Ruby入門1

~*Ruby*の基礎を学んで使ってみよう~

課外プロジェクト 2014 4/27

d8161105 渡部未来

目的

- *Ruby*の特徴と基本的な書き方を学ぶ
- プログラミング言語の様々な特徴を学び、メリット・デメリットを学ぶ
- プログラミング言語の得手不得手を考える

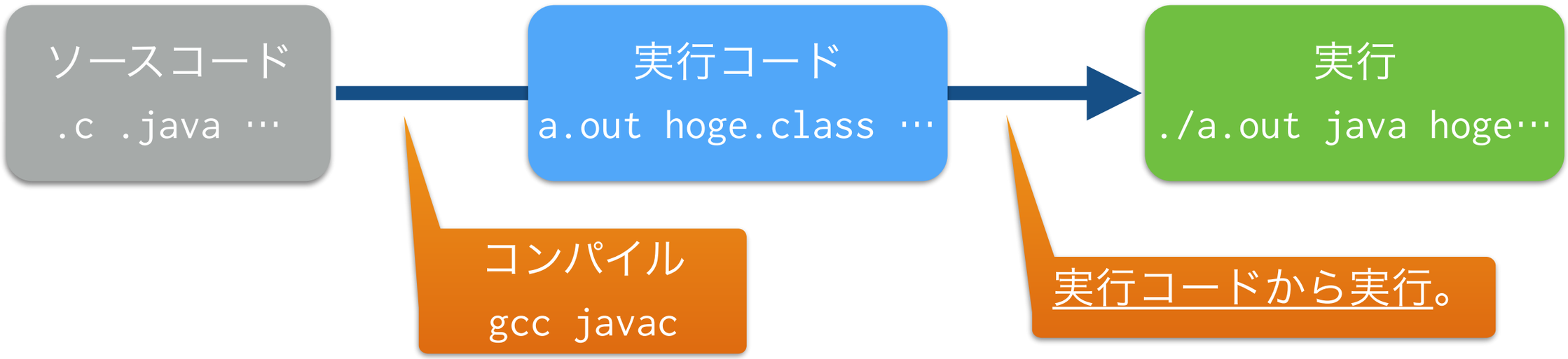
※本資料は、Ruby 1.9.3について解説する。
リファレンスマニュアル: http://docs.ruby-lang.org/ja/1.9.3/doc/index.html

プログラミング言語 *Ruby*

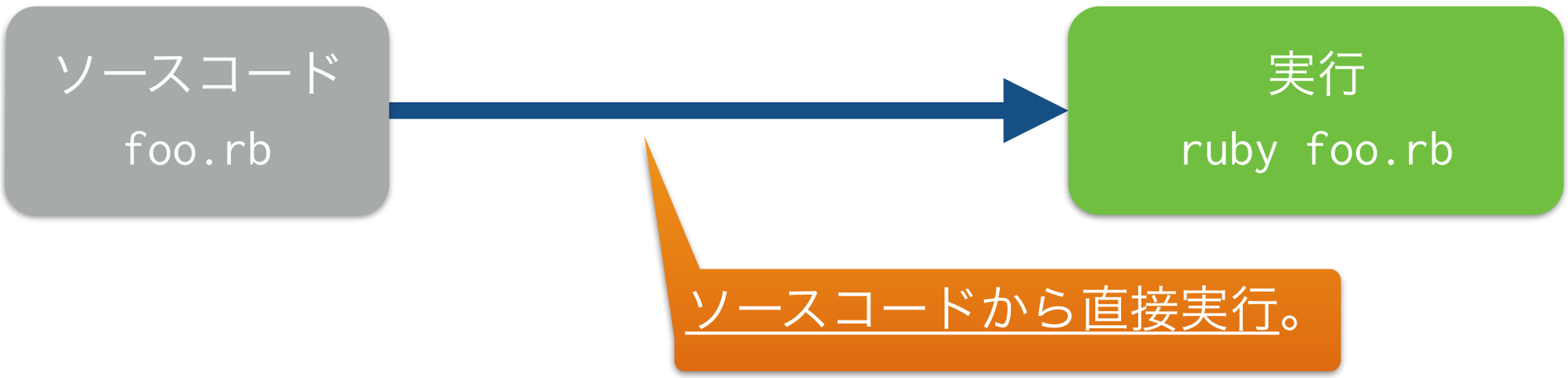
- まつもとゆきひろさん(*Matz*)によって開発された汎用プログラミング言語
- 国産のプログラミング言語なので日本語資料が多い
- インタプリタ言語
- 動的型付け言語
- 文法・ライブラリが豊富
- 純粋なオブジェクト指向言語
- *Web*アプリケーション・フレームワーク ***Ruby on Rails***によりヒット

Ruby | インタプリタ言語

コンパイラ言語:



インタプリタ言語(*Ruby*):



Ruby | 動的型付け言語

静的型付け言語：

型宣言をする。(例外あり)
コンパイル時に型が決まる。

```
int x;   // 整数型
char c; // 文字型
```

例: *Fortran, C, C++, Java, Objective-C*

動的型付け言語 (*Ruby*)：

変数に代入された値で型が決まる。
実行時に型が決まる。

```
x = 1          // 整数型
x =  'aa'      // 文字型
```

例: *JavaScript, Perl, Python, PHP, Ruby*

Ruby | オブジェクト指向言語

- *Ruby*は**純粋なオブジェクト指向言語**
- データは、**すべてオブジェクト**である
- オブジェクトは**アイデンティティ**を持つ
- オブジェクトは**状態 (Field)**と**振る舞い (Method)**を持つ
- 状態は、いわゆる変数のこと
- 振る舞いは、いわゆる関数のこと

非オブジェクト指向言語:

データの計算処理は、
関数・演算子に任せる。

```
strlen( “string” )    // 6
atoi( “123” )         // 123
toupper( “abc” )      // ABC
```

オブジェクト指向言語 (*Ruby*):

データはオブジェクト。
オブジェクトの計算処理は、
フィールド・メソッドを使用する。

```
“string” .length     // 6
“123” .to_i          // 123
“abc” .upcase        // ABC
```

オブジェクト指向 | サンプル

- 簡単な集計プログラムを考える
- csv (カンマ区切りの値)ファイルを読み込み、二次元配列へ落とし込む
- 二次元配列から集計をする

オブジェクト指向 | 非オブジェクト指向 その1

- グローバル変数は、いつ・どこで書き換えられているか分からないため危険
- もし、扱うcsvが複数になったら…？ (row1, row2などのグローバル変数が増加、管理が難しくなる)

```
#define N 100
// グローバル変数
int row;
int column;

void input(int[][]);
int calc(int[][]);

int main(){
    int res;
    int col[N][N];
    input(col); // row, columnが確定
    res = calc(col);
}
```

オブジェクト指向 | 非オブジェクト指向 その2

- 構造体で扱いたいメンバが増加していくと管理が困難に
- 作成した構造体を扱う専用の関数を用意しなければならない

```
#define N 100
typedef struct {
    int col[N][N];
    int row, column;
} Grade;
void input(Grade*);
int calc(Grade*);

int main(){
    int res;
    Grade *grade;
    input(grade);
    res = calc(grade);
}
```

オブジェクト指向 | オブジェクト指向(*Ruby*)

- フィールドは、グローバル変数と異なり自身のクラスにスコープが限定される
- 振る舞いを定義することで、専用の関数を作ることができる
- 新しい型を作ったので、インスタンス化することで量産が可能

```
class Grade
    def initialize(csv)             // コンストラクタ
        @data = csv.getDate         // csvデータから二次元配列を取得
                                    // 本来は自分で実装
        @row = csv.getRow           // 行数を取得、本来は自分で実装
        @column = csv.getColumn     // 列数を取得、本来は自分で実装
    end
    def calc
        @row                        // フィールドはクラス内なら自由に取得できる
        @column
    end
end

grade = new Grade(csv)              // インスタンス化すれば、複数csvも扱える
res = grade.calc
```