

# 仕様とコーディング

仕様を正しく読み取り、綺麗なコーディングを目指そう

課外プロジェクト資料 2014 4/22

d8161105 渡部未来



## ◇ 目的

- ▶ 通常の場合、いきなり何も考えずにコーディング・開発などはできない
- ▶ そこで開発したい対象から何を作りたいかを考え(顧客がいる場合はヒアリング)**仕様書**・**設計書**をまとめる
- ▶ 仕様書・設計書から**矛盾なく**プログラムへ変換することが大切

## ◇ 仕様書

- ▶ **仕様**とは、材料・製品・サービスが**明確に満たさなければならない要求事項**である
  - ▶ 仕様を記述した文書を**仕様書**と呼ぶ
- 引用：wikipedia
- ▶ 授業で言えば、課題の問題文が仕様書であり、課題そのものが**成果物**と言える

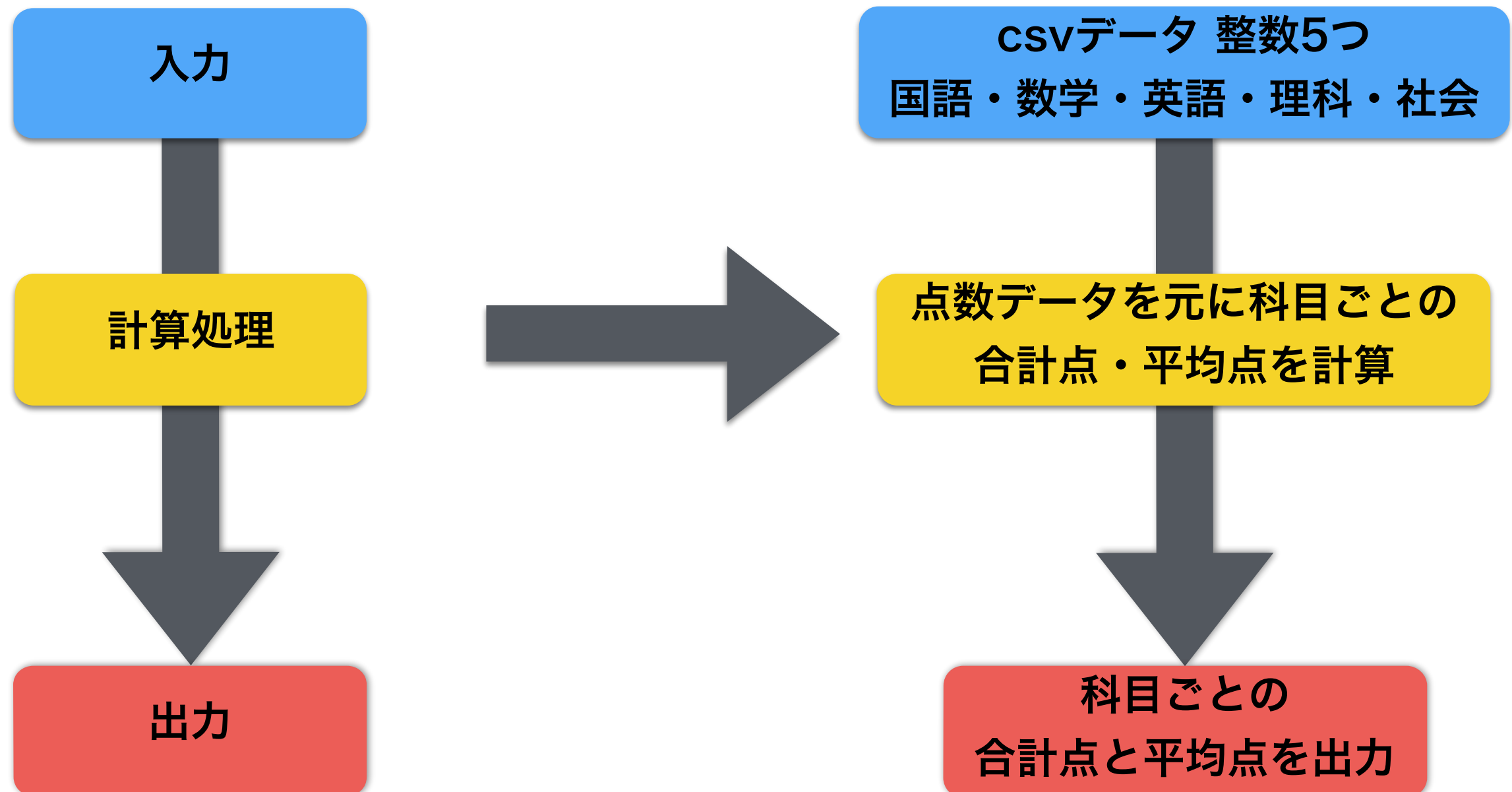
## ◇ 仕様書 | 要求を読み取る

仕様書、または設計書などは、決まった書き方などはない。ただし簡易なものを作るときや読み取る場合などは、以下のような処理の流れをパターンとして捉えることができる。

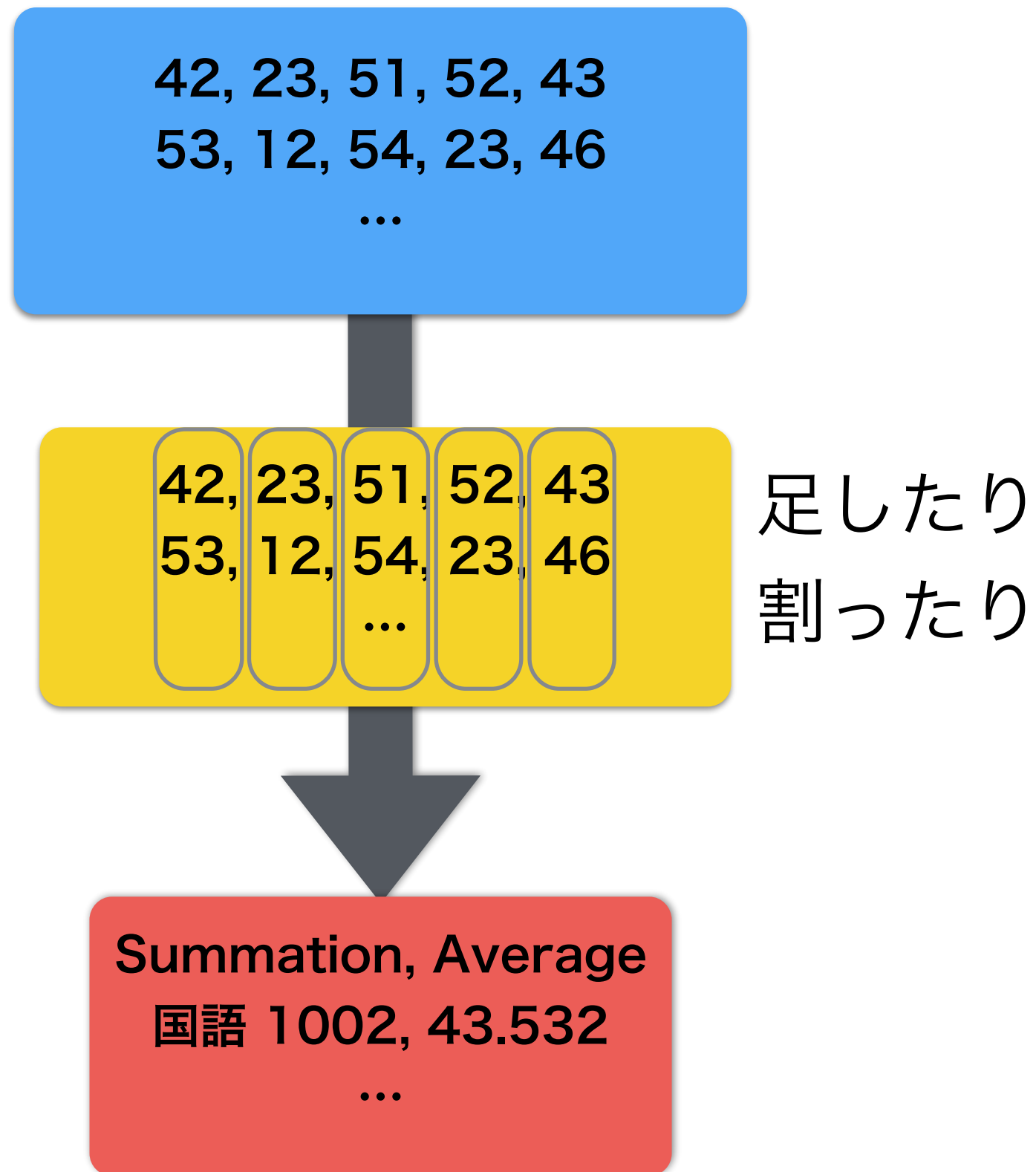


## ◇ 仕様書 | 要求を読み取る 簡単な例

学生の科目ごとの合計点と平均点を算出し、表示せよ。入力はcsv(カンマ区切りのデータ)で、国語・数学・英語・理科・社会の順で与えられる。なお、csvファイルが見つからない場合をエラー文を表示せよ。



## ◇ 仕様書 | 要求を読み取る 簡単な例



## ◇ 仕様書 | 要求を読み取る まとめ

- ▶ 処理の流れをイメージする = なるべく分割する
- ▶ 仕様書からキーワードを抜き出し、流れに当てはめる
- ▶ キーワードからイメージを具体化する
- ▶ 具体化する際に、プログラムに落としては駄目！  
表・図・数式・簡単な言葉だけに留めること！

## ◇ コーディング

- ▶ 仕様書・設計書からプログラムへと変換する
- ▶ やみくもに変換するのではなく、仕様書・設計書から使われる**変数の名前**・**型**などを具体的に書き出す
- ▶ 大きな処理の流れを`main`関数に書き、意味のある処理は**関数へまとめる**
- ▶ 綺麗なプログラムとは、処理の流れが**簡潔**でわかりやすく、変数名、関数名から**意味が汲み取りやすい**



## ◇ コーディング | 意味のある変数名

悪い例：

```
// 教科の合計を算出  
int a, b, c, d, e, f=0;  
  
f = a + b + c + d + e;
```

変数名に意味がなく、  
コメントがあっても不明瞭。

良い例：

```
int jpn, mth, eng, sci, soc;  
int sum = 0;  
  
sum = jpn + mth + eng + sci + soc;
```

**意味のある変数名**ならば、  
コメントの必要もない。

# ◇ コーディング | 関数

悪い例：

```
// aが偶数かつbが奇数のとき真
if( a%2 == 0 && b%2 != 0){
    ...
}
```

条件が複雑になればなるほど、  
コメントの説明も長くなり、解読が難しい。

良い例：

```
if(is_even_and_odd(a,b)){
    ...
}
```

条件が複雑でも関数単位で管理することができる、  
**意味のある関数名**が処理内容を物語っている。

## ◇ コーディング | 関数 悪い例

*main*関数に全ての処理が集中し、**全体の流れが汲み取りにくい**。  
解読に時間の掛かるプログラムである。

```
int main(){
    int i,a[30],sum=0;
    for(i=0; i<30;i++){
        scanf( "%d" ,&a[i]);
    }
    for(i=0; i<30;i++){
        sum += a[i];
    }
    for(i=0; i<30;i++){
        printf( "a[%d] = %d\n" ,i,a[i]);
    }
    printf( "sum = %d\n" ,sum);
    return 0;
}
```

## ◇ コーディング | 関数 良い例

処理の流れが、4ページのようなパターンに当てはまり、とても読みやすいプログラムになっており、*main*関数で管理する変数も減って解読の負担が減っている。

```
int main(){
    int col[30],sum=0;

    // 入力
    input(col);
    // 計算処理
    sum = calc_summation(col);
    // 出力
    print_collection(col);
    printf( "sum = %d\n" ,sum);

    return 0;
}
```

## ◇ コーディング | まとめ

- ▶ コード量が多少増えても、意味のある変数名・意味のある関数名に分割する
- ▶ *main*関数に書く処理は最小限に抑え、処理の流れが見えるようにする
- ▶ 意味のある処理をなるべく関数へ分割することで、変更に強いプログラムになる

## ◆ 課題

- ▶ 5ページの仕様から自分なりのイメージを書き出してみよう
- ▶ イメージを元にプログラムを書いてみよう
- ▶ コーディングの良い例・悪い例は**ベストプラクティス・バッドプラクティス**と呼ぶ。ベストプラクティスを *Github* から探してみよう！