, response) {  
  var lookup;  
  lookup = path.basename(decodeURI(request.url));  
  return pages.forEach(function(page) {  
    var result;  
    if (page.route === lookup) {  
      page.output();  
      result = {  
        number: "" + count  
      };  
      response.writeHead(200, {  
        "content-Type": "text/json"  
      });  
      return response.end(JSON.stringify(result));  
    }  
  });  
}).listen(8080);
```

```
}).call(this);
```

小さすぎて読めない…。

オンライン
Countアプリ完成！

おつかれさまでした