静的型付き関数型言語のススメ

@tmaeda

2013.03.20 Ruby勉強会@札幌

@tmaeda

- (株)アンタスで主にPHPで仕事をしています。
- 関数型は趣味です。OCamlとHaxeが好き。近いうちに仕事で使いたいなぁ。

2013年2月7日

snoozer05 opened this issue a month ago

そろそろ#24の準備をするぞー

Edit

No one is assigned

Milestone: **Ruby勉強会@札幌#24**

思いついたことをつらつらと:

- 募集は http://rubysapporo.doorkeeper.jp/ を使いたい
- news に「勉強会やるよ」ページ(ex: http://ruby-sapporo.org/news/2012/05/11/workshop-23.html)を作る必要があるんだな
    - もっと軽量化していけるとよさそう
- 『初めてのRuby』の読み合わせはいつも通りに
- **@tmaeda** さんに関数型言語の話をしてもらいたいなー(ちらっ)
- 一般発表の募集はイベント案内のページにPR貰えばいいのかなー

2013年2月7日

snoozer05 opened this issue a month ago

そろそろ#24の準備をするぞー

Edit

No one is assigned

Milestone: **Ruby勉強会@札幌#24**

- もっと軽量化していけるとよさそう
- 『初めてのRuby』の読み合わせはいつも通りに
- **@tmaeda** さんに関数型言語の話をしてもらいたいなー(ちらっ)
- 一般発表の募集はイベント案内のページにPR貰えばいいのかなー

@tmaeda さんに関数型言語の話を

2013年2月7日

2013年2月7日

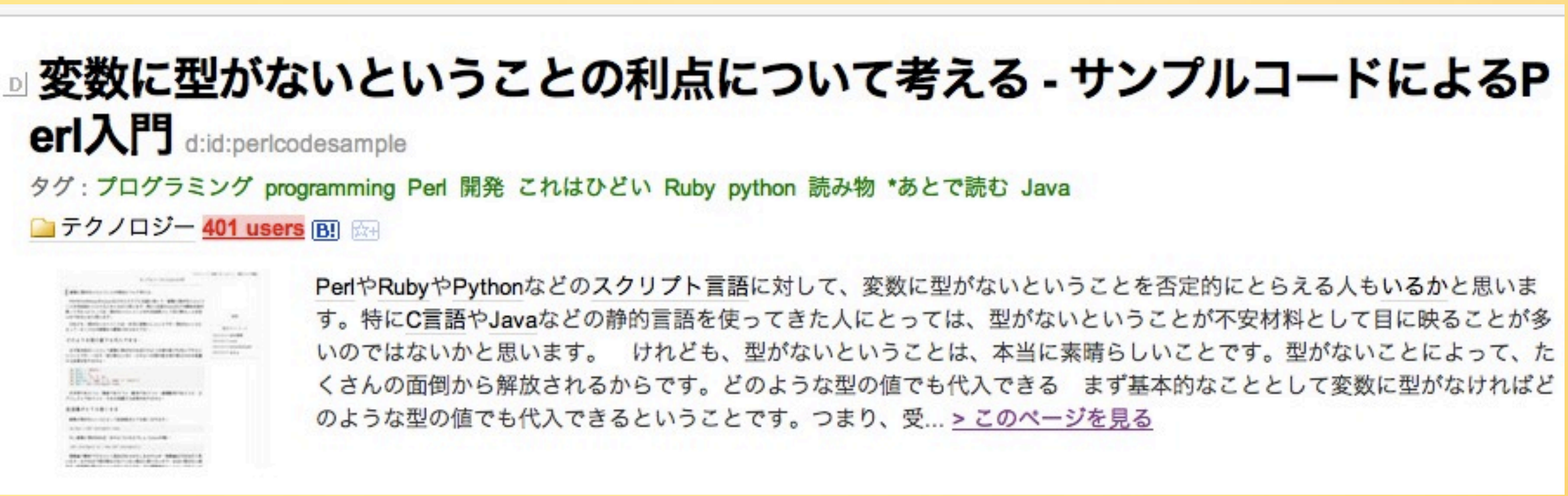
2013年2月7日

2013年2月27日

2013年2月27日

戦いは動的陣営から始まった

2013年2月27日



変数に型がないということの利点について考える - サンプルコードによるPerl入門 d:id:perlcodesample

タグ : プログラミング programming Perl 開発 これはひどい Ruby python 読み物 *あとで読む Java

テクノロジー 401 users B!

PerlやRubyやPythonなどのスクリプト言語に対して、変数に型がないということを否定的にとらえる人もいるかと思います。特にC言語やJavaなどの静的言語を使ってきた人にとっては、型がないということが不安材料として目に映ることが多いのではないかと思います。　けれども、型がないということは、本当に素晴らしいことです。型がないことによって、たくさんの面倒から解放されるからです。どのような型の値でも代入できる　まず基本的なこととして変数に型がなければどのような型の値でも代入できるということです。つまり、受... > このページを見る

http://d.hatena.ne.jp/perlcodesample/20130227/1361928810

2013年2月27日

見事な大炎上

せっかく下火に
なってきたのに、
また蒸し返し
ちゃっていいの？

動的陣営の大ボス@sumimさんも今日の勉強会に参加するというのでドキドキしていたけど、今日の自己紹介によると@sumimさんは今Scalaのtraitや Rubyのrefinementsに夢中らしいので、心配事がひとつだけ減りました♪

この物語は私の関数型の平凡な日常を淡々と描くものです。過度なツッコミはしないでください。

静的型付き関数型言語のススメ

@tmaeda

2013.03.20 Ruby勉強会@札幌

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

関数型プログラミング

- 副作用をなるべく少なく
- 関数を第一級オブジェクトとして扱う(関数が関数を受けとったり、関数が関数を返したり)

副作用？

- 同じ引数を与えれば常に同じ戻り値を返すのが「副作用がない」状態
(数学の「関数」と同じ)
- 副作用
  - 入出力(ファイル、DB、ネットワーク、環境変数など)
  - 破壊的代入

関数型プログラミングがしやすいような機能がある言語＝関数型言語

主な関数型言語

静的型付け	ML(73), Clean(87), SML(90), Haskell(90), OCaml(96), Scala(03), F#(05), SML#(12)
動的型付け	Lisp(58), Scheme(75), Erlang(86), Clojure(07)

※カッコ内の数字は登場年

そういう言語があるのは知ってるけどさぁ、実際使われてるわけ？研究者とか趣味の人だけなんでしょ？

関数型の実例

Field Reports OCaml　検索

https://github.com/haxe-mirrors/haxe

https://github.com/xen-org/xen-api

https://github.com/facebook/pfff

https://github.com/facebook/lex-pass

Scala at foursquare　検索

Tumblr における Scala　検索

今日は主にOCamlを題材に静的型付き関数型言語の話をします。

※以下、「関数型」と言ったら静的型付き関数型言語のことを指します。

ブラウザで動く対話環境Try OCamlで
試すのも良いでしょう。

Try OCaml

検索

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

静的型付き関数型で開発しているときの気持ち

- JavaとRubyとOCamlで開発しているときの気持ちをタイムラインで比較してみたいと思います

※タイムラインって何？って方はこちら
➜ http://www.agileacademy.com.au/agile/sites/default/files/Timeline%20Retropectives%202011.pdf

Javaでの開発

(゜∀゜)

(´・ω・`)

Javaでの開発

（゜∀゜）

できたー

(￣・ω・￣)

Javaでの開発

(゜∀゜)

コンパイルエラー

(￣・ω・￣)

Javaでの開発

（゜∀゜）

うーん、何がダメなのかなー

(￣・ω・￣)

Javaでの開発

（゜∀゜）

これか！コンパイル通った！

(￣・ω・￣)

Javaでの開発

（゜∀゜）

動いた！

(￣・ω・￣)

Javaでの開発

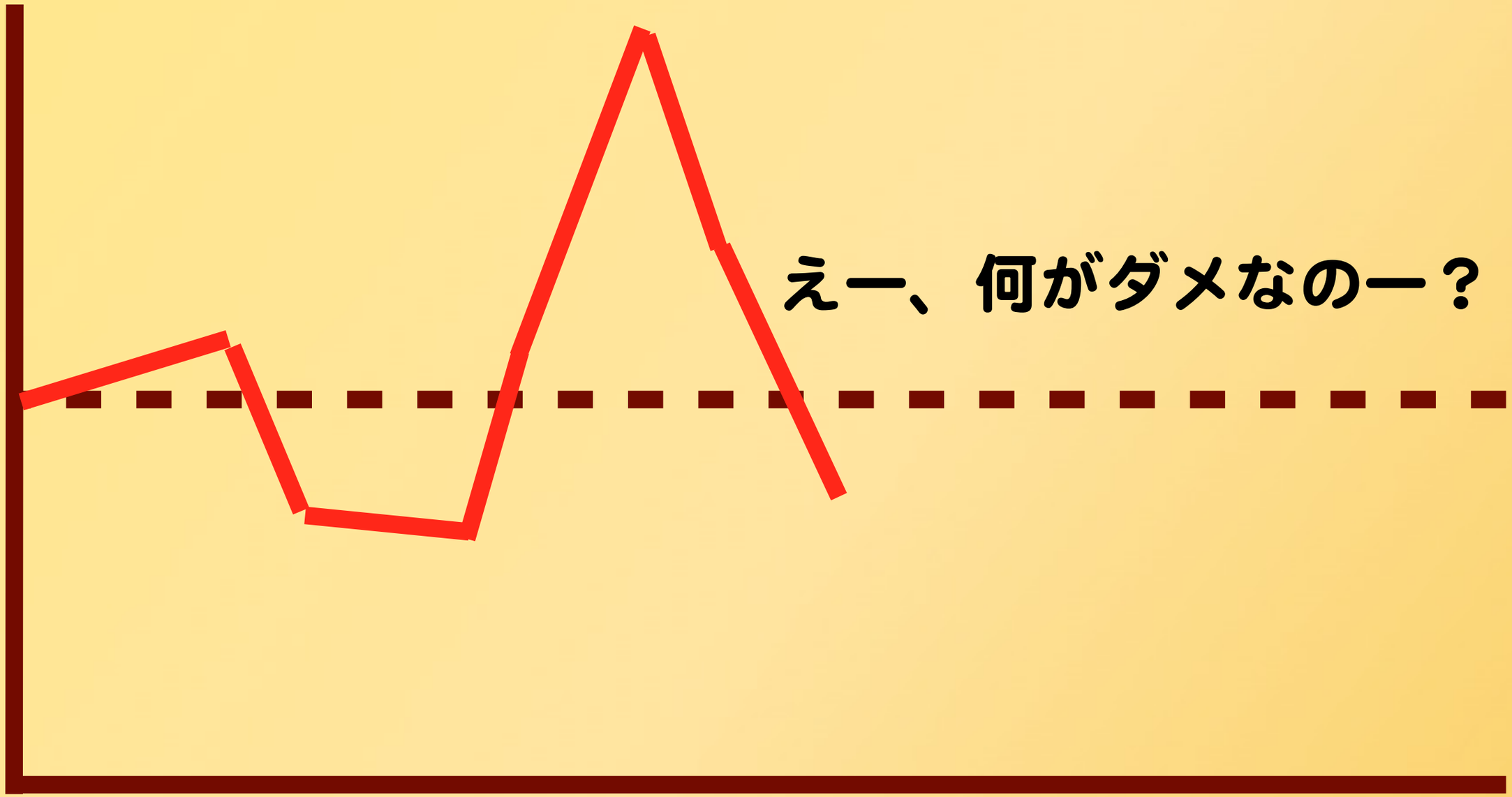
（゜∀゜）

ぬるぽ(｀・ω・´)

(´・ω・`)

Javaでの開発

(゜∀゜)

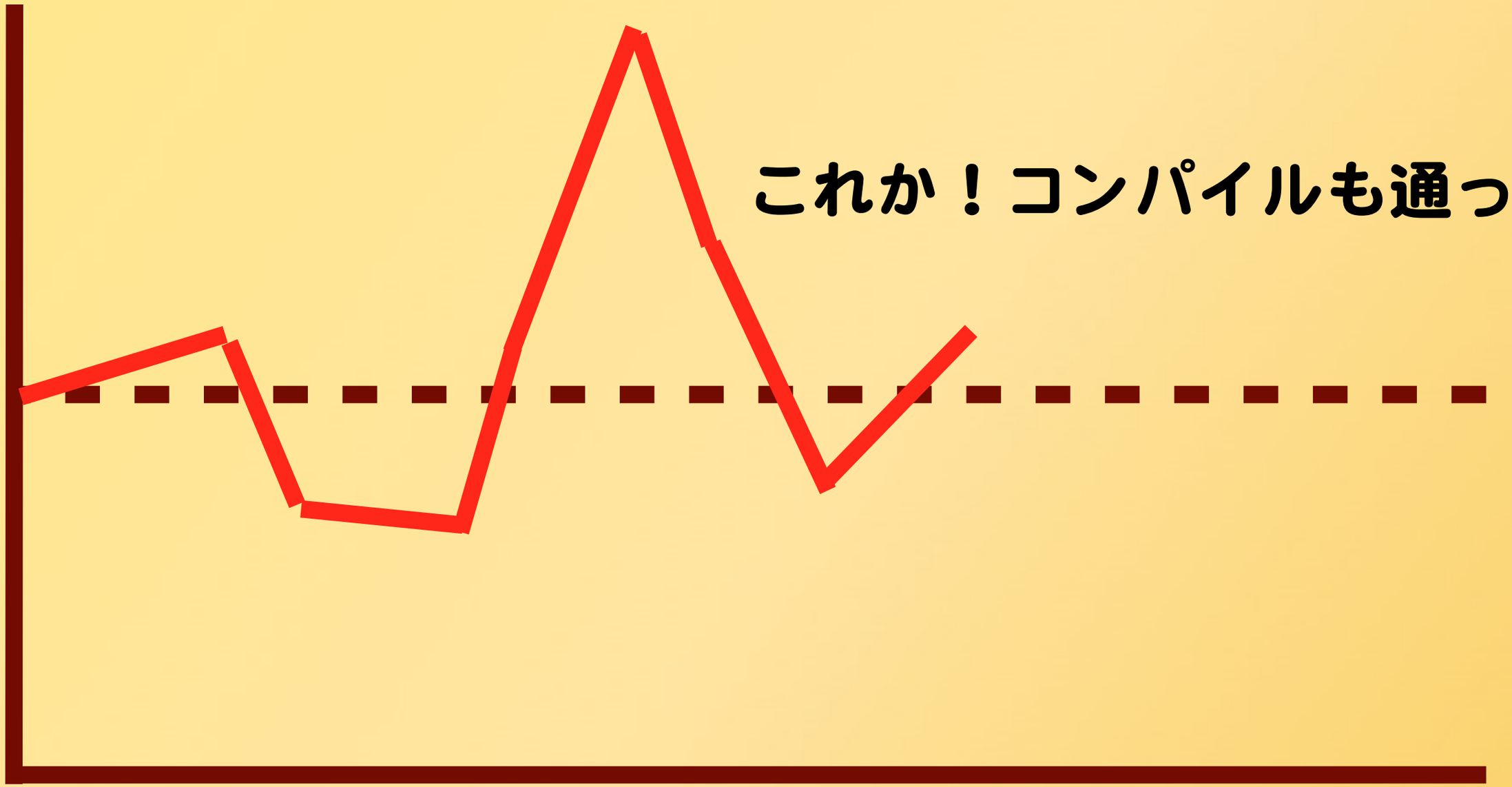


えー、何がダメなのー？

(´・ω・`)

Javaでの開発

（゜∀゜）



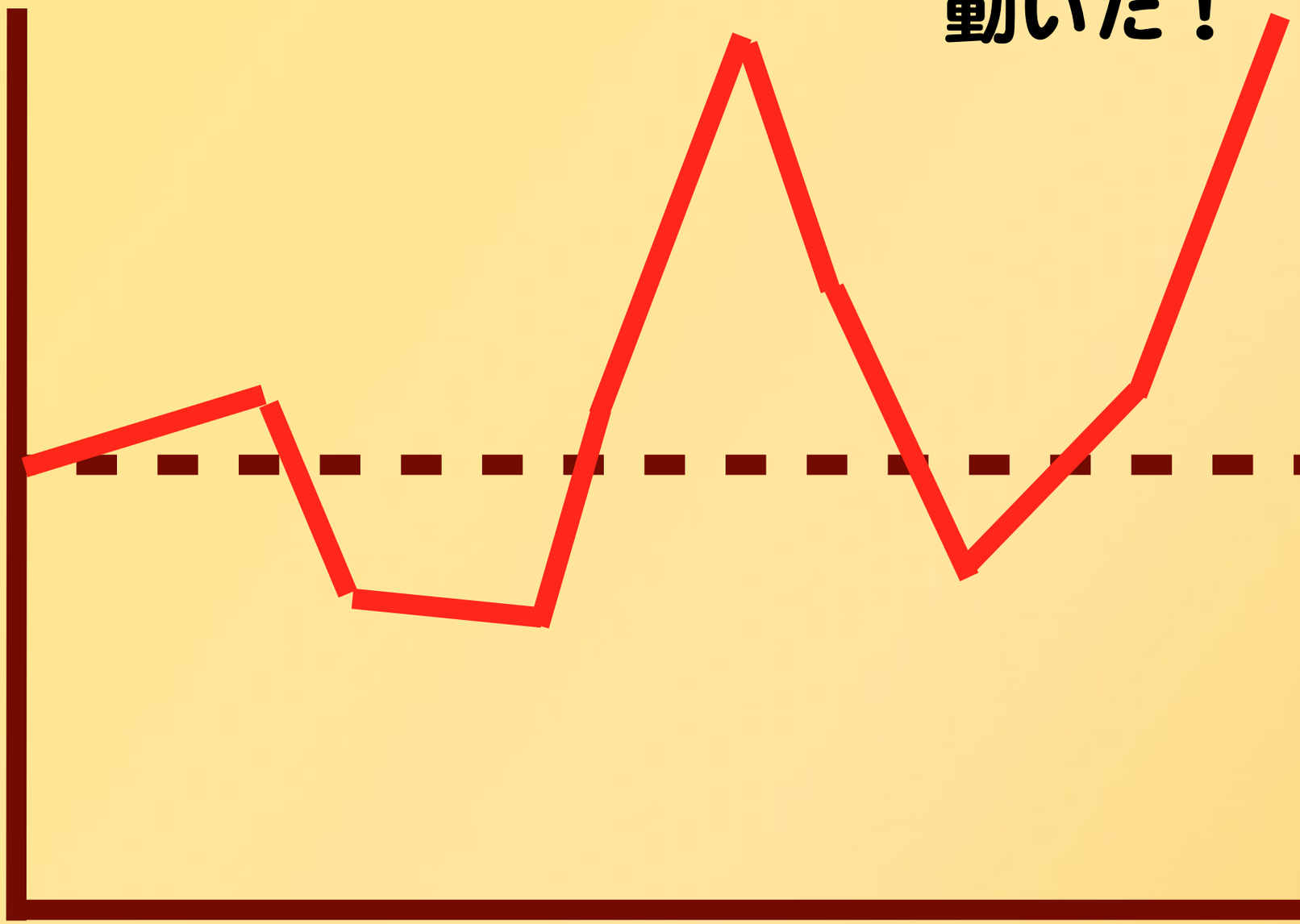
これか！コンパイルも通った！

(￣・ω・￣)

Javaでの開発

（゜∀゜）

動いた！

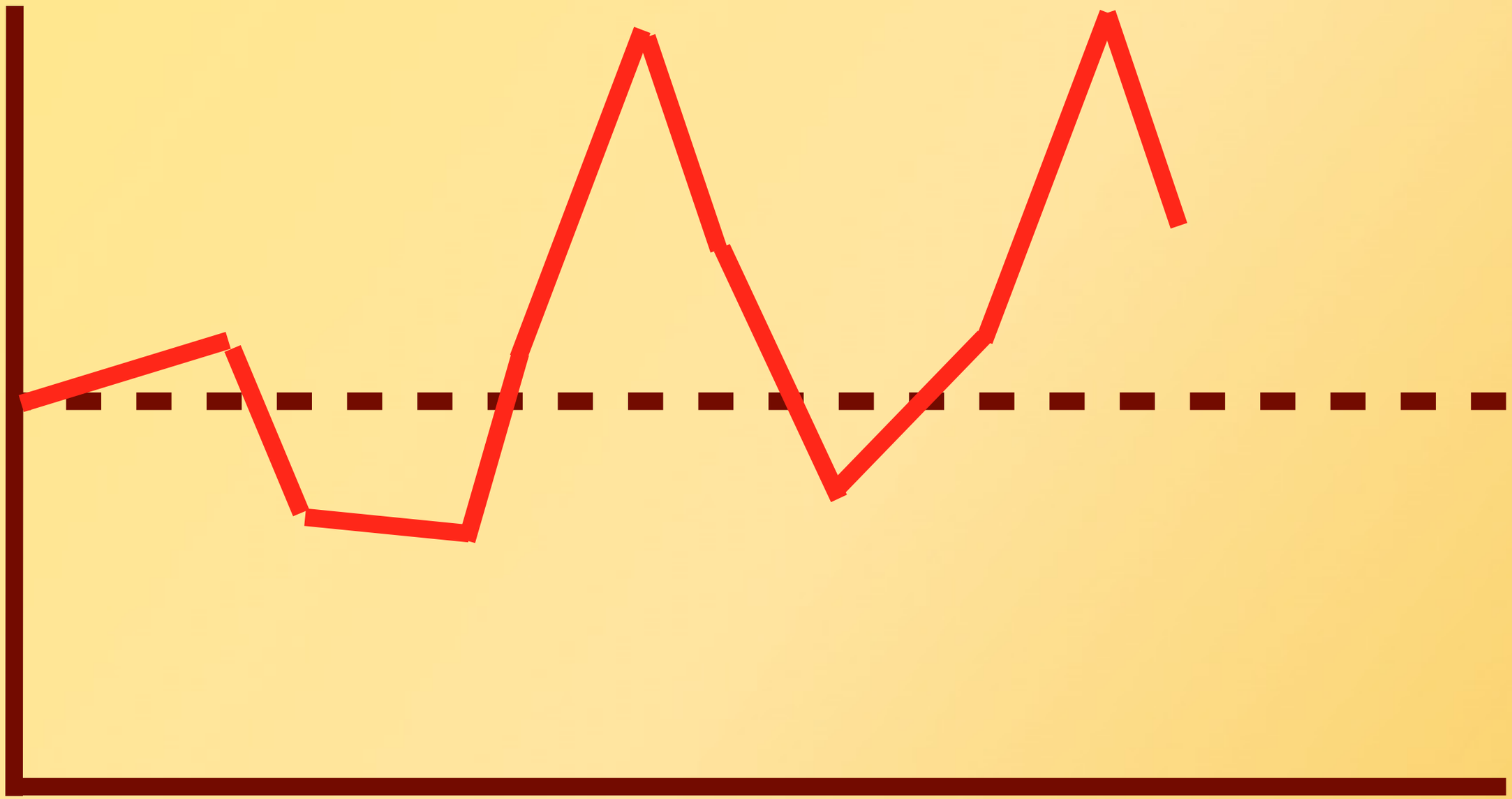


(￣・ω・￣)

Javaでの開発

（゜∀゜）

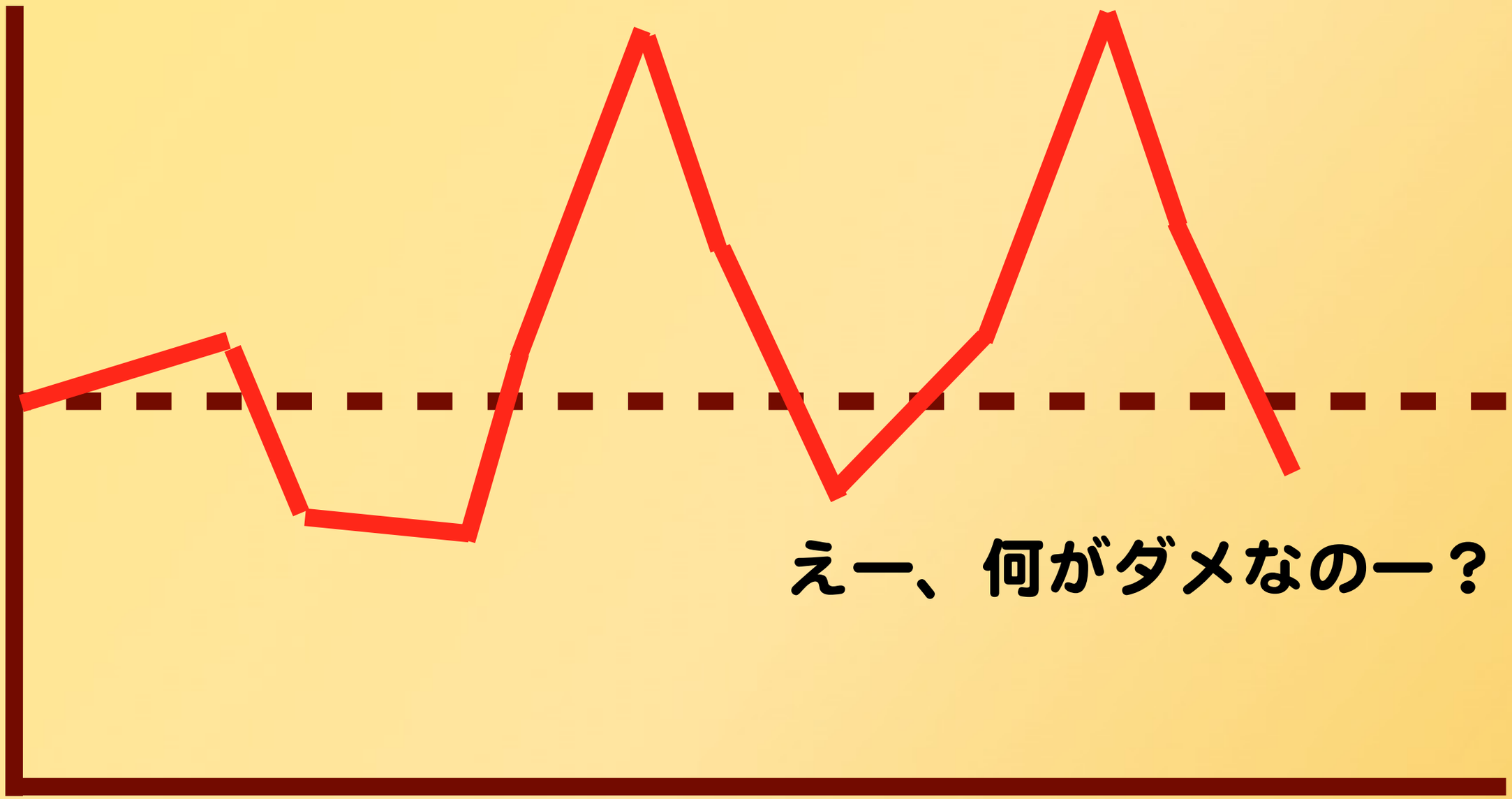
ぬるぽ(￣・ω・￣)



(￣・ω・￣)

Javaでの開発

(゜∀゜)

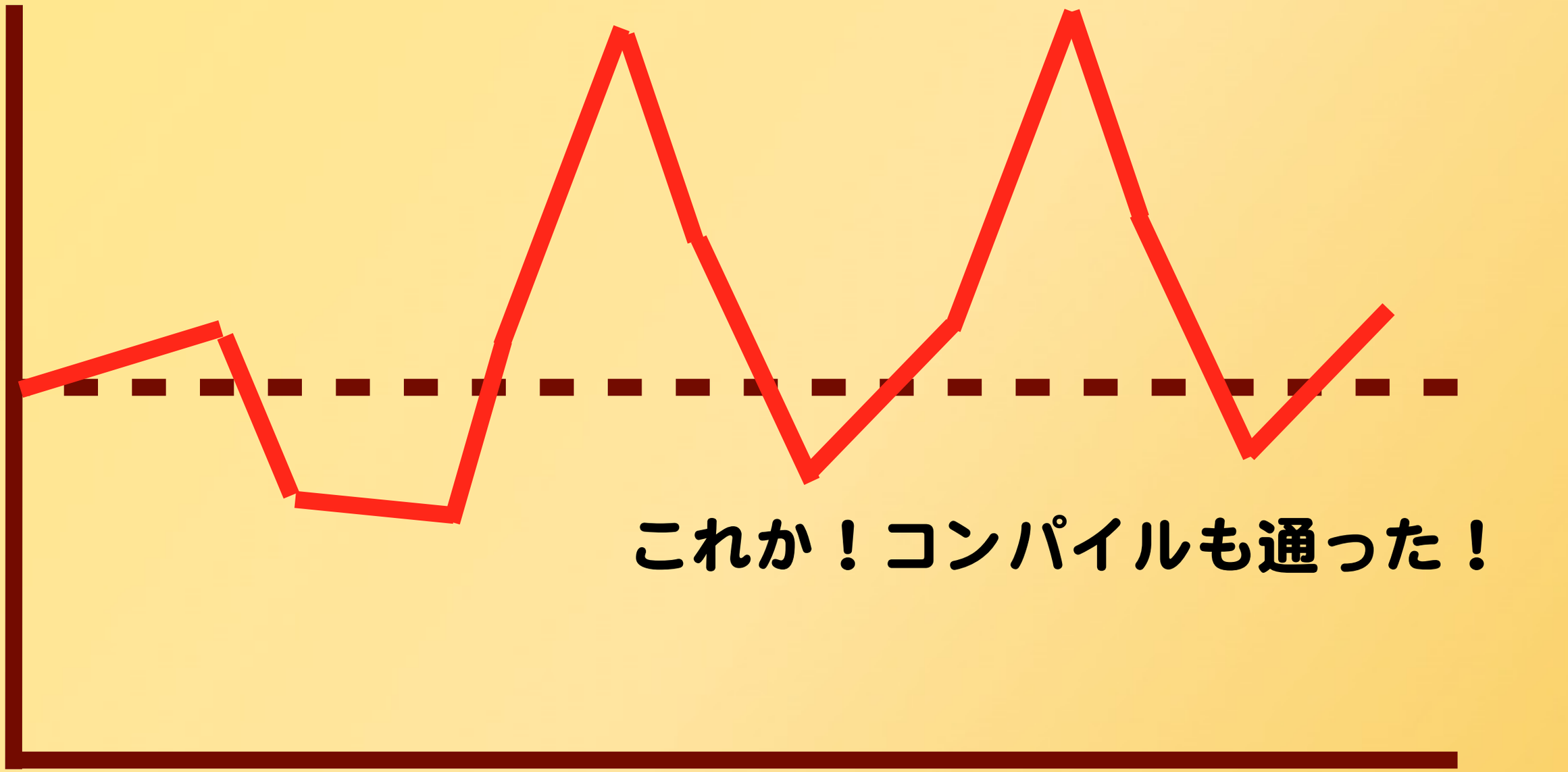


えー、何がダメなのー？

(￣・ω・￣)

Javaでの開発

（゜∀゜）



これか！コンパイルも通った！

(￣・ω・￣)

Javaでの開発

（゜∀゜）

コンパイル通っても実行時に落ちることが多いので、
コンパイルによる型チェックが有り難い物だと言う
感覚をあまり持てない。

(￣・ω・￣)

Rubyでの開発

Rubyでの開発

（゜∀゜）

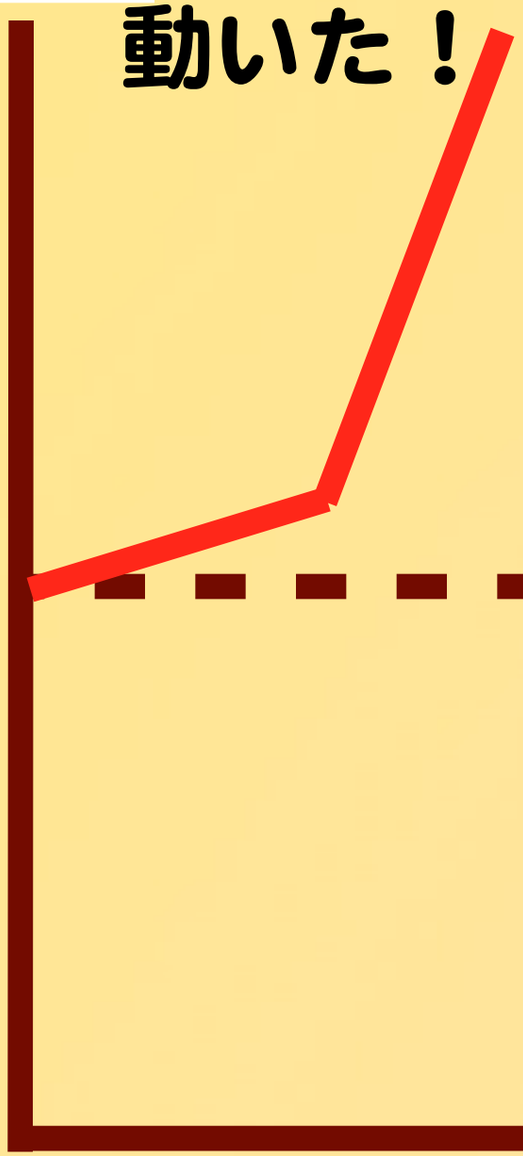
できたー

(ˉ・ω・ˉ)

Rubyでの開発

（゜∀゜）

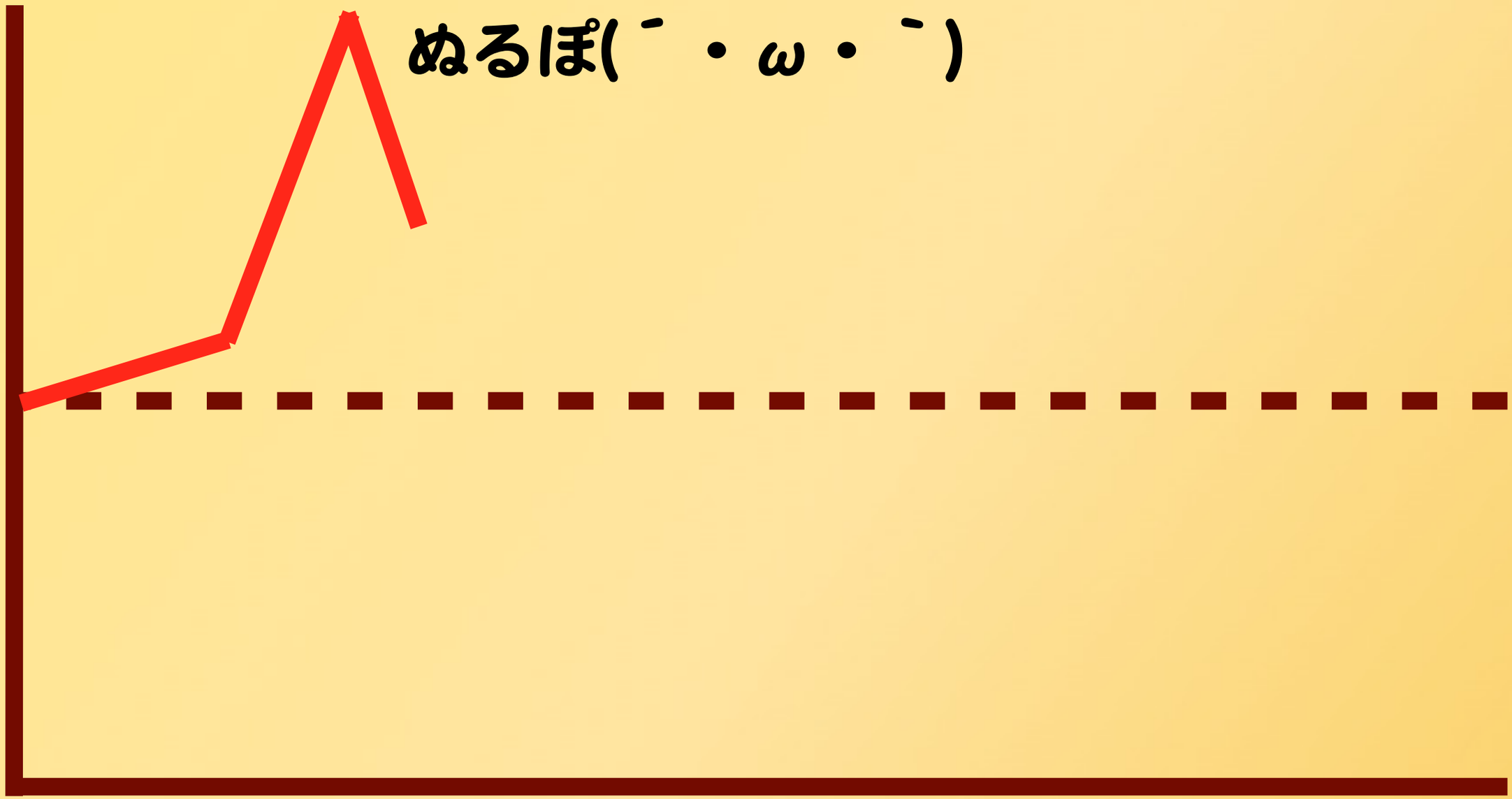
動いた！



(￣・ω・￣)

Rubyでの開発

(゜∀゜)

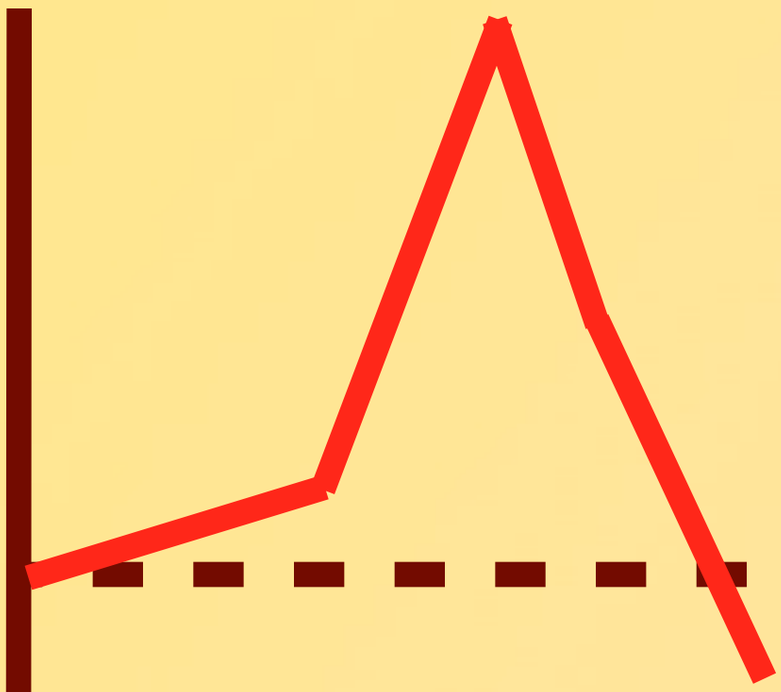


ぬるぽ(ﾟ・ω・ﾟ)

(ﾟ・ω・ﾟ)

Rubyでの開発

（゜∀゜）

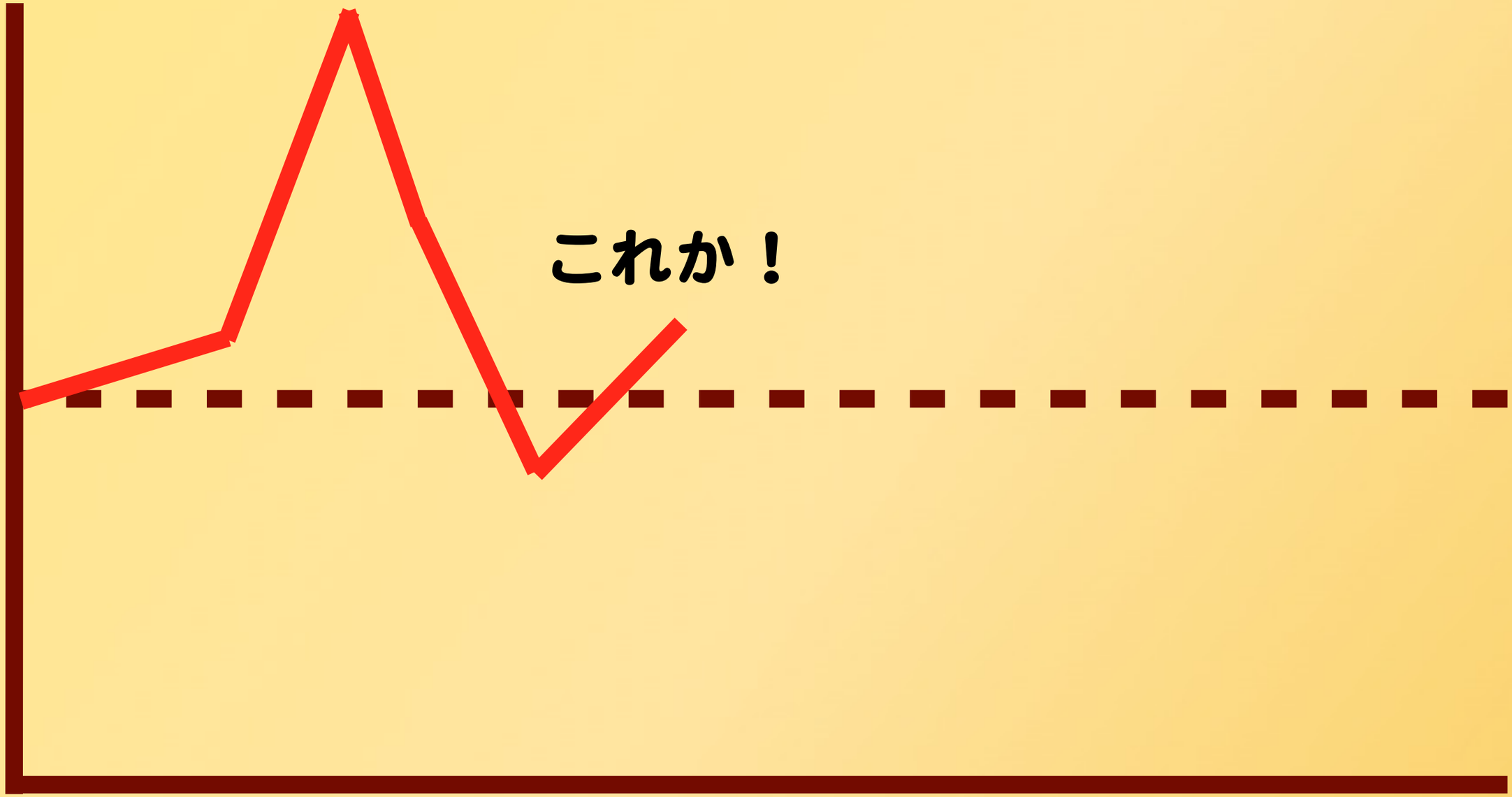


うーん、何がダメなのかなー

(￣・ω・￣)

Rubyでの開発

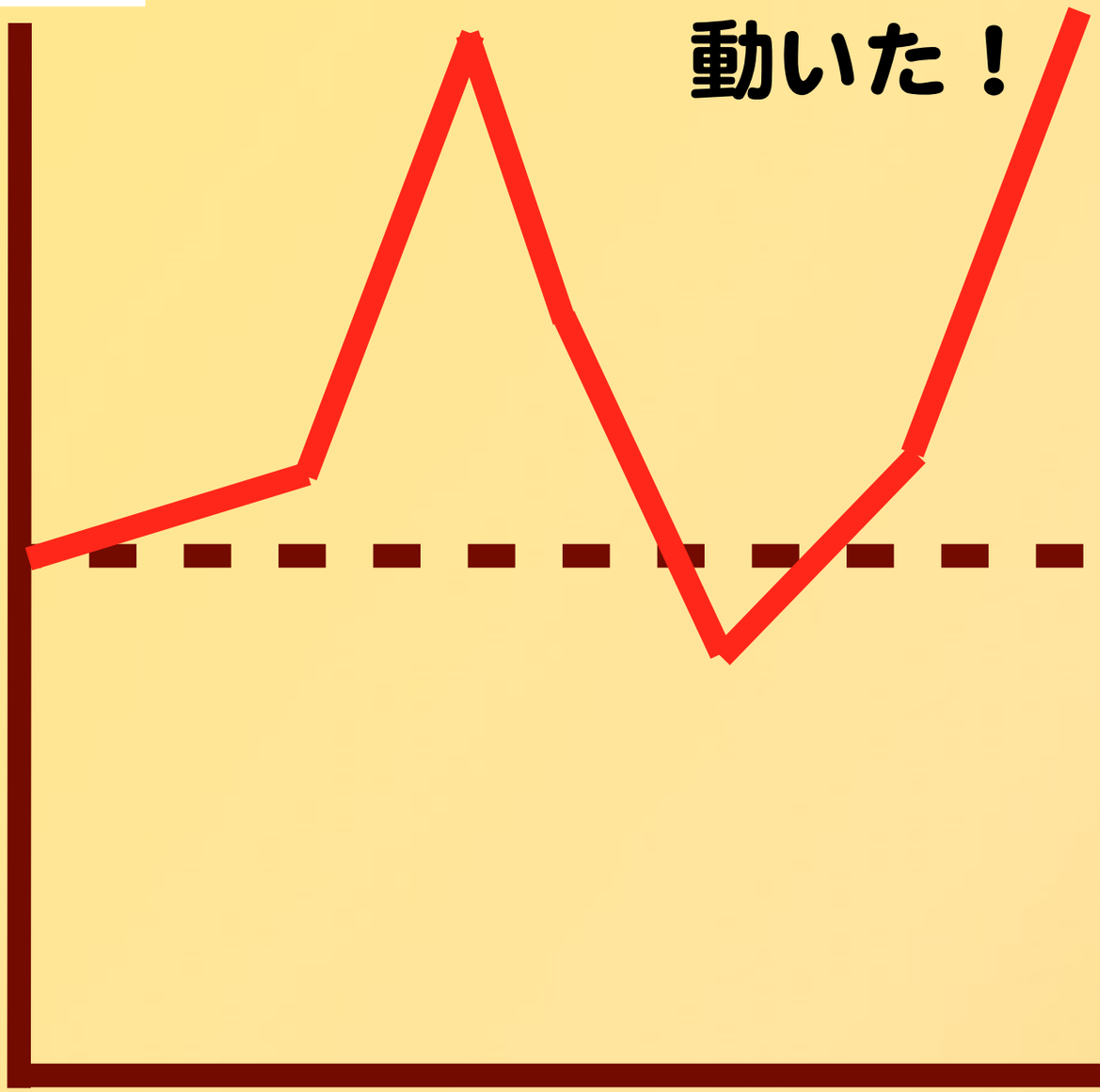
（゜∀゜）



(￣・ω・￣)

Rubyでの開発

（゜∀゜）

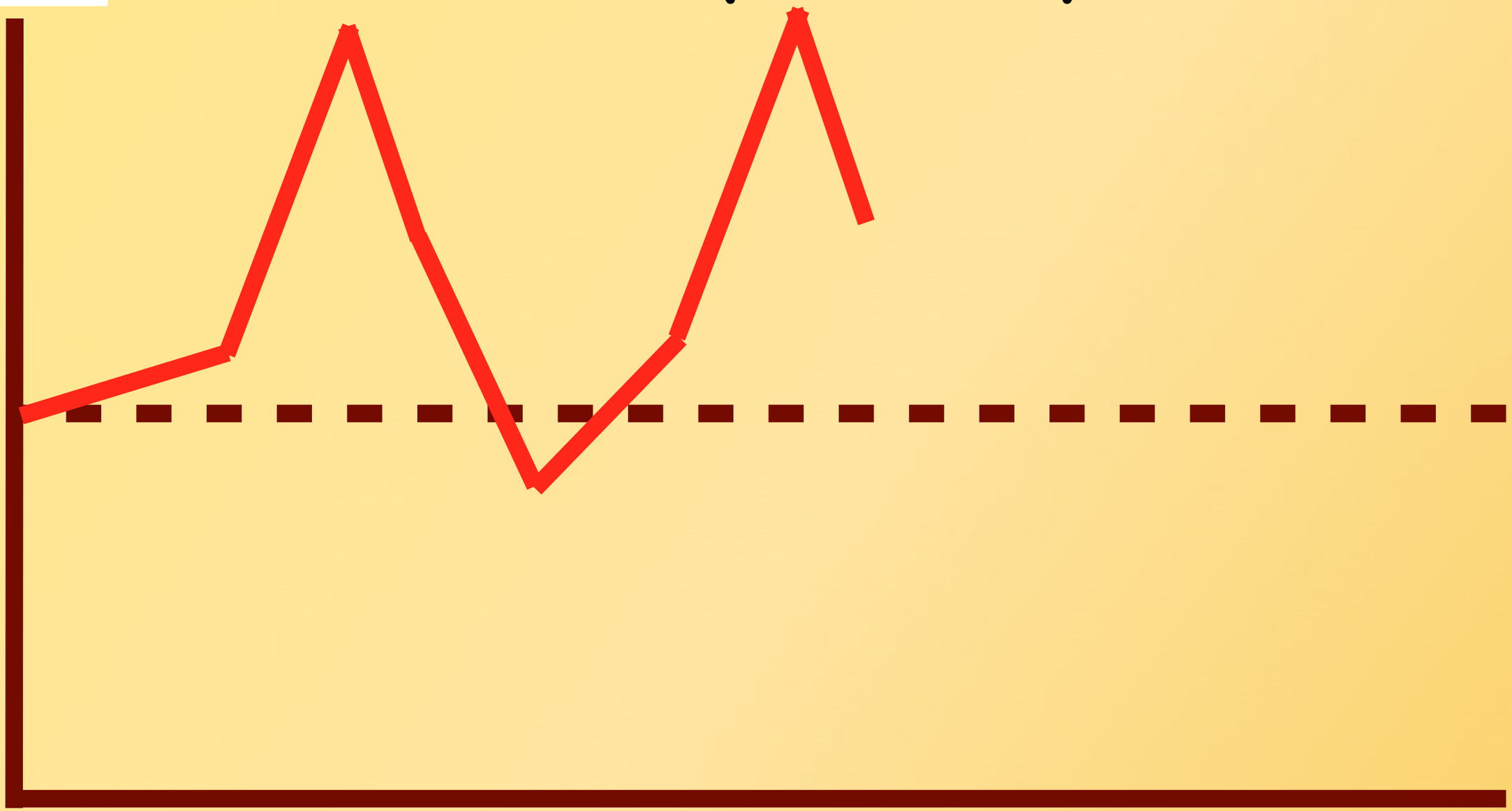


(ˉ・ω・ˉ)

Rubyでの開発

（゜∀゜）

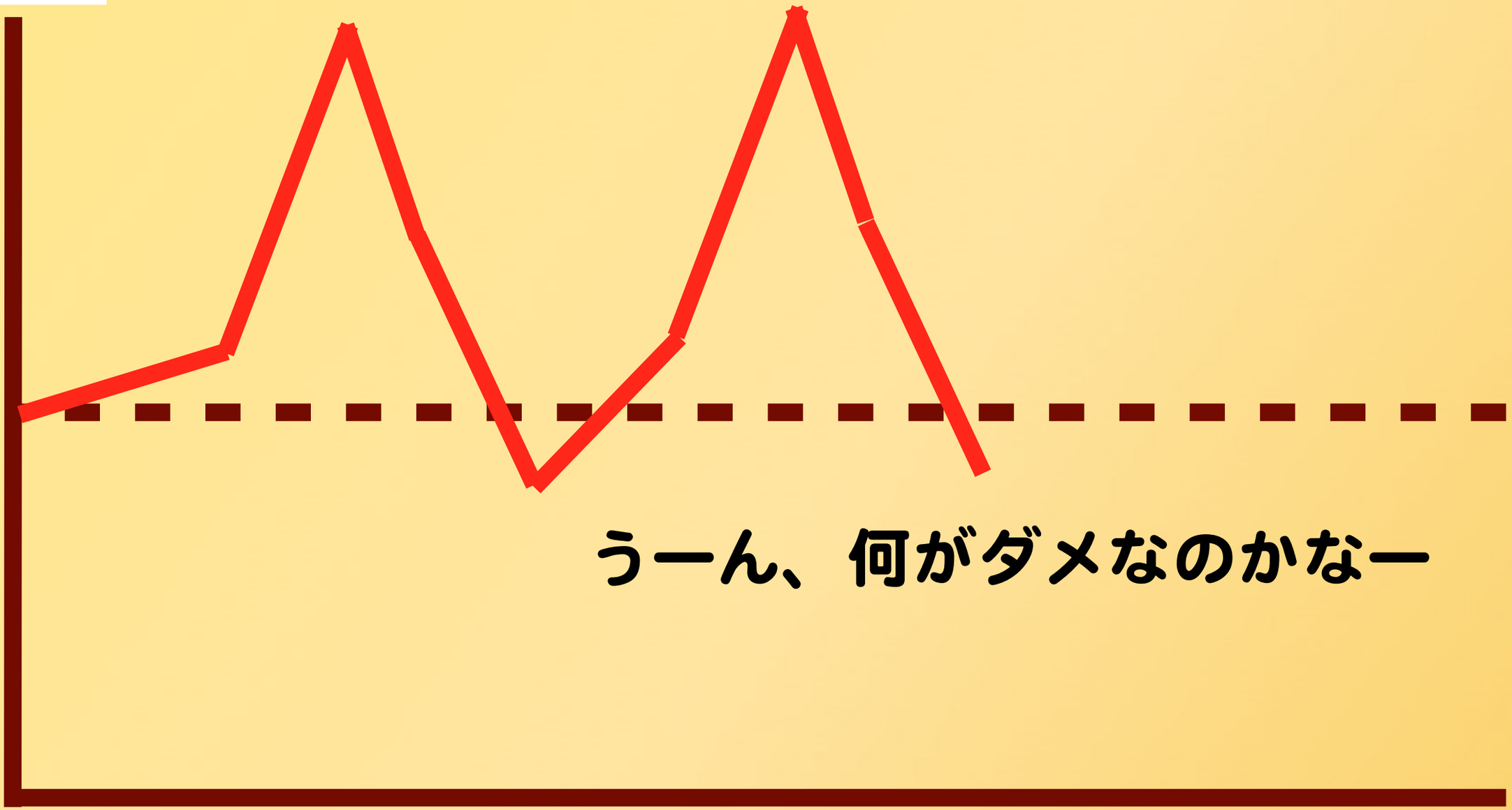
ぬるぽ(｀・ω・´)



(´・ω・`)

Rubyでの開発

（゜∀゜）

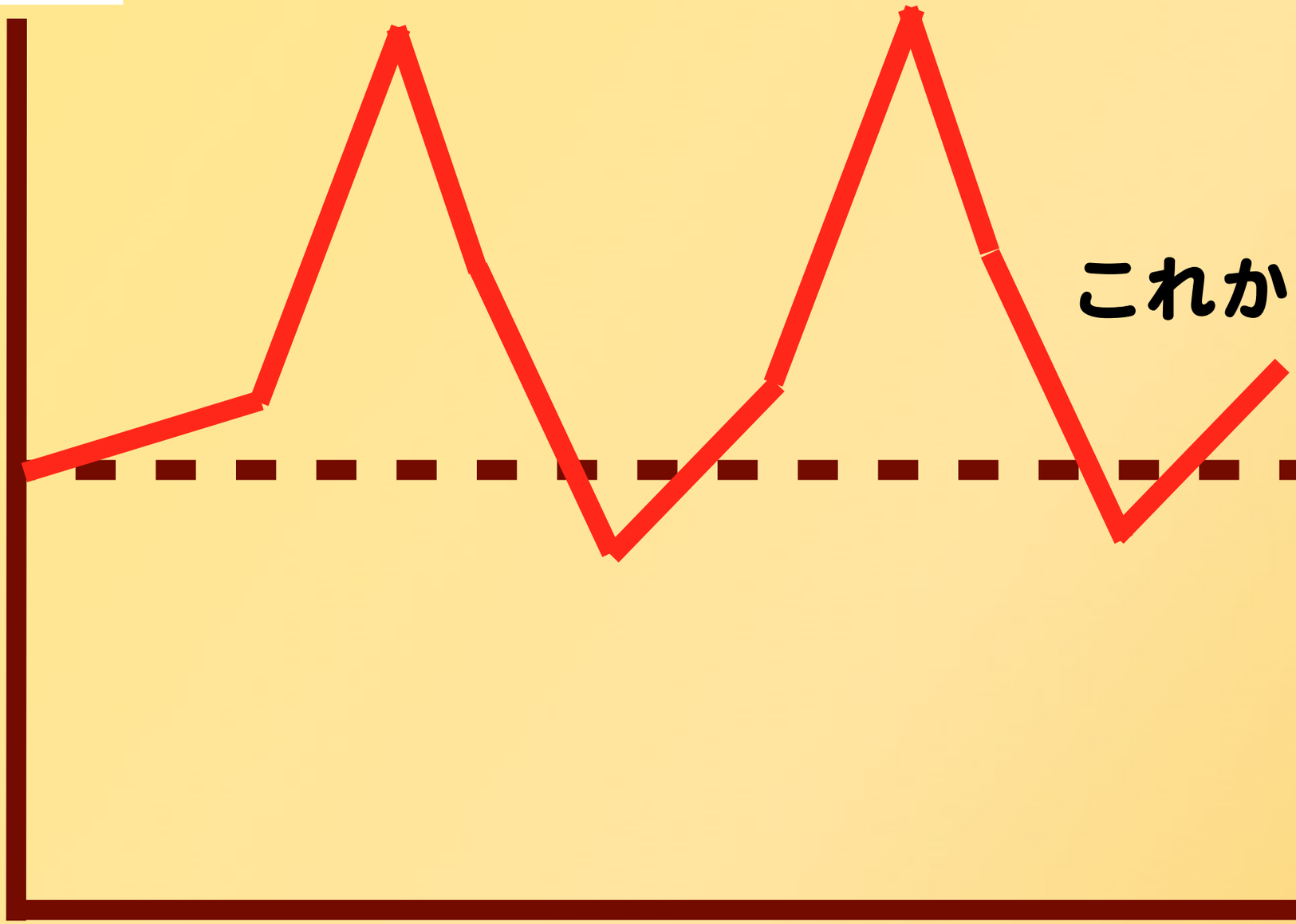


うーん、何がダメなのかなー

(￣・ω・￣)

Rubyでの開発

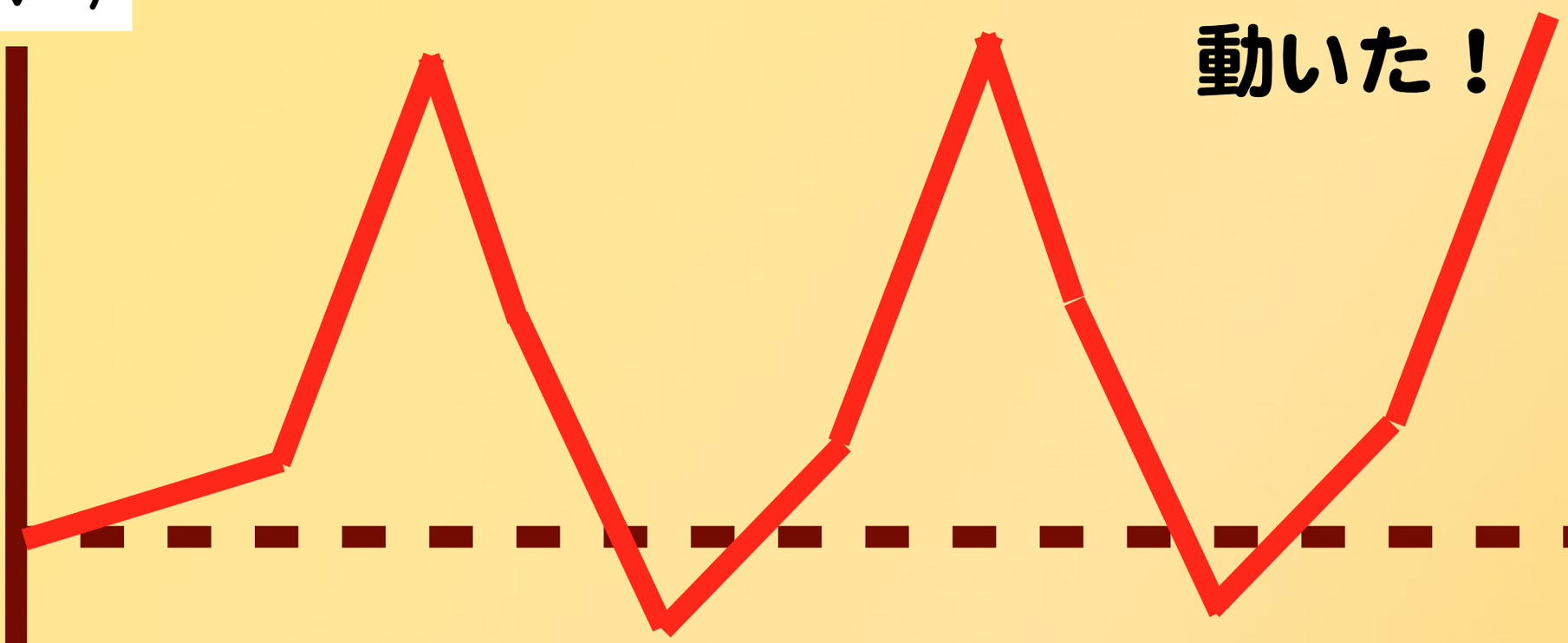
（゜∀゜）



(￣・ω・￣)

Rubyでの開発

（゜∀゜）



早い段階で「動いたー」という喜びを得られるので、
その喜びを糧にその後の実行時エラーの対処もがんばれる。
但し、いちいち動かさないとエラーがわからないので
エラーが全てつぶれるまでには時間がかかる。

(￣・ω・￣)

関数型での開発(初心者)

(゜∀゜)

(´・ω・`)

関数型での開発(初心者)

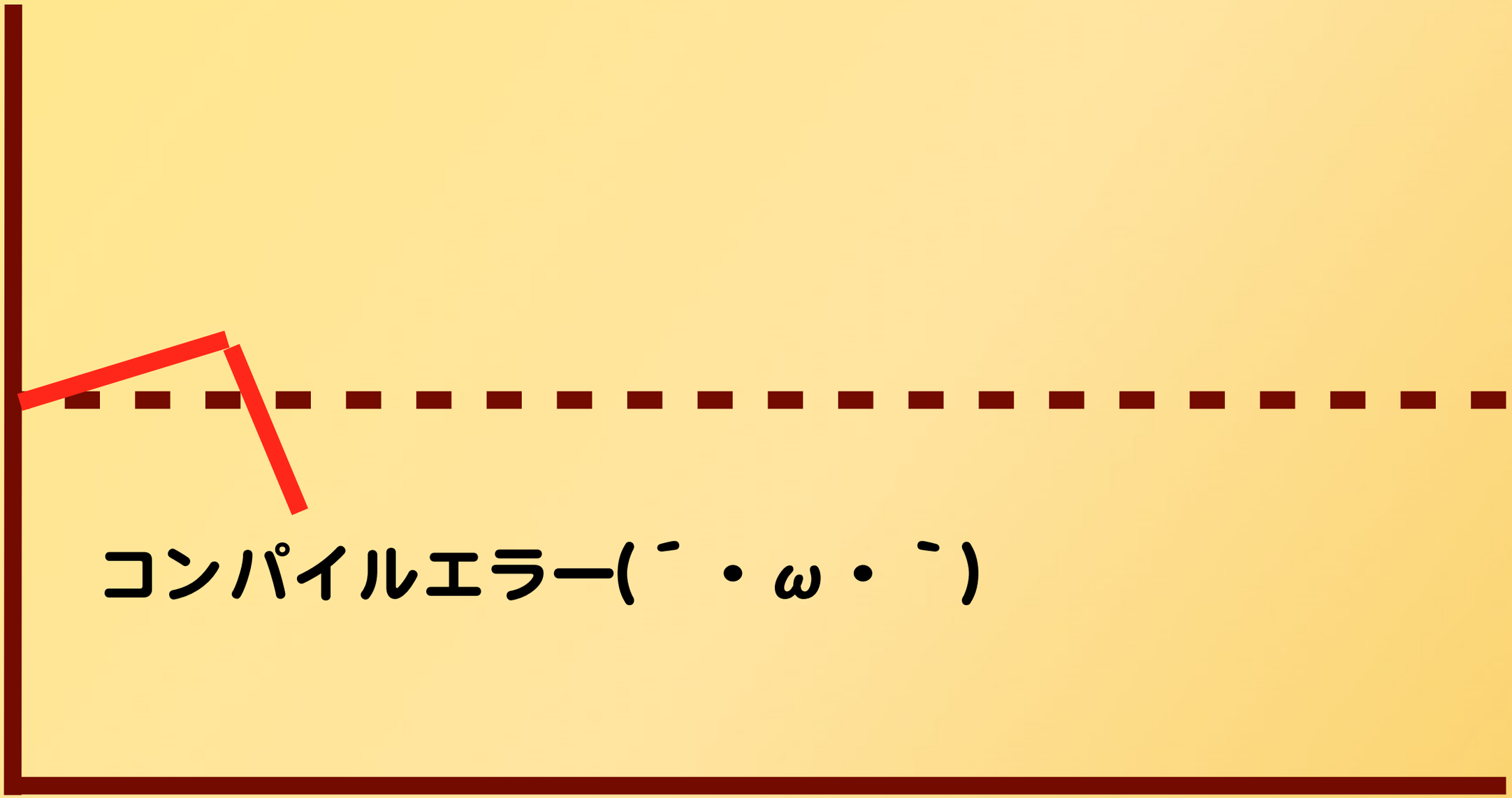
(゜∀゜)

できたー

(¯・ω・¯)

関数型での開発(初心者)

(゜∀゜)

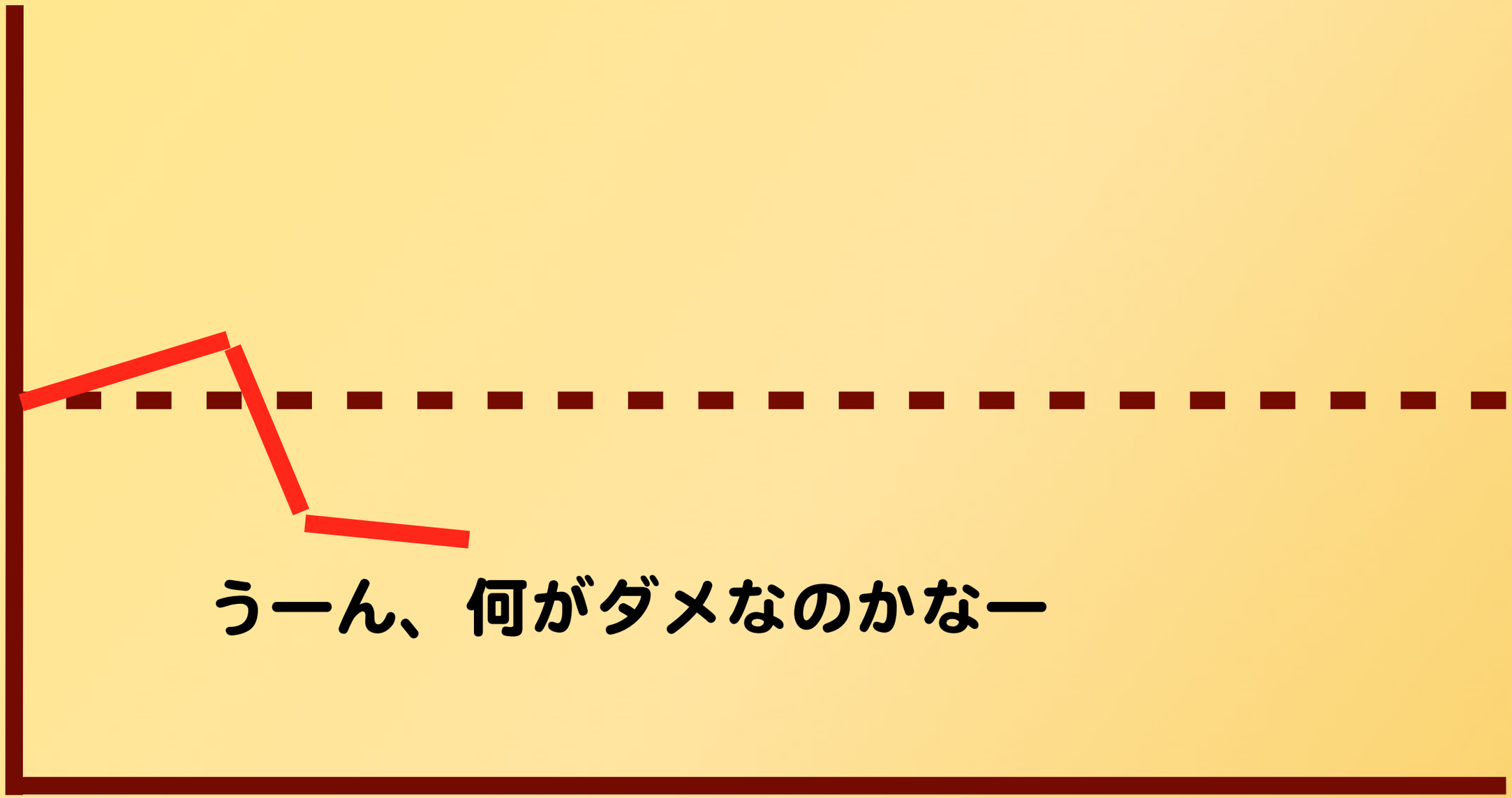


コンパイルエラー(´・ω・`)

(´・ω・`)

関数型での開発(初心者)

(゜∀゜)



うーん、何がダメなのかなー

(￣・ω・￣)

関数型での開発(初心者)

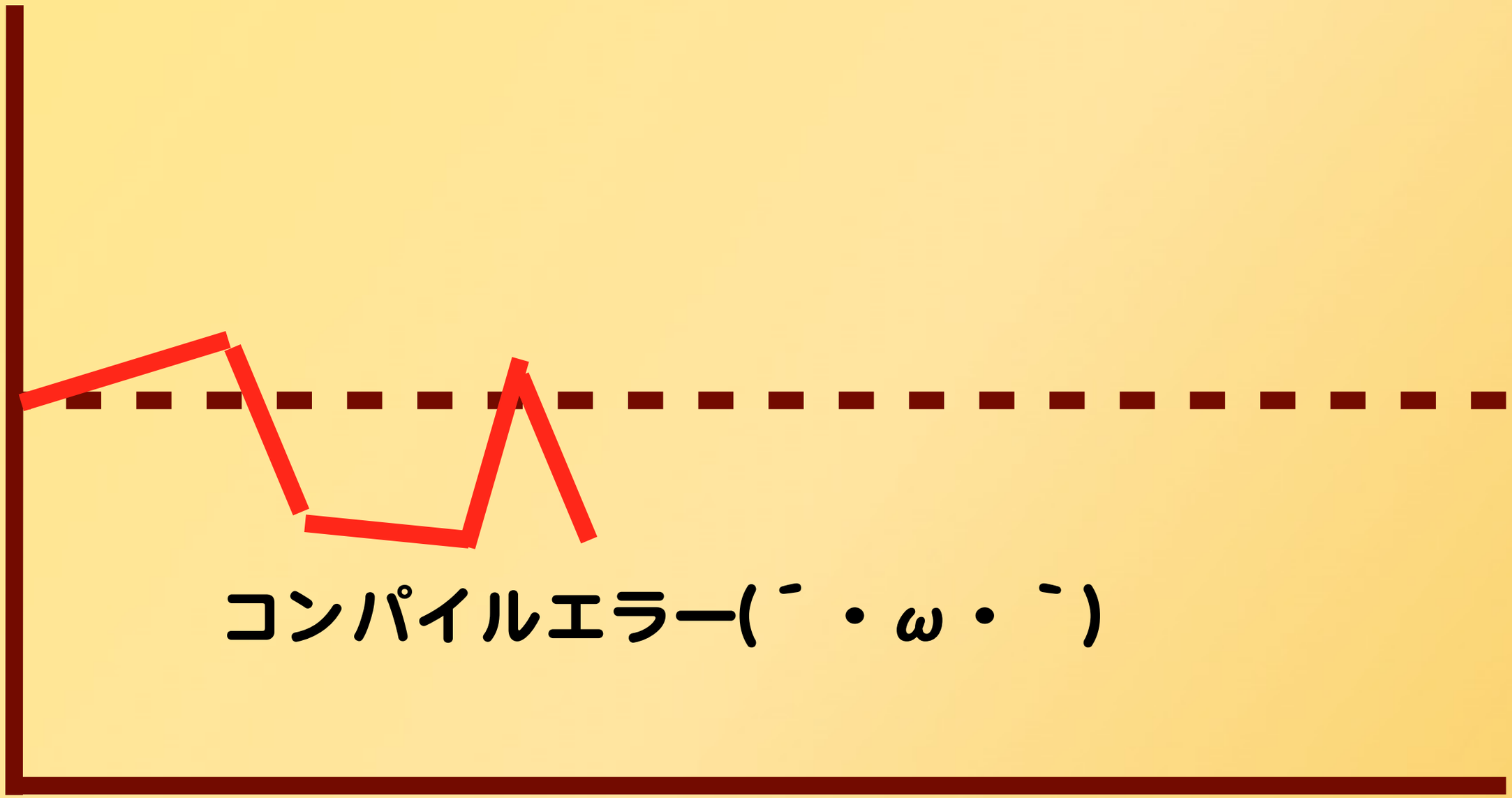
(゜∀゜)

これか！

(ˉ・ω・ˉ)

関数型での開発(初心者)

(゜∀゜)

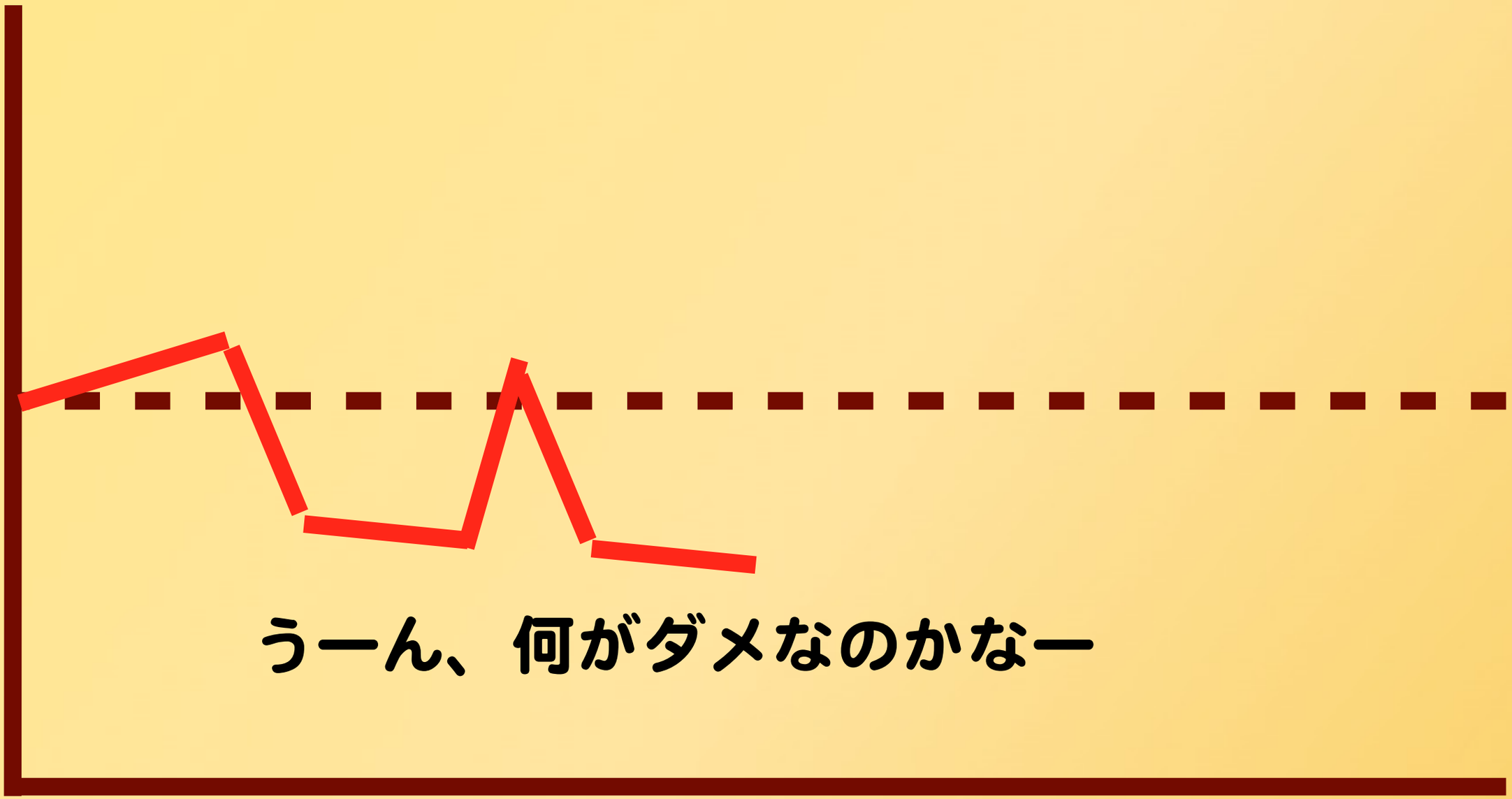


コンパイルエラー(´・ω・`)

(´・ω・`)

関数型での開発(初心者)

(゜∀゜)



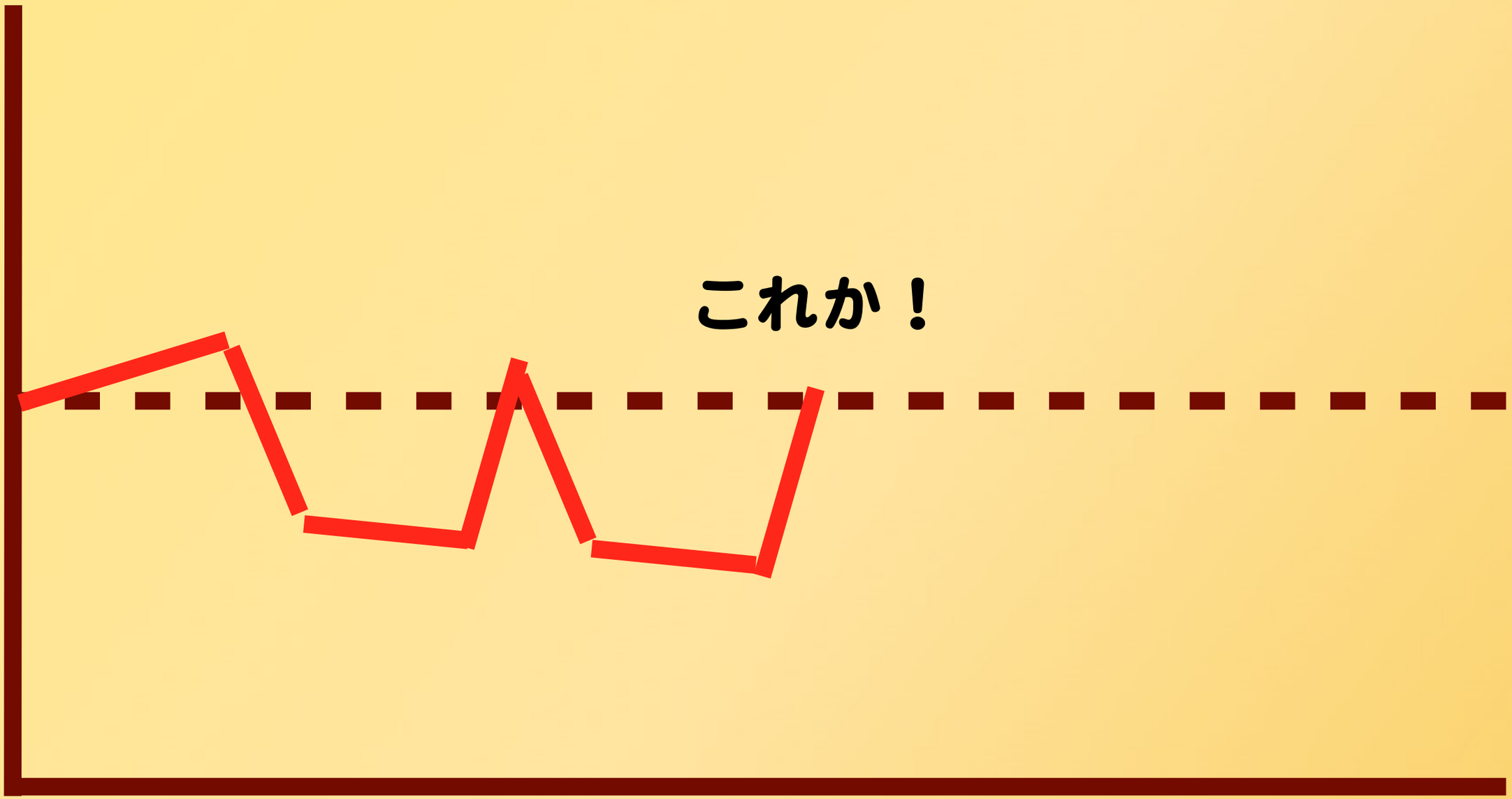
うーん、何がダメなのかなー

(￣・ω・￣)

関数型での開発(初心者)

（゜∀゜）

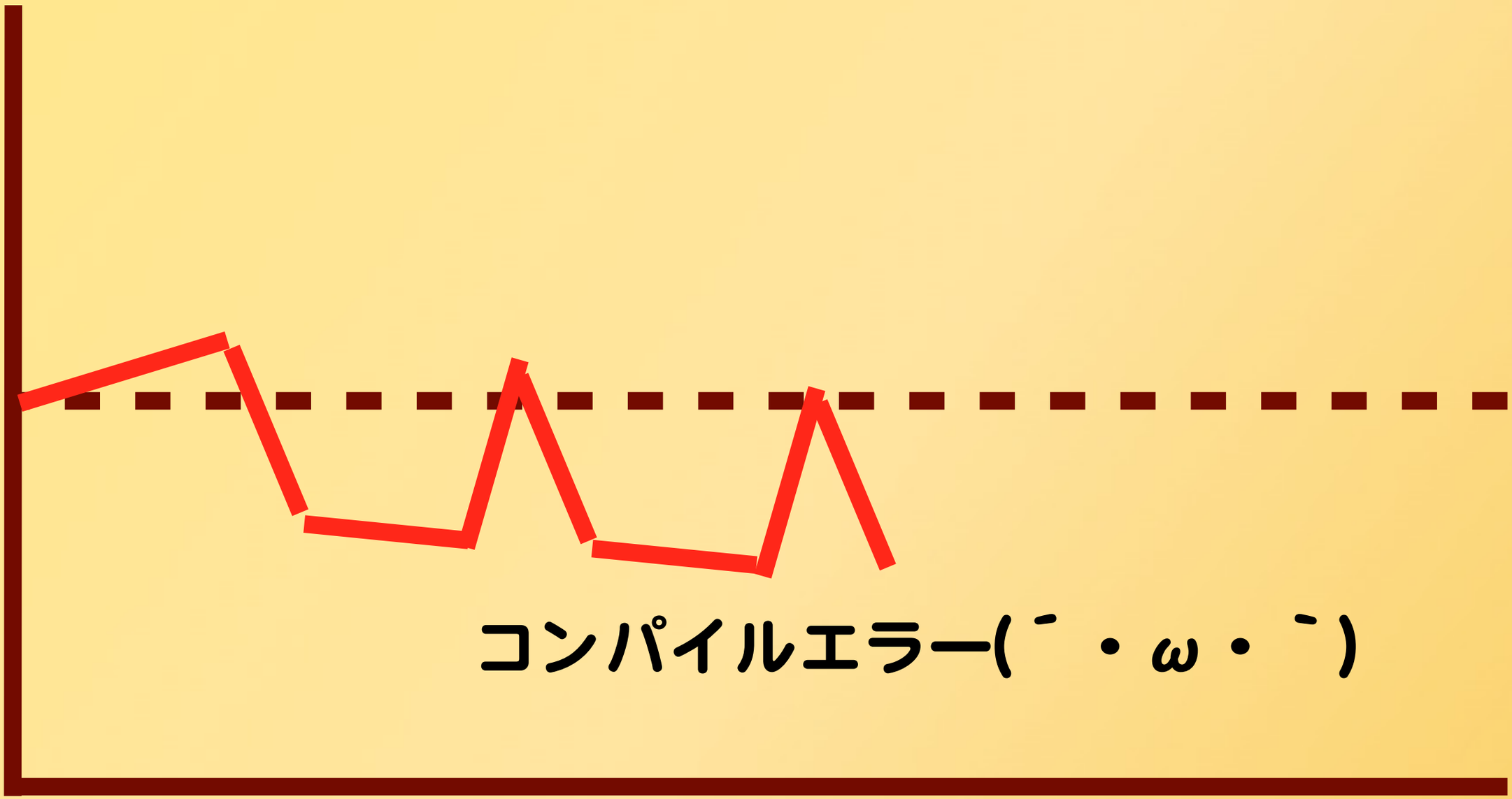
これか！



(￣・ω・￣)

関数型での開発(初心者)

(゜∀゜)

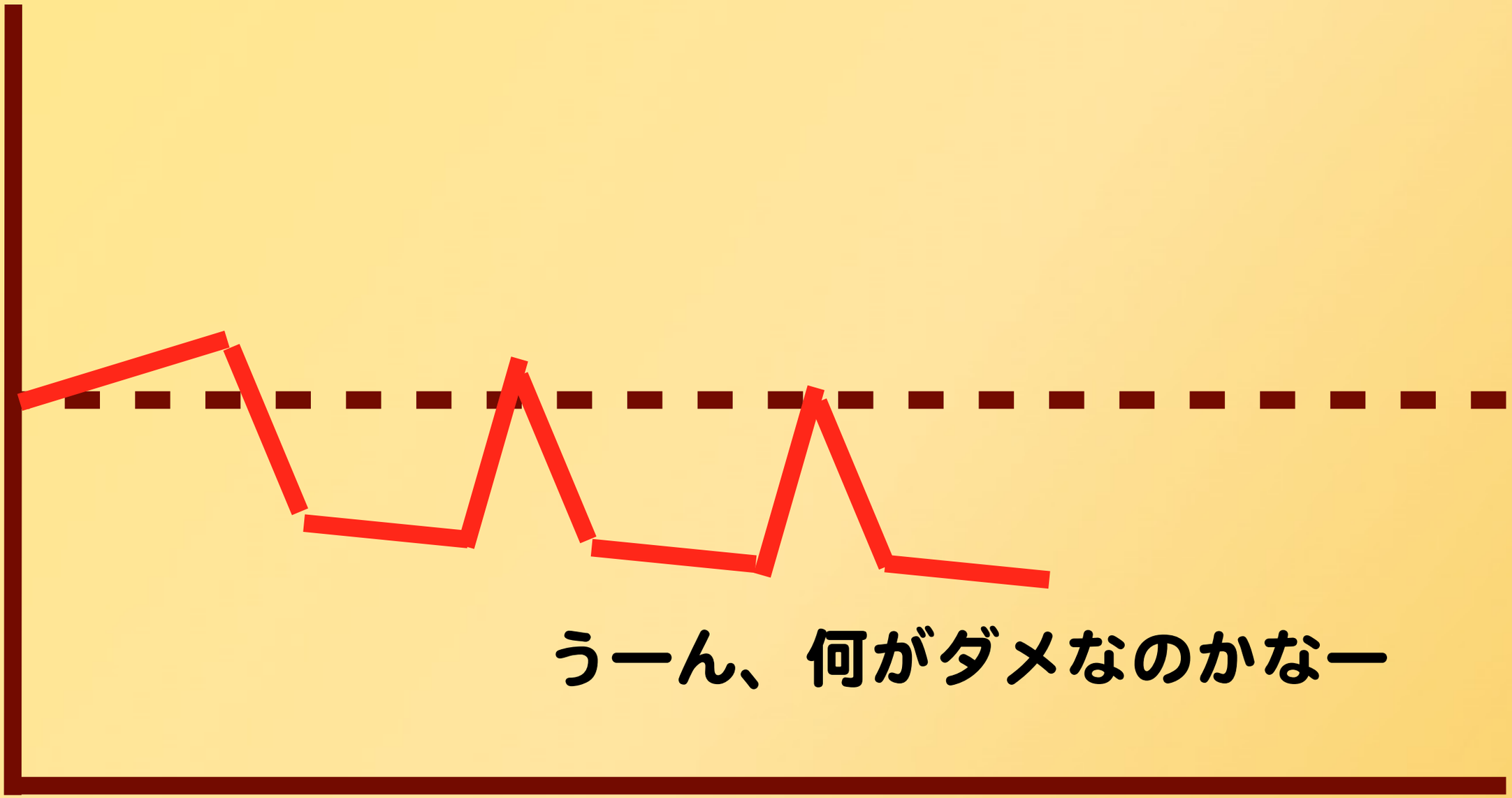


コンパイルエラー(ˊ・ω・ˋ)

(ˊ・ω・ˋ)

関数型での開発(初心者)

(゜∀゜)

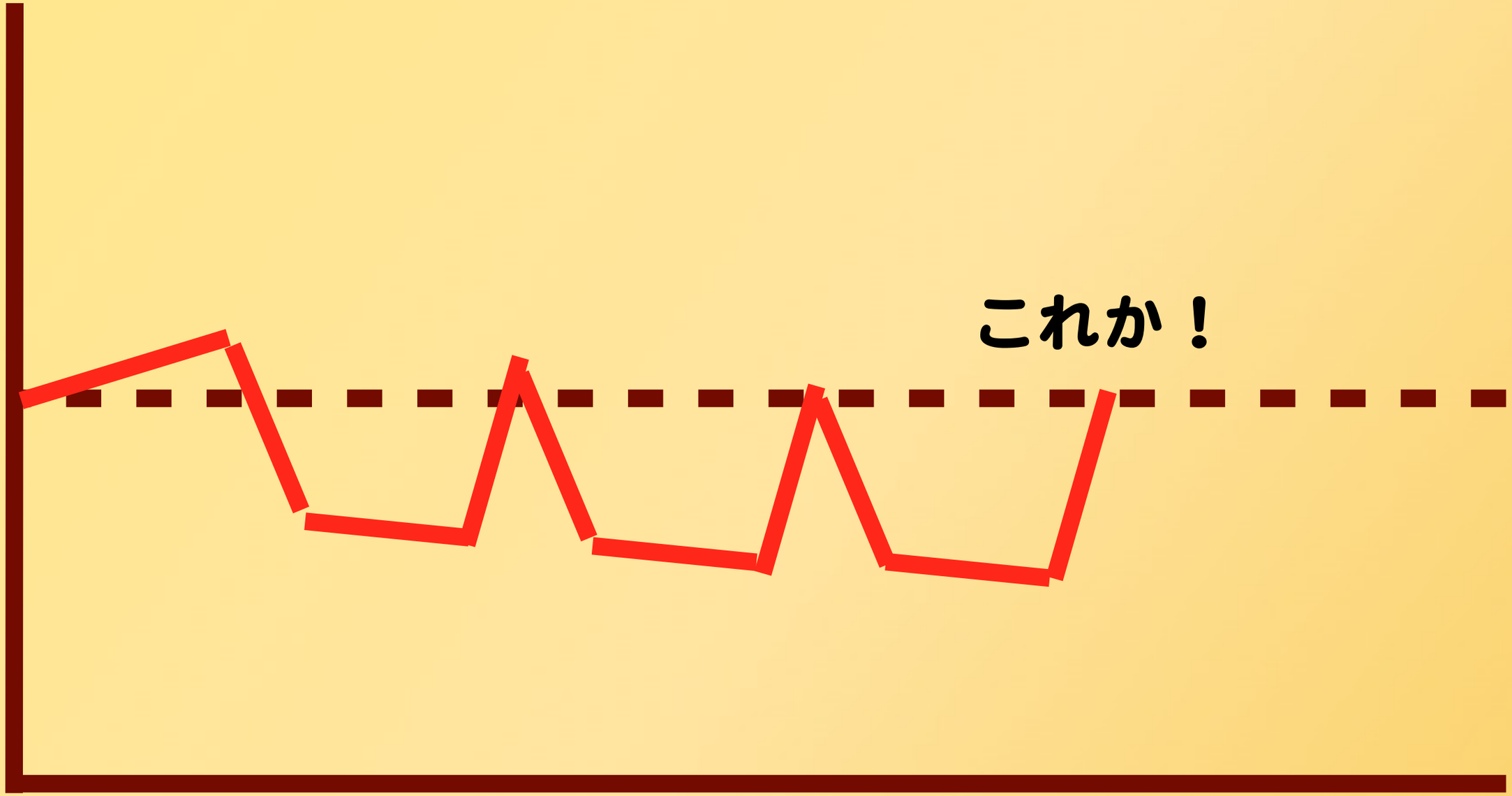


うーん、何がダメなのかなー

(￣・ω・￣) 動かせずにいっぱいコンパイラに文句言われるのでどんどん気分が落ちていく

関数型での開発(初心者)

(゜∀゜)



(￣・ω・￣)

関数型での開発(初心者)

(゜∀゜)

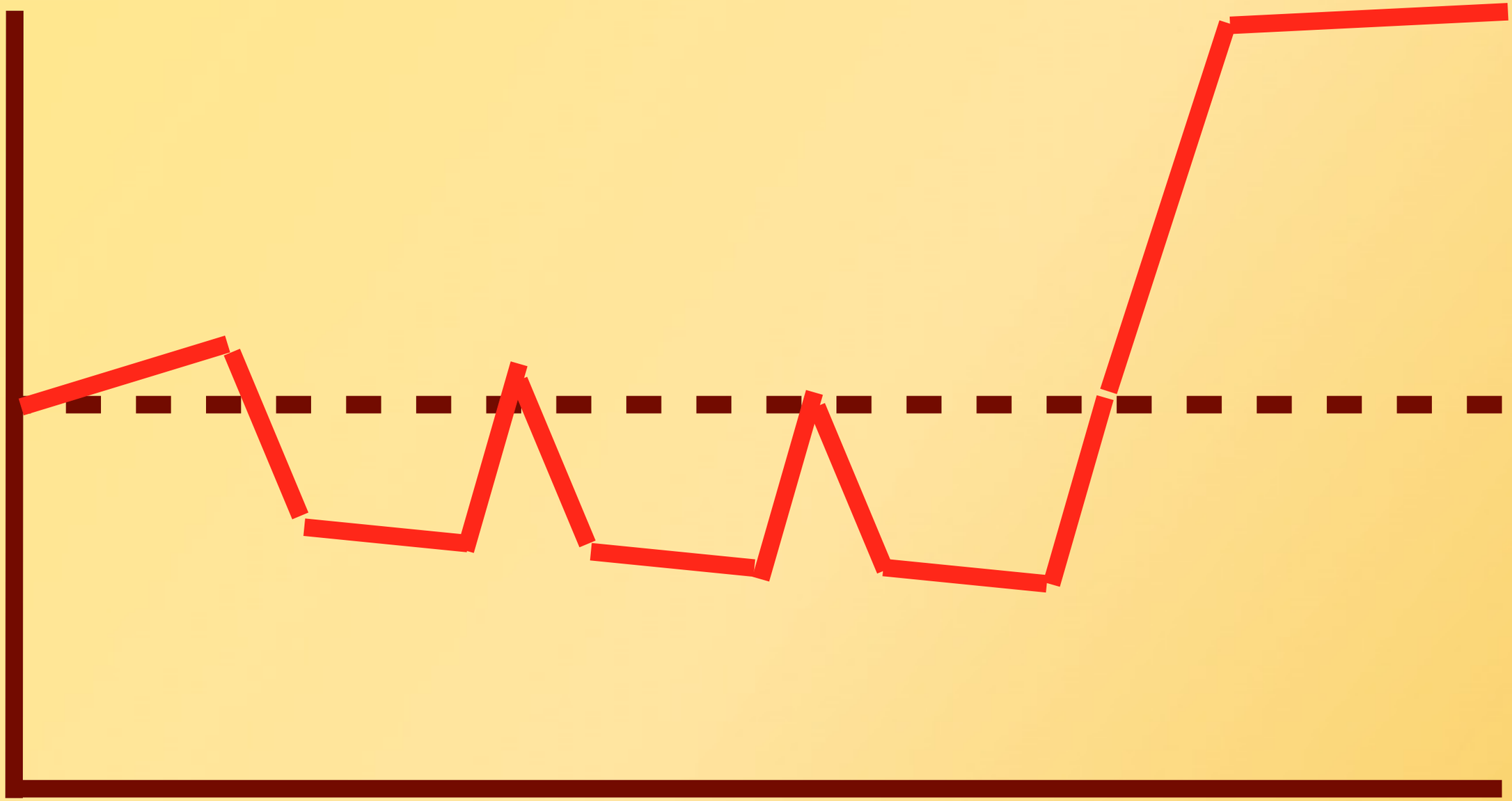
コンパイル通ったーーーーーー！

(ˉ・ω・ˉ)

関数型での開発(初心者)

実行時エラー起きねーーー！

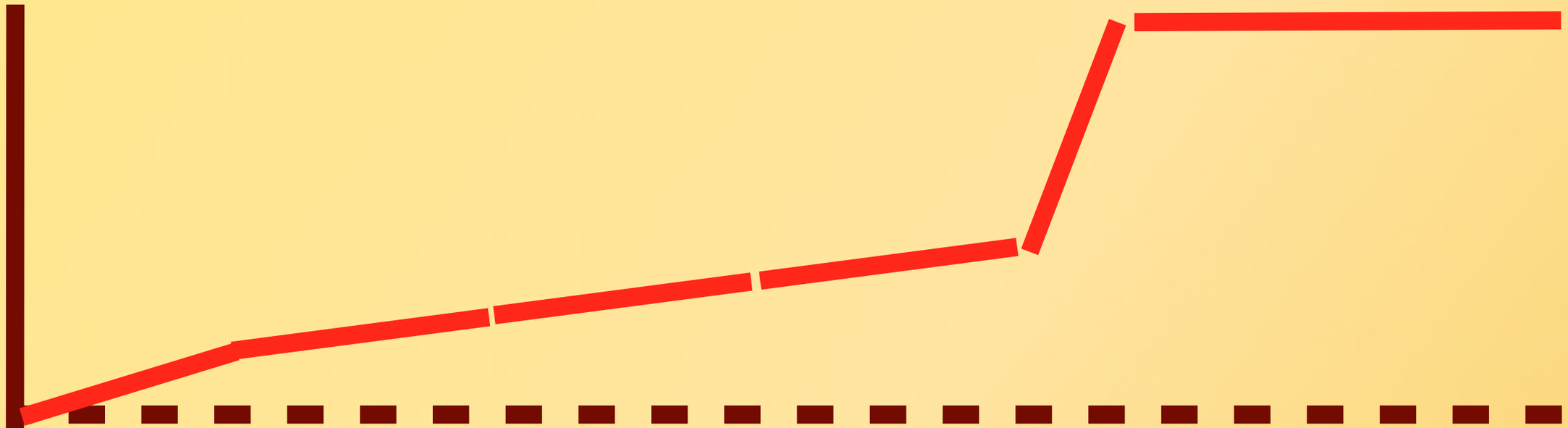
（゜∀゜）



(￣・ω・￣)

関数型での開発(慣れると)

(゜∀゜)



「コンパイルが通ればほとんど実行時エラーが起きない」を体験すると、
コンパイルエラーが起きても「はいはい、ご忠告ありがとね♪」と解釈できるようになり、凹まなくなってくる

(￣・ω・￣) ※何も気をつけなくても実行時エラーがゼロになるということではないです。コンパイラのチェック機能をうまく活かすように書き方に気をつけるという人間側の努力も必要です。

気持ちの変化

- 最初は口うるさく感じられたコンパイラが愛おしくなってくる
- この強力なチェック機能をもっと活用して、どんどん人的ミスが起きる可能性をつぶして行こう！
- 世の中の全てを型で表せばいいんだ！
- エラーはほとんどコンパイラが検出してくれるので、自分はロジックを組むことに集中できる

動的陣営の反論

- えー、型を間違うなんて滅多にないよー。user型が要求されているところにuser以外のデータなんか渡さないし…
- メソッド名の間違いなんてしないよー、エディタが補完してくれるしー。

これ型エラーです！

```
params[:name] => nil
```

あれー、値がとれてくるはずなんだけどなー

これ型エラーです！

```
params[:name] => nil
```

あれー、値がとれてくるはずなんだけどなー

あ！こうか！（シンボルと文字列の間違い）

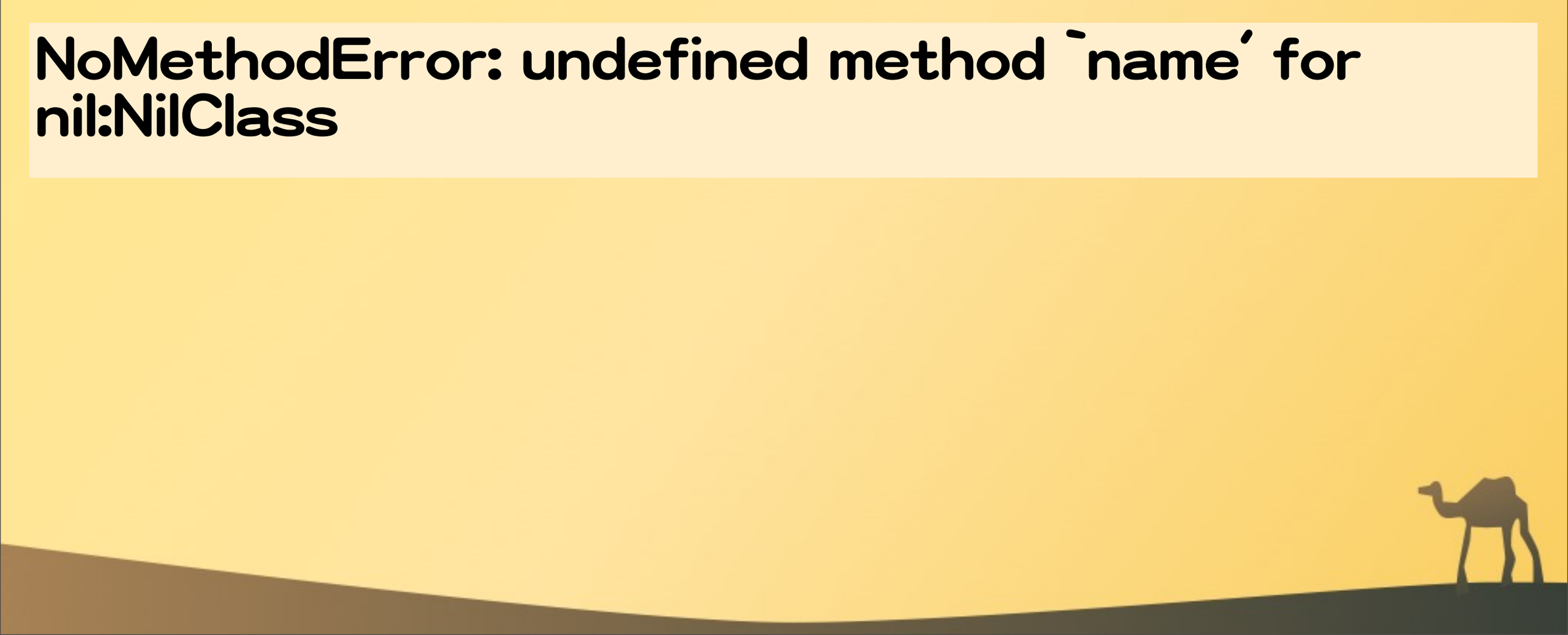
```
params["name"] => "foo"
```

これ型エラーです！

```
<%= user.group.name %>
```

userが属するgroupの名前を表示しようとしたとき。
groupに属していないuserがいると...

```
NoMethodError: undefined method `name' for nil:NilClass
```



明らかな型エラーだと思われているもの(※)以外にも、型エラーとして検出できるエラーがいっぱいあります。

※user型が期待されているところにuser以外のものを渡す、みたいなやつ

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

静的型付き関数型によくある機能

- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

静的型付き関数型によくある機能

- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

静的型付けと型推論

```
File* file = fopen("foo.txt", "r");
int c = fget(file); // コンパイルエラー(正：fgetc)
```

```
file = open("foo.txt", "r")
c = file.get # 実行時エラー(正： getc)
```

```
let file = open_in "foo.txt" in
let c = input_chr file (* コンパイルエラー(正： input_char) *)
```

静的型付けと型推論

言語	型宣言	静的型チェック
C	必要	有
Ruby	不要	無
OCaml	不要	有

⇒不要な型宣言を書かずに本質に集中でき、その後コンパイラでチェックできる！

静的型付き関数型によくある機能

- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

静的型付き関数型によくある機能

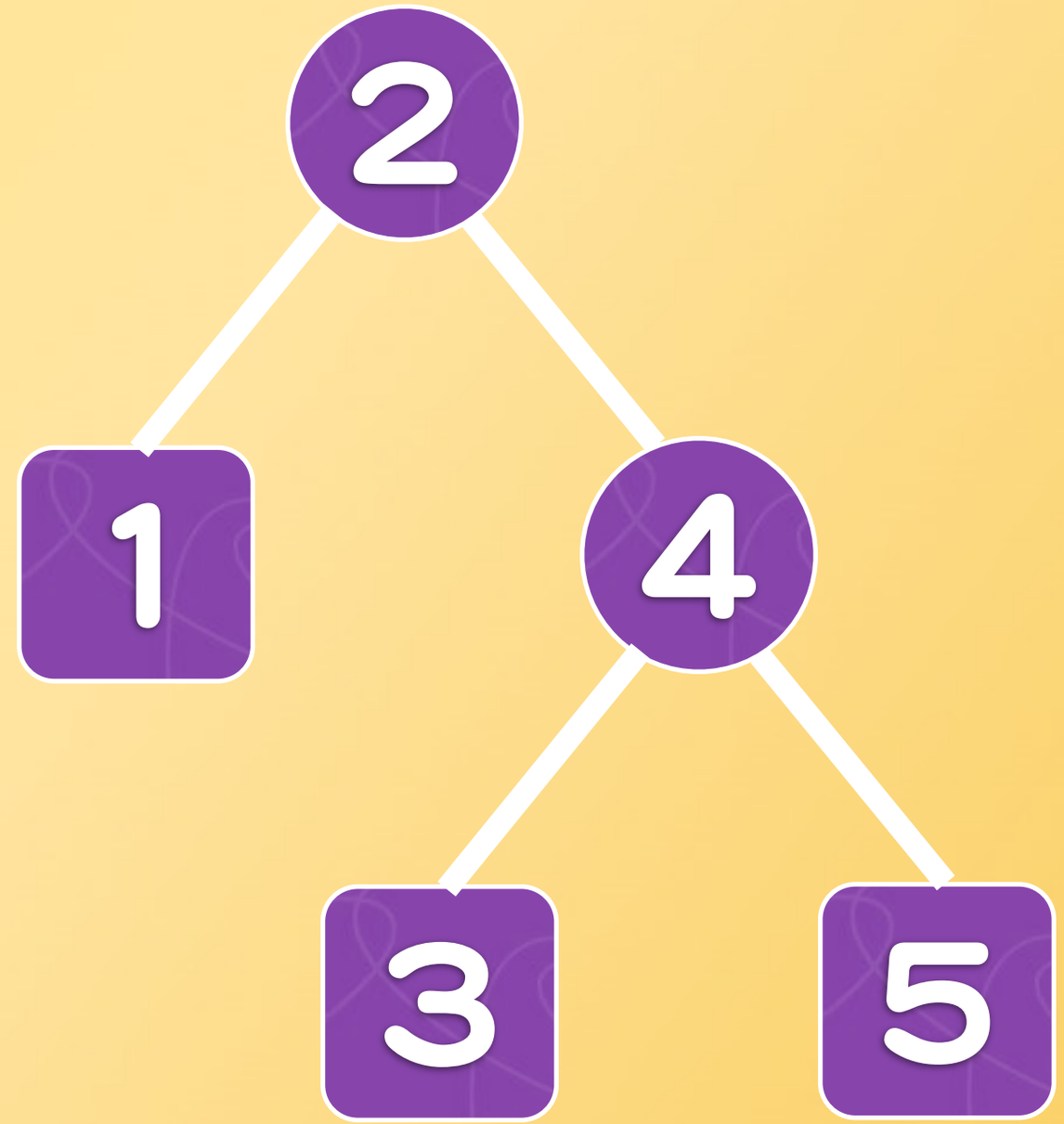
- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

代数的データ型

- 型の直積の直和の総和（日本語でおk？）
- 直積
  - 複数の値の組み合わせ
  - Cの構造体とかOOのValueObjectに相当？
- 直和
  - 複数の値のどれか一つを選択
  - Cの共用体とかOOのCompositeパターンに相当？

Rubyで木

```
class Branch
  attr_accessor :left, :value, :right
  def initialize(left, value, right)
    raise ArgumentError unless
      [Branch, Integer].any?{|t| left.is_a?(t)}
    raise ArgumentError unless
      value.is_a?(Integer)
    raise ArgumentError unless
      [Branch, Integer].any?{|t| right.is_a?(t)}
    @left = left
    @value = value
    @right = right
  end
end
tree = Branch.new(1, 2, Branch.new(3,4,5))
```



※ルーズにやるなら、Hashで済ませてしまうという手もあるが…

```
{:left => 1, :value => 2, :right =>
   {:left => 3, :value => 4, :right => 5}
```

代数的データ型で木

```
type tree_t = Leaf of int
            | Branch of tree_t * int * tree_t
let tree =
    Branch ((Leaf 1), 2,
            (Branch ((Leaf 3), 4, (Leaf 5))))
```

Rubyに慣れているとだいぶ戸惑いますが、型！＝クラスです。
OCamlでは型は小文字で表記します(CやJavaの組み込み型と同じ)。
OCamlではコンストラクタが大文字で始まります。

代数的データ型で木

再帰もオッケー

```
type tree_t = Leaf of int
            | Branch of tree_t * int *tree_t
```

直和　　直積

⇒複雑なデータ構造もシンプルに書き表せ、しかもコンパイラでチェックできる！

静的型付き関数型によくある機能

- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

静的型付き関数型によくある機能

- 強い型付けと型推論
- 代数的データ型
- パターンマッチ

パターンマッチ

- 値だけではなく、値の構造まで比較して分岐できる(Rubyで言う所の)case式
- 構造にマッチさせ、マッチしたものに対して何かの処理をするという意味では、正規表現やXPathを使っているときの感覚に近い。正規表現やXPathでやっているようなことが、プログラミング言語の中で扱っているあらゆるデータ構造に対して行える。

Rubyで再帰で配列の合計

```
def sum(array)
  case
  when []; 0
  else; array.first + sum(array[1..-1])
  end
end
```

※Rubyのcase式も相当高機能なので、関数型にそんなに負けてないですが…

OCamlで再帰で配列の合計

```
let rec sum_list = function
  [] -> 0
  | head::tail -> head + (sum_list tail)
```

パターンマッチによる束縛

```
| head::tail -> head + (sum_list tail)
```

マッチする構造に名前をつけて...

こっちで使えるので読みやすい

Rubyだと...

```
else; array.first + sum(array[1..-1])
```

読みにくい

```
else; head, *tail = array
      head + sum(tail)
```

こういう方法もあるけど、
パターンマッチに比べると冗長

※パターンマッチっぽいことを実現するライブラリもあるが、
無理矢理感は否めない
http://sapporo.rubykaigi.org/2012/ja/schedule/details/11.html

パターンマッチによる網羅性チェック

```
let rec sum_list = function
                (* ←要素が空の場合を考慮し忘れると... *)
  head::tail -> head + (sum_list tail)
```

```
  Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
```

空の場合を忘れてるよー、と教えてくれる

パターンマッチによる網羅性チェック２

0からnまでの合計を求める

```
let rec sum_int = function
  n when n > 0 -> n + sum_int (n - 1)
| n -> 0
```

パターンマッチによる網羅性チェック2

0からnまでの合計を求める

```
let rec sum_int = function
  n when n > 0 -> n + sum_int (n - 1)
        (* <- | n -> 0 のケースを書き忘れると... *)
```

```
Warning 25: bad style, all clauses in this pattern-matching are guarded.
```

条件付きのケースしか記述されてないよー、
と警告してくれる

パターンマッチによる網羅性チェック3

```
type 'a option = None
 | Some of 'a
```

※Haskellで言うところのMaybe

```
let bar = function
  Some n -> print_endline (string_of_int n)
  | None -> print_endline "値なし"
```

パターンマッチによる網羅性チェック3

```
type 'a option = None
  | Some of 'a
```

※Haskellで言うところのMaybe

```
let bar = function
  Some n -> print_endline (string_of_int n)
```

(* ← 値がない場合(None)を書き忘れると、
同様に警告してくれる。)

⇒NULLチェックを忘れることがない！！

※余談だがgcc+enumでも警告してくれるらしい

NULLは10億ドルの失敗

Photograph by Rama, Wikimedia Commons, Cc-by-sa-2.0-fr

http://qconlondon.com/london-2009/presentation/Null+References:+The+Billion+Dollar+Mistake

クイックソートの開発者でもある
著名な計算機科学者
トニー・ホーア氏

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

対話環境

- 関数型でも大抵使えます(もちろん言語によりますが)

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

ダックタイピング

- ダックタイピングは何が嬉しいのか
  - 継承関係にないオブジェクト群を共通に扱える
- OCamlは構造的部分型というのがあるので、ダックタイピングできますがあまり積極的には使わない気が…

```
let foo obj = obj#x;;
val foo : < x : 'a; .. > -> 'a = <fun>
```

※xというメソッドがあるオブジェクトなら
何でもいいよ、という型

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

1.month.ago

```
let (|>) x f = f x
 1 |> month |> ago
```

http://tmaeda.s45.xrea.com/td/20111219.html#p01

- 関数を適用するだけの演算子(|>)を定義できる(UNIXのパイプみたいな感じ)
- Rubyは「全てがオブジェクト」というルールを作ることで、１というただの数字にも特別な機能をつけることができた
- 一方、関数型は関数の繋がり方をプログラミングした

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

obj.methods

- オブジェクトに問い合わせるような考え方ではない
- インストールされているライブラリのインターフェースファイルなどをパースして、モジュール名やメソッド名の一覧を生成する、というやり方になる

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

メタプログラミング

- 実行時にはできないので、コンパイル時にコードやシンタックスツリーを自動生成する（マクロ）ようなやり方になります。camlp4(OCaml)とかTemplate Haskell(Haskell)とか。

```
share [mkPersist sqlSettings, mkMigrate "migrateAll"] [persist|
Person
    name String
    age Int Maybe
    deriving Show
|]
```

HaskellのYesodの例

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

dRubyみたいなの

- 他のプロセスの何らかのインターフェースを叩くということは、大抵は事前にそのインターフェースがわかっているはず。
- なので、大抵はプロトコルやスキーマなどをきっちり型で定義して通信するようなやり方になる。
- プロトコルを扱う関数を自動生成するマクロみたいなのは作れるかも？

私が愛するRubyのアレは？

- 対話環境
- ダックタイピング
- 1.month.ago
- obj.methods.grep(/hoge/)
- メタプログラミング(attr_accessorとか)
- dRubyみたいなの

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

質問回答コーナー

- Rubyの方が高速に処理できること？
  - 実行速度という点で言えば、Rubyに負けることはほぼ無いと思います。
  - 不得意なことという点で言えば、ソースコードを変更できる状況にないオブジェクトなりメソッドなりの挙動を変えるとか、実行したまま挙動を変えるとか(オープンクラス)
- デバッグは？
  - 他の言語と基本的には同じだと思います。printデバッグとかデバッガで値を確認したりとか。

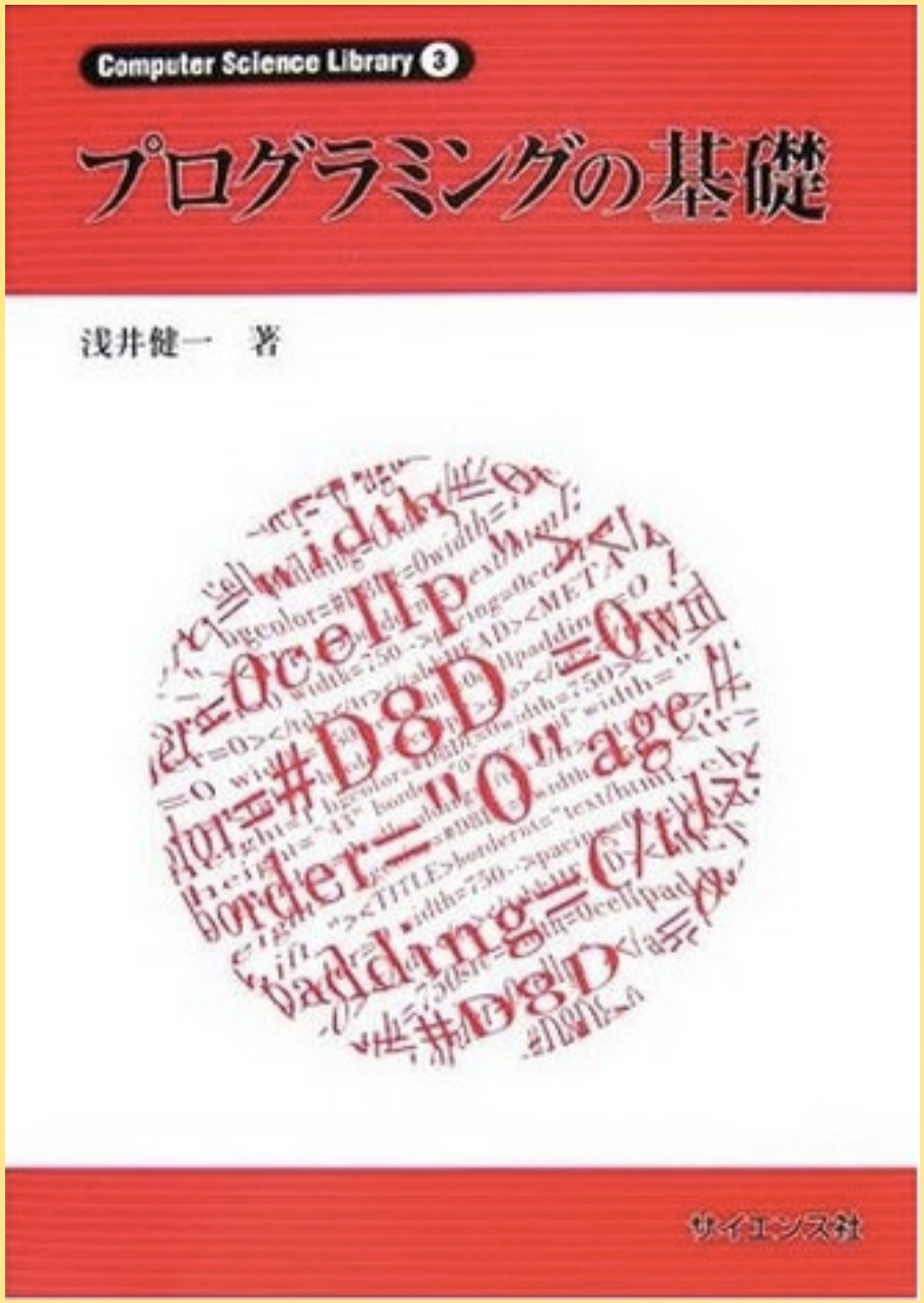
今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

今日のお話

- 関数型言語って何？
- 関数型で開発しているときの気持ち
- 静的型付き関数型によくある機能
- Rubyのアレは関数型でできるの？
- 質問回答コーナー
- おすすめの本とか言語とか

オススメの入門書



OCamlで
プログラミングを
学ぶ本

オススメの入門書

OCamlを
がっつり学ぶ本
(電子書籍で買え
るよ)

オススメの入門書

Haskellを
大変丁寧に学ぶ本

オススメの言語

- Haxe -- 文法が馴染みやすく、今をときめくJavaScriptの代わりとして使える
- OCaml -- モナドとか理解しなくても静的型付き関数型の多くの恩恵を受けられる
- Haskell -- 文法が美しい
- F# -- Windowsの資産が使える
- Scala -- Javaの資産が使える
- SML# -- Cとの親和性が高い。SQLリテラルなどもある。

Haxe3（へっくす）

- 強い静的型付け
- 型推論
- パターンマッチ
- 代数的データ型
- ActionScriptっぽい文法
- いろんな言語に変換できる
- nullはあるけど、パターンマッチと代数的データ型があるので、nullを使わずにコードを書くことが可能

静的型付き関数型言語でもっと楽しましょう！