

PHPer チャレンジ解説 デジタルサーカス

nsfisis (いまむら)

PHPerKaigi 2023

自己紹介

nsfisis (いまむら)

- 今年と昨年の作問を担当
- 昨年の PHPer チャレンジ 1 位

はじめに

解いていただきありがとうございます！

はじめに

解いていただきありがとうございます！

時間の関係上、Q3 と Q4 をメインに解説します

Q1、Q2、Q5 を駆け足で終わらせ、Q3、Q4 の解説へ

Q1

An Art of Computer Programming

解き方



QR コードが書かれた PNG 画像になっている

(QR コードは (株) デンソーウェブの登録商標です。)

解き方



QR コードが書かれた PNG 画像になっている

読み込んでみると、

```
Guess password. $ echo "password" | php Q1.png >/dev/null
```

と表示される

解き方



"Password is one of the PHPer tokens."

解き方



"Password is one of the PHPer tokens."

パスワード: #iwillblog

解き方



"Password is one of the PHPer tokens."

パスワード: #iwilllblog

```
$ echo "#iwilllblog" | php Q1.png >/dev/null
```

トークン: #ModernPHPisStaticallyTypedLanguage



画像中央部



Piet (難解プログラミング言語) のソースコード

1 つ 1 つのピクセルが 1 命令を構成する

画像中央部



Piet (難解プログラミング言語) のソースコード
PHP として実行したときは、Piet のインタプリタ

画像中央部



Piet (難解プログラミング言語) のソースコード

どこで何をやっているの？

- 左上：入力受け付け
- 上の辺、右の辺：パスワードの検証
- 下の辺、左の辺：トークンの出力
- 上の辺の 2 列目：不正解のメッセージ出力

Q2

The False Awakens

概要

Star Wars のパロディ

解き方

There are ~550 "false"s here. Change one of them to "true".

解き方

There are ~550 "false"s here. Change one of them to "true".

想定解 : brute-force (Force の力を信じるのじゃ)

ソースコードの概要

FALSE (難解プログラミング言語) のソースコードとインタプリタ

!![] の形になっているのが FALSE のソースコード

Q5

Are You a Coffee Pot?

概要

RubyKaigi 2022 の TRICK の銀賞作品である "Most interactive code" (Tomoya Ishida (tompng) 氏) にインスパイアされた作品

概要

RubyKaigi 2022 の TRICK の銀賞作品である "Most interactive code" (Tomoya Ishida (tompng) 氏) にインスパイアされた作品

("Most interactive code"はこのトークン問題よりも高度ですが、「アニメーション画像をブラウザで表示する」というアイデアをお借りしています)

解き方

1. PHP のビルトインサーバを使って実行する
2. アニメーション画像が生成されるのを眺める
3. ティーポットの裏にトークンがある

元ネタ

418 I'm a teapot

HTCPCP (Hyper Text Coffee Pot Control Protocol) という HTTP を真似たジョーク RFC で定義されているステータスコード

HTCPCP をティーポットが受信したとき、コーヒープロトコルには対応していない、と返すレスポンス

実装

3D レンダリング + animated PNG 生成

Animated PNG の生成は `imagepng()` などを使わずに、素の PHP で書いている。

**La
ra
ve
l**

概要

Q3 は何をするプログラムか？

概要

Q3 は何をするプログラムか？

任意のプログラムを、1 行がちょうど 2 文字になるように変換するプログラム

ここでは、Laravel を動かしている

1 行 2 文字で書いた 「A」 と出力するプログラム

```
<?  
( '  
x!  
~!  
'^  
'z  
Hd  
G'  
)(  
'K  
'^  
( '  
' )  
) ;
```

1 行 2 文字縛りの何が難しいか

2 文字までのトークンしか使えない

- ほとんどのキーワードが書けない
 - 書けるのは `as`、`do`、`fn`、`if`、`or` のみ
- ほとんどの関数が呼べない
 - 呼べるのは `_` (`gettext` の別名)、`pi`、`dl` のみ
- 空でない文字列が書けない
 - クォート、中身の文字、クォート、で最低 3 文字必要
ではどうするか？

1 行 2 文字に変形する

```
<?php  
echo "A";
```

このコードをベースに変形していく。

1 行 2 文字に変形する

```
<?php  
echo "A";
```

ステップ 1: <?php を短縮する

1 行 2 文字に変形する

```
<?  
echo "A";
```

<?php から <? へ

short_open_tag オプションが必要

1 行 2 文字に変形する

```
<?  
echo "A";
```

ステップ 2: キーワード `echo` を排除する

1 行 2 文字に変形する

```
<?  
printf("A");
```

echo を printf に

1 行 2 文字に変形する

```
<?  
printf("A");
```

ステップ 3: 関数呼び出しを variable function 経由にする

1 行 2 文字に変形する

```
<?  
$p = "printf";  
  
$p("A");
```

printf を一度 \$p に代入して呼び出し

1 行 2 文字に変形する

```
<?  
$p = "printf";  
  
$p("A");
```

ステップ 4: 文字列リテラルをまとめる

1 行 2 文字に変形する

```
<?  
$p = "printf";  
$a = "A";  
$p  
(#  
$a  
);
```

文字列リテラルを代入しているところを無視すれば、1 行 2 文字になった

1 行 2 文字に変形する

```
<?  
$p = "printf";  
$a = "A";  
$p  
(#  
$a  
);
```

ステップ 5: 文字列を幅 2 文字以下で生成する

1 行 2 文字に変形する

```
<?php
$a = "12345";
$b = "world";
// $a ^ $b は次のコードと同じ
$result = '';
for ($i = 0; $i < min(strlen($a), strlen($b)); $i++) {
    $result .= $a[$i] ^ $b[$i];
}
echo $result;
// => F]AXQ
```

文字列の XOR 演算を使う

文字列をバイト列と見做し、各要素に XOR 演算を適用して結合する

1 行 2 文字に変形する

```
<?  
( '  
x!  
~!  
'^ // => "printf"  
'z  
Hd  
G'  
)(  
'K  
'^ // => "A"  
( '  
' )  
) ;
```

1 行 2 文字縛り

- 任意の関数を呼び出せるようになった
- 任意の文字列を書けるようになった

Laravel を動かしていく

1 行 2 文字縛り

- 任意の関数を呼び出せるようになった
- 任意の文字列を書けるようになった

Laravel を動かしていく

これだけだと言語機能が全然足りない！

eval の壁

任意のプログラムを動かすなら eval か？

eval の壁

任意のプログラムを動かすなら eval か？

eval は関数ではないので、`"eval"("...")` と呼べない

eval の壁

eval の壁を突破する方法

eval の壁

eval の壁を突破する方法

PHP の VM の状態を書き換えて、次の命令を強制的に eval にする

eval の壁

eval の壁を突破する方法

PHP の VM の状態を書き換えて、次の命令を強制的に eval にする

PHP の公開 API を FFI で叩く

実行中の opcode を置き換える

```
<?php
```

```
$s = "ソースコード（実際には 2文字幅で生成）";
```

```
// FFI を使って、次に実行する opcode を書き換える
```

```
$s = pi(); // ダミーの命令
```

```
!$s;      // ダミーの命令
```

実行中の opcode を置き換える

```
<?php
```

```
$s = "ソースコード (実際には 2文字幅で生成)";
```

```
// FFI を使って、次に実行する opcode を書き換える
```

```
$s = eval($s);
```

```
return $s;
```

まとめ

- 文字列は XOR 演算で作る
- 関数は variable function で呼び出す
- eval は FFI で opcode を書き換える

任意のプログラムをこの縛りの下で動かせる

トークン

#ThereAreOnlyTwoHardThingsInComputerScience

CacheInvalidationAndNamingThings

- Phil Karlton

キャッシュのバグを解決したとき社内 Slack にいつも貼るやつ

Q4

Quadrilingual PHPer

概要

Ruby、Perl、Bash、PHP の polyglot

Polyglot: 単一のソースコードが複数の言語として解釈可能なプログラム

簡略化版

```
#<?php
false && 1 << 0;
$a='a';
$/* 0;
# */$a
=begin
();1?
0
: //; echo bash; : << 'nil;'
```


簡略化版

```
1;function begin(){  
echo"php\n";$a=<<<nil  
=end  
puts "ruby"  
'  
  
=cut  
print "perl\n"; '# '  
nil;
```

PHP として読む

```
#<?php
```

PHP タグの前の文字列はそのまま出力される

PHP として読む

```
false && 1 << 0;
```

falseかつ1の0回シフト。特に意味はない

PHP として読む

```
$a='a';
```

変数 `$a` に文字列 `'a'` を代入

PHP として読む

```
$/* 0;  
# */$a  
=begin  
();
```

/* から */ はコメント

Variable variable (\$\$a) で変数 \$a に begin() の返り値を代入

PHP として読む

```
1?  
0  
: //; echo bash; : << 'nil;'  
1;
```

// からは後ろはコメント

コメントや改行を取り除くと、`1 ? 0 : 1;`。式に特に意味はない

PHP として読む

```
function begin(){}
```

先ほど呼んでいた `begin()` を定義

PHP として読む

```
echo "php\n"
```

ここは普通の PHP

PHP として読む

```
$a=<<<nil
=end
puts "ruby"
'

=cut
print "perl\n"; '# ';
nil;
```

nil をデリミタとして使ったヒアドキュメント

Ruby として読む

```
#<?php
```

#はコメント

Ruby として読む

```
false && 1 << 0;
```

falseかつ1の0回シフト。特に意味はない

Ruby として読む

```
$a='a';
```

変数 `$a` に文字列 `'a'` を代入

Ruby の変数には通常 `$` を付けないが、付けるとグローバル変数になる

Ruby として読む

```
$/* 0;
```

変数 `$/` に `0` をかける

`$/` は Ruby に用意されている特殊なグローバル変数

Ruby として読む

```
# */$a
```

#はコメント

Ruby として読む

```
=begin  
();1?  
0  
: //; echo bash; : << 'nil;'  
1;function begin(){  
echo"php\n";$a=<<<nil  
=end
```

=begin から =end は複数行コメント

Ruby として読む

```
puts "ruby"
```

普通の Ruby

Ruby として読む

```
'  
=cut  
print "perl\n"; '# ';
```

' から ' は文字列リテラル

' ; はコメント

Ruby として読む

```
nil;
```

nil は PHP でいう null

Perl として読む

ほぼ Ruby と同じ

違うのは、`=begin` から `=end` ではなく、`=begin` から `=cut` までがコメントである点

Bash として読む

```
#<?php
```

#はコメント

Bash として読む

```
false && 1 << 0;
```

false コマンドを実行し、成功すれば 1 コマンドを実行する

Bash として読む

```
false && 1 << 0;
```

false コマンドを実行し、成功すれば 1 コマンドを実行する

false は、絶対に失敗するコマンド

Bash として読む

```
false && 1 << 0;
```

<< 0 はヒアドキュメント。次に 0 が来るまでのテキストを、1 コマンドの標準入力に送る

Bash として読む

```
false && 1 << 0;  
$a='a';  
$/* 0;  
# */$a  
=begin  
( );1?  
0
```


Bash として読む

```
: //; echo bash; : << 'nil;'
```

- ：は何もしないコマンド
- ：コマンドに引数 // を指定して実行

Bash として読む

```
: //; echo bash; : << 'nil;'
```

echo bash; は普通の Bash

Bash として読む

```
: //; echo bash; : << 'nil;'
```

その次の `:` も何もしない

`<< 'nil;'` もヒアドキュメント (デリミタ: `nil;`)

Bash として読む

```
: //; echo bash; : << 'nil;'  
1;function begin(){  
echo"php\n";$a=<<<nil  
=end  
puts "ruby"  
'  
  
=cut  
print "perl\n"; '# '  
nil;
```

トークンの元ネタ

トークン: `#EngineeringIsProgrammingIntegratedOverTime`

(エンジニアリングはプログラミングの時間積分である)

『Software Engineering at Google』の中で出てくるやたらカッコイイ言葉

おわりに

改めて、解いていただきありがとうございました！

おまけ

枚数削減前のスライド

Q1

An Art of Computer Programming

タイトル

"An Art of Computer Programming"

"The Art of Computer Programming" (Knuth) のパロディタイトル

解き方



QR コードが書かれた PNG 画像になっている

(QR コードは (株) デンソーウェブの登録商標です。)

解き方



QR コードが書かれた PNG 画像になっている

読み込んでみると、

```
Guess password. $ echo "password" | php Q1.png >/dev/null
```

と表示される

解き方



"Password is one of the PHPer tokens."

解き方



"Password is one of the PHPer tokens."

パスワード: #iwillblog

解き方



"Password is one of the PHPer tokens."

パスワード: #iwilllblog

```
$ echo "#iwilllblog" | php Q1.png >/dev/null
```

トークン: #ModernPHPisStaticallyTypedLanguage

なぜ実行できるのか

- PNG 画像の構造 : ヘッダ、画像データ、フッタ

なぜ実行できるのか

- PNG 画像の構造 : ヘッダ、画像データ、フッタ
 - フッタの後ろのデータは無視される

なぜ実行できるのか

- PNG 画像の構造 : ヘッダ、画像データ、フッタ
 - フッタの後ろのデータは無視される
- PHP ソースコードの構造 : PHP タグ、プログラム

なぜ実行できるのか

- PNG 画像の構造 : ヘッダ、画像データ、フッタ
 - フッタの後ろのデータは無視される
- PHP ソースコードの構造 : PHP タグ、プログラム
 - PHP タグの前のデータはそのまま標準出力に出力される

なぜ実行できるのか

- PNG 画像の構造 : ヘッダ、画像データ、フッタ
 - フッタの後ろのデータは無視される
- PHP ソースコードの構造 : PHP タグ、プログラム
 - PHP タグの前のデータはそのまま標準出力に出力される

1. PNG のヘッダ
2. PNG の画像データ
3. PNG のフッタ
4. PHP タグ
5. PHP プログラム

画像中央部の秘密



画像中央部の秘密



Piet (難解プログラミング言語) のソースコード

1 つ 1 つのピクセルが 1 命令を構成する

画像中央部の秘密



Piet (難解プログラミング言語) のソースコード

Q: どこで何をやっているの? A: わからん

画像中央部の秘密



Piet (難解プログラミング言語) のソースコード

Q: どこで何をやっているの? A: わからん

- 左上: 入力受け付け
- 上の辺、右の辺: パスワードの検証
- 下の辺、左の辺: トークンの出力
- 上の辺の 2 列目: 不正解のメッセージ出力

小ネタ



実はパスワードが違ふときのメッセージを間違えていた

- 今の挙動: 401 Unauthorized
- 以前の挙動: 403 Forbidden

小ネタ



実はパスワードが違うときのメッセージを間違えていた

Q: 直すだけの画像の余白が足りない。どう直す？

A: メッセージ出力前にインタプリタで書き換える

```
fwrite(  
    STDERR,  
    str_replace('403 Forbidden', '401 Unauthorized', $0),  
    );
```

小ネタ

なぜパスワードを `#iwilllblog` にしたか？

このトークン問題は、PHPerKaigi 2022 の終了直後から作り始めていた
何がトークンに選ばれるかわからないので、毎年使われているこれにした

Q2

The False Awakens

概要

Star Wars のパロディ

解き方

There are ~550 "false"s here. Change one of them to "true".

解き方

There are ~550 "false"s here. Change one of them to "true".

想定解 : brute-force (Force の力を信じるのじゃ)

解き方

A long time ago in a galaxy far,
far away....

解き方

```
##### # # ##### #####  
# # # # # #  
##### ##### ##### #####  
# # # # # #  
# # # # ##### # #####
```

```
##### # # ### #####  
# # # # # #  
### ##### # # ###  
# # # # # #  
# ##### # ### #####
```


解き方

Episode VII
THE FALSE AWAKENS

#May_the_False_be_with_you

ソースコードの概要

FALSE (難解プログラミング言語) のソースコードとインタプリタ

- `!![]` の形になっているのが FALSE のソースコード
 - Xorshift を実装している
 - 今回 `false` から `true` にした場所が、乱数のシード値に対応
 - シード値を 0 にするとランダムイズがおこなわれなくなる
- `false()` 関数が FALSE のインタプリタ

小ネタ

```
assert(8 === PHP_INT_SIZE, "略");
```

このソースコードに書かれた比較演算は、「ヨーダ記法」になるようにした

小ネタ

```
assert(8 === PHP_INT_SIZE, "...");
```

このソースコードに書かれた比較演算は、「ヨーダ記法」になるようにした

ヨーダ記法: `1 === $x` のように、定数を左辺に置く書き方のこと

ヨーダは Star Wars の登場人物で、緑色の小柄な老人

小ネタ

```
$FALSe = $faLSe*+(!false.!false)*!false;
```

書き換える「正解」の変数 \$FALSe は、書き換え前の値だと 66 になっている

小ネタ

```
$FALSe = $faLSe*+(!false.!false)*!false;
```

書き換える「正解」の変数 \$FALSe は、書き換え前の値だと 66 になっている
これは、「オーダー 66」にちなんでいる

**La
ra
ve
l**

概要

Q3 は何をするプログラムか？

概要

Q3 は何をするプログラムか？

任意のプログラムを、1 行がちょうど 2 文字になるように変換するプログラム
ここでは、Laravel を動かしている

1 行 2 文字で書いた 「A」 と出力するプログラム

```
<?  
( '  
x!  
~!  
'^  
'z  
Hd  
G'  
)(  
'K  
'^  
( '  
' )  
) ;
```

1 行 2 文字縛りの何が難しいか

2 文字までのトークンしか使えない

- ほとんどのキーワードが書けない
 - 書けるのは `as`、`do`、`fn`、`if`、`or` のみ
- ほとんどの関数が呼べない
 - 呼べるのは `_` (`gettext` の別名)、`pi`、`dl` のみ
- 空でない文字列が書けない
 - クォート、中身の文字、クォート、で最低 3 文字必要
ではどうするか？

1 行 2 文字に変形する

```
<?php  
echo "A";
```

このコードをベースに変形していく。

1 行 2 文字に変形する

```
<?php  
echo "A";
```

ステップ 1: <?php を短縮する

1 行 2 文字に変形する

```
<?  
echo "A";
```

<?php から <? へ

short_open_tag オプションが必要

1 行 2 文字に変形する

```
<?  
echo "A";
```

ステップ 2: キーワード `echo` を排除する

1 行 2 文字に変形する

```
<?  
printf("A");
```

echo を printf に

1 行 2 文字に変形する

```
<?  
printf("A");
```

ステップ 3: 関数呼び出しを variable function 経由にする

1 行 2 文字に変形する

```
<?  
$p = "printf";  
  
$p("A");
```

printf を一度 \$p に代入して呼び出し

1 行 2 文字に変形する

```
<?  
$p = "printf";  
  
$p("A");
```

ステップ 4: 文字列リテラルをまとめる

1 行 2 文字に変形する

```
<?  
$p = "printf";  
$a = "A";  
$p  
(#  
$a  
);
```

文字列リテラルを代入しているところを無視すれば、1 行 2 文字になった

1 行 2 文字に変形する

```
<?  
$p = "printf";  
$a = "A";  
$p  
(#  
$a  
);
```

ステップ 5: 文字列を幅 2 文字以下で生成する

1 行 2 文字に変形する

```
<?php
$a = "12345";
$b = "world";
// $a ^ $b は次のコードと同じ
$result = '';
for ($i = 0; $i < min(strlen($a), strlen($b)); $i++) {
    $result .= $a[$i] ^ $b[$i];
}
echo $result;
// => F]AXQ
```

文字列の XOR 演算を使う

文字列をバイト列と見做し、各要素に XOR 演算を適用して結合する

1 行 2 文字に変形する

```
<?  
( '  
x!  
~!  
'^  
'z  
Hd  
G'  
)(  
'K  
'^  
( '  
' )  
) ;
```

1 行 2 文字に変形する

```
<?  
( '  
x!  
~!  
'^ // => "printf"  
'z  
Hd  
G'  
)(  
'K  
'^ // => "A"  
( '  
' )  
) ;
```


1 行 2 文字縛り

- 任意の関数を呼び出せるようになった
- 任意の文字列を書けるようになった

Laravel を動かしていく

1 行 2 文字縛り

- 任意の関数を呼び出せるようになった
- 任意の文字列を書けるようになった

Laravel を動かしていく

これだけだと言語機能が全然足りない！

eval の壁

任意のプログラムを動かすなら `eval` か？

eval の壁

任意のプログラムを動かすなら eval か？

eval は関数ではないので、`"eval"("...")` と呼べない

eval の壁

eval の壁を突破する方法

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する
 - Laravel が動くクオリティで

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する
 - Laravel が動くクオリティで
 - PHPPerKaigi 2023 に間に合うように

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する
 - Laravel が動くクオリティで
 - PHPPerKaigi 2023 に間に合うように
 - nikic 氏を 5 人用意してください

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する
 - Laravel が動くクオリティで
 - PHPPerKaigi 2023 に間に合うように
 - nikic 氏を 5 人用意してください
- PHP の VM の状態を書き換えて、次の命令を強制的に `eval` にする

eval の壁

eval の壁を突破する方法

- 任意の関数が呼べることを利用して、PHP の処理系を頑張って実装する
 - Laravel が動くクオリティで
 - PHPerKaigi 2023 に間に合うように
 - nikic 氏を 5 人用意してください
- PHP の VM の状態を書き換えて、次の命令を強制的に `eval` にする
 - PHP の公開 API を FFI で叩く

実行中の opcode を置き換える

```
<?php
```

```
$s = "ソースコード (実際には 2文字幅で生成)";
```

```
// FFI を使って、次に実行する opcode を書き換える
```

```
$s = pi(); // ダミーの命令
```

```
!$s;      // ダミーの命令
```

実行中の opcode を置き換える

```
<?php
```

```
$s = "ソースコード (実際には 2文字幅で生成)";
```

```
// FFI を使って、次に実行する opcode を書き換える
```

```
$s = eval($s);
```

```
return $s;
```

まとめ

- 文字列は XOR 演算で作る
- 関数は variable function で呼び出す
- eval は FFI で opcode を書き換える

任意のプログラムをこの縛りの下で動かせる

トークン取得の裏ルート

FFI を使う関係上、バージョン依存が激しい

パッチバージョンが変わるだけで動かなくなる

動かない環境でもトークンが得られるよう、裏ルートを作っている

トークン取得の裏ルート

トークンは、`__halt_compiler`を使って、ソースコードの最後に置いてあるデータは Base64 でエンコードされている

トークン取得の裏ルート

```
[
  "<?php" => " ",
  "$this->host().':'. $this->port()," =>
    "$this->host().':'. $this->port(),"-d","ffi.enable=1"," ↵
    ↵ -d","opcache.enable=0","-d","short_open_tag=1","",
  "'9.47.0'" =>
    "'9.47.0 #ThereAreOnlyTwoHardThingsInComputerScienceCa ↵
    ↵ cheInvalidationAndNamingThings'",
]
```

トークン

#ThereAreOnlyTwoHardThingsInComputerScience

CacheInvalidationAndNamingThings

- Phil Karlton

キャッシュのバグを解決したとき社内 Slack にいつも貼るやつ

Q4

Quadrilingual PHPer

概要

Ruby、Perl、Bash、PHP の polyglot

Polyglot: 単一のソースコードが複数の言語として解釈可能なプログラム

簡略化版

```
#<?php
false && 1 << 0;
$a='a';
$/* 0;
# */$a
=begin
();1?
0
: //; echo bash; : << 'nil;'
```

簡略化版

```
1;function begin(){  
echo"php\n";$a=<<<nil  
=end  
puts "ruby"  
'  
  
=cut  
print "perl\n"; '# '  
nil;
```

PHP として読む

```
#<?php
```

PHP タグの前の文字列はそのまま出力される

PHP として読む

```
false && 1 << 0;
```

falseかつ1の0回シフト。特に意味はない

PHP として読む

```
$a='a';
```

変数 `$a` に文字列 `'a'` を代入

PHP として読む

```
$/* 0;  
# */$a  
=begin  
());
```

/* から */ はコメント

Variable variable (\$\$a) で変数 \$a に begin() の返り値を代入

PHP として読む

```
1?  
0  
: //; echo bash; : << 'nil;'  
1;
```

// からは後ろはコメント

コメントや改行を取り除くと、`1 ? 0 : 1;`。式に特に意味はない

PHP として読む

```
function begin(){}
```

先ほど呼んでいた `begin()` を定義

PHP として読む

```
echo "php\n"
```

ここは普通の PHP

PHP として読む

```
$a=<<<nil
=end
puts "ruby"
'

=cut
print "perl\n"; '# ';
nil;
```

nil をデリミタとして使ったヒアドキュメント

Ruby として読む

```
#<?php
```

#はコメント

Ruby として読む

```
false && 1 << 0;
```

falseかつ1の0回シフト。特に意味はない

Ruby として読む

```
$a='a';
```

変数 `$a` に文字列 `'a'` を代入

Ruby の変数には通常 `$` を付けないが、付けるとグローバル変数になる

Ruby として読む

```
$/* 0;
```

変数 `$/` に `0` をかける

`$/` は Ruby に用意されている特殊なグローバル変数

Ruby として読む

```
# */$a
```

#はコメント

Ruby として読む

```
=begin  
();1?  
0  
: //; echo bash; : << 'nil;'  
1;function begin(){  
echo"php\n";$a=<<<nil  
=end
```

=begin から =end は複数行コメント

Ruby として読む

```
puts "ruby"
```

普通の Ruby

Ruby として読む

```
'  
=cut  
print "perl\n"; '# ';
```

' から ' は文字列リテラル

' ; はコメント

Ruby として読む

```
nil;
```

nil は PHP でいう null

Perl として読む

ほぼ Ruby と同じ

違うのは、`=begin` から `=end` ではなく、`=begin` から `=cut` までがコメントである点

Bash として読む

```
#<?php
```

#はコメント

Bash として読む

```
false && 1 << 0;
```

false コマンドを実行し、成功すれば 1 コマンドを実行する

Bash として読む

```
false && 1 << 0;
```

false コマンドを実行し、成功すれば 1 コマンドを実行する

false は、絶対に失敗するコマンド

Bash として読む

```
false && 1 << 0;
```

<< 0 はヒアドキュメント。次に 0 が来るまでのテキストを、1 コマンドの標準入力に送る

Bash として読む

```
false && 1 << 0;  
$a='a';  
$/* 0;  
# */$a  
=begin  
( );1?  
0
```

Bash として読む

```
: //; echo bash; : << 'nil;'
```

- : は何もしないコマンド
- : コマンドに引数 // を指定して実行

Bash として読む

```
: //; echo bash; : << 'nil;'
```

echo bash; は普通の Bash

Bash として読む

```
: //; echo bash; : << 'nil;'
```

その次の `:` も何もしない

`<< 'nil;'` もヒアドキュメント (デリミタ: `nil;`)

Bash として読む

```
: //; echo bash; : << 'nil;'  
1;function begin(){  
echo"php\n";$a=<<<nil  
=end  
puts "ruby"  
'  
  
=cut  
print "perl\n"; '# '  
nil;
```

トークンの元ネタ

トークン: #EngineeringIsProgrammingIntegratedOverTime

(ソフトウェアエンジニアリングはプログラミングの時間積分である)

『Software Engineering at Google』の中で出てくるやたらカッコイイ言葉。Google社内で使われているらしいが、初出がここかは不明。原文を確認したところ"Software engineering is programming integrated over time."だった。

Q5

Are You a Coffee Pot?

概要

RubyKaigi 2022 の TRICK の銀賞作品である "Most interactive code" (Tomoya Ishida (tompng) 氏) にインスパイアされた作品

概要

RubyKaigi 2022 の TRICK の銀賞作品である "Most interactive code" (Tomoya Ishida (tompng) 氏) にインスパイアされた作品

("Most interactive code"はこのトークン問題よりも高度ですが、「アニメーション画像をブラウザで表示する」というアイデアをお借りしています)

解き方

1. PHP のビルトインサーバを使って実行する
2. アニメーション画像が生成されるのを眺める
3. ティーポットの裏にトークンがある

元ネタ

418 I'm a teapot

HTCPCP (Hyper Text Coffee Pot Control Protocol) という HTTP を真似たジョーク RFC で定義されているステータスコード

元ネタ

418 I'm a teapot

HTCPCP (Hyper Text Coffee Pot Control Protocol) という HTTP を真似たジョーク RFC で定義されているステータスコード

HTCPCP をティーポットが受信したとき、コーヒープロトコルには対応していない、と返すレスポンス

実装

3D レンダリング + animated PNG 生成

Animated PNG の生成は `imagepng` などを使わずに、素の PHP で書いている。

ティーポットのオブジェクトは Base64 でエンコードされており、`__halt_compiler` を使ってソースの後ろに埋め込まれている。

小ネタ

```
echo base64_decode(  
  'PGhlYWQ+PHRpdGxkknbSBhIHRlYXBvdDwvdGl0bGU+PC9oZWFKPjxw ↵  
  ↵ PjxpbWcgc3JjPS90ZWFWb3Q'  
  . 'ucG5nIHdpZHRoPTQxOCBoZWlnaHQ9NDE4PjwvcD48c2NyaXB0PmNvb ↵  
  ↵ nNvbGUubG9nKGF0b2IoJw== '  
  . 'bm8gdG9rZW4gaGVyZSA7LSk='  
  . base64_decode('JykpPC9zY3JpcHQ+') ;
```

いかにもな Base64 でエンコードされた文字列。デコードすると

小ネタ

```
<head>
  <title>I'm a teapot</title>
</head>
<p>
  <img src=/teapot.png width=418 height=418>
</p>
<script>
  console.log(atob('bm8gdG9rZW4gaGVyZSA7LSk='))
</script>
```

この怪しげなスクリプトは？