iOS授業4日目資料

課題発表

発表が終わったら拍手しましょう👏

はじめに

はじめに：今日やること

◆1.TableViewやセルを作成する方法(復習)

◆2.Delegateを理解する

◆3.TableViewやDelegateに慣れる

TableViewやセルを作成する方法

UIViewController + UITableView vs UITableViewController

TableViewを表示する場合以下の２つの選択肢があります

	UIViewController + UITableView	UITableViewController
概要	VC内にTableViewを手動で配置	最初からVCとTableViewが一体化
レイアウト	TableViewに対して 自由に制約をつけてレイアウトできる	最初から画面一杯に表示されていて、 制約を変更することができない
初期作業 の違い	IBOutletでの紐付けや、 VCに対するdelegate/datasource設定 が必要	IBOutletでの紐付けや、 VCに対するdelegate/dataSource設定 は不要（最初からできている）
StaticCell	使用不可 *	使用可能

今回はこっち

* UIViewControllerにContainerViewを入れて、その中にUITableViewControllerをembedすることで使用可能

今回はUIViewControllerでの作成をやってみます

【一緒にやってみよう】

UIViewController + UITableView + Xibで
TableViewをミニマム実装しよう

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

作成の流れを抑えよう（全体の手順）

Cell側の作業

① TableViewCellの見た目を定義するためのXibと、
TableViewCellを制御するためのSwiftファイルを作成

② ①で作ったXibを使ってセルの見た目を設定して
配置したパーツをCellのSwiftファイルにOutletで接続

VC側の作業

③ ViewControllerにTableViewを配置してOutletで接続

④ ViewControllerで作ったセルを利用するコードや
TableViewを動かすためのコードを書いてあげる

作成の流れを抑えよう（①Cell用のファイルを作成）

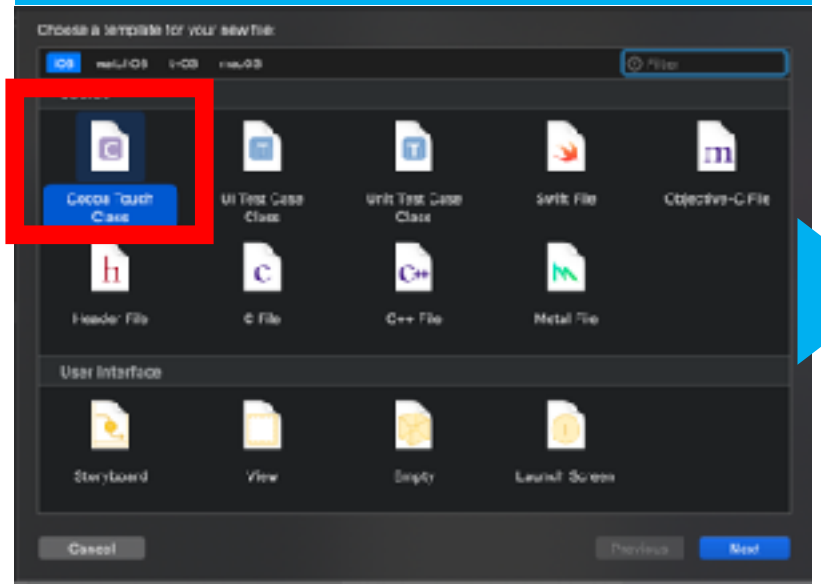
1-1. File > New > File > Cocoa Touch Class を選択

1-2.「Subclass of」欄で「UITableViewCell」を選択

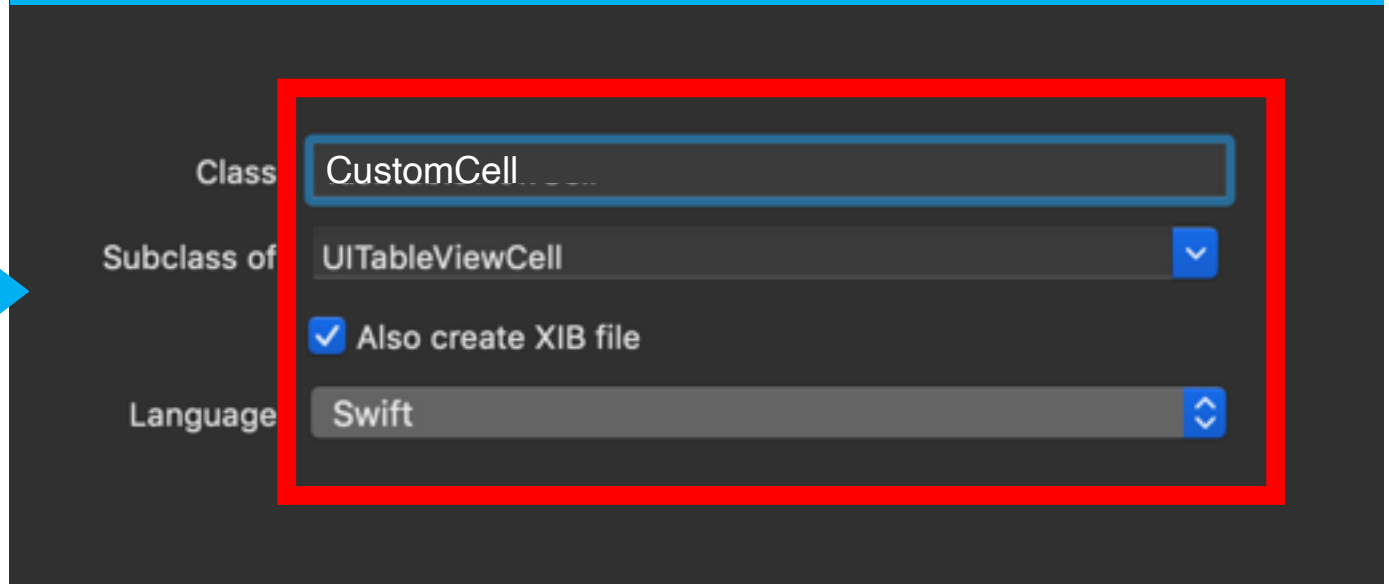
1-3.「Class」欄でクラス名を入力したら
「Also create XIB file」をチェックして「Next」

1-4. 保存場所を指定して「Create」

1-1. Cocoa Touch Classを選択



1-2～1-3. ファイル作成時の設定



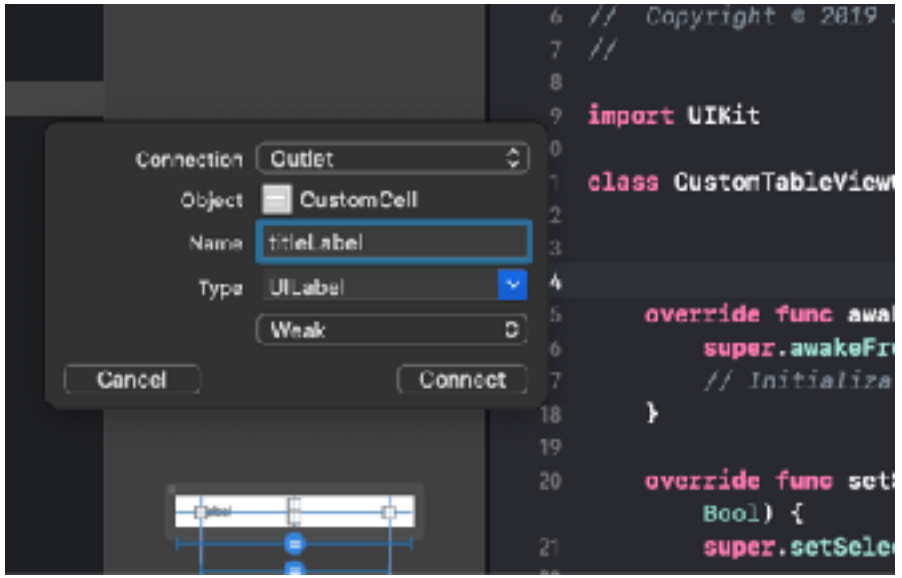
作成の流れを抑えよう（②Cellの見た目を設定）

2-1. CellのXibを開いてUIパーツを配置

2-2. CellのSwiftファイルにOutletで接続

2-1. Xibにラベルを配置

2-2. Swiftファイルに接続

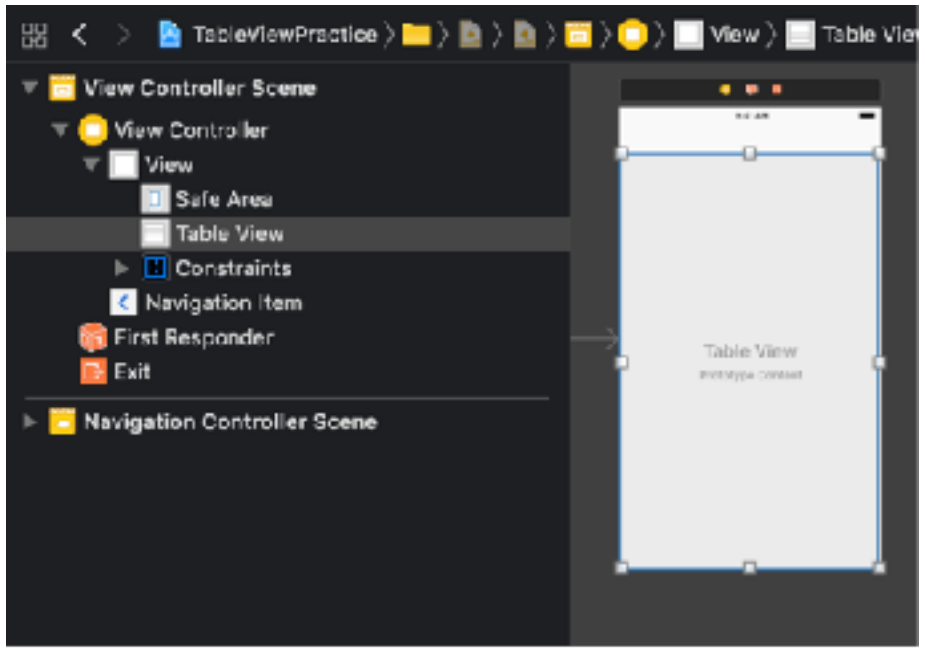


作成の流れを抑えよう（③ VCにTableViewを配置）

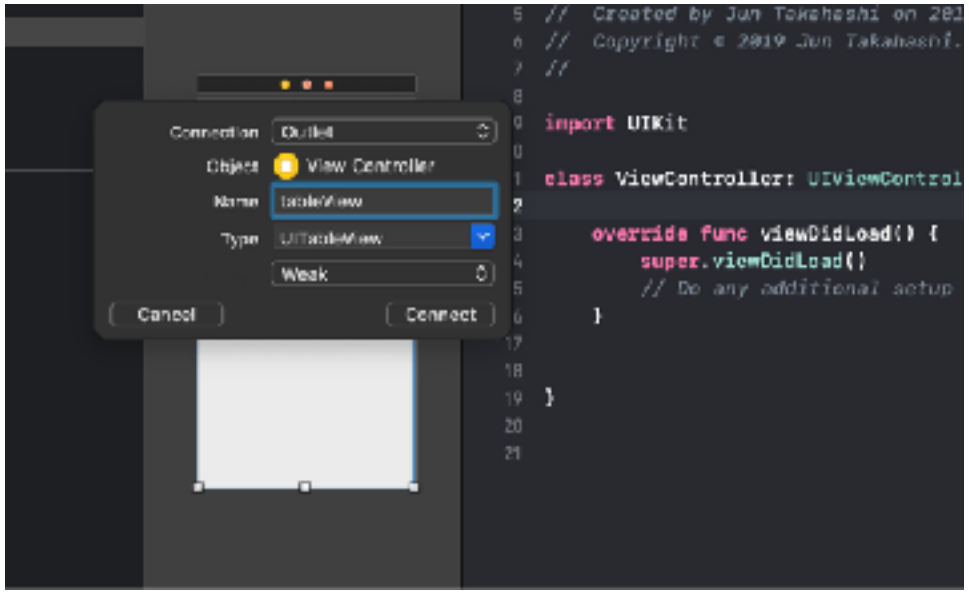
3-1. StoryBoard上でTableViewを配置

3-2. ViewControllerのSwiftファイルにOutletで接続

4-1. TableViewを配置



4-2. Swiftファイルに接続



作成の流れを抑えよう（④ ViewControllerの設定）

4-1. TableViewで作ったセルを利用できる状態にする

4-1. DidLoadで、TableViewに作ったXibの情報を登録する作業をする

```
//TableViewCellのクラス名を指定してNibを作成
let nib = UINib(nibName: "CustomCell", bundle: nil)

//TableViewCellにcellのIdentifierを指定して登録
tableView.register(nib,
                   forCellReuseIdentifier: cellId)
```

コンパイル後のXibがNibなので、Xib≒Nibと理解していいです

NibのNameとCellIDを入れ子にしないように注意😳

作成の流れを抑えよう（④ ViewControllerの設定）

4-2. TableViewの処理をVC側に委譲させる設定を行う

4-2. classの宣言のところでDelegateとDataSourceを追記して継承させる

```
class ViewController:
UIViewController,UITableViewDelegate,UITableViewDataSource {
```

4-2. DidLoadのタイミングでTableViewのdelegate/datasourceにVCを設定

```
//UIの処理の委譲
tableView.delegate = self
//データの処理の委譲
tableView.dataSource = self
```

Delegateは特に理解が難しい概念と言われています

あとで解説するので、ざっくり理解してみましょう😄

作成の流れを抑えよう（④ ViewControllerの設定）

4-3. 必須で設定しなければならない
DataSourceメソッド「numberOfRowsInSection」で
返すセルの個数を決めてあげる

4-3. numberOfRowsInSectionの設定

```
func tableView(_ tableView: UITableView,
                    numberOfRowsInSection section: Int) -> Int {
    //表示するセルの個数をIntで返す
    return taskList.count
}
```

この場合はVC内にある「taskList」配列の個数を
返してあげています

作成の流れを抑えよう（④ ViewControllerの設定）

4-4. 必須で設定しなければならない
DataSourceメソッド「cellForRowAt」で
返すセルの内容を決めてあげる

4-4. cellForRowAtメソッドで返すセルの内容を決める

```
func tableView(_ tableView: UITableView,
               cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    //再利用セルをキューから取り出す
    let cell = tableView.dequeueReusableCell(withIdentifier: "CustomCell",
                        for: indexPath) as! CustomTableViewCell
    //タイトルラベルを設定
    cell.titleLabel.text = taskList[indexPath.row]

    return cell
}
```

UITableViewCellのサブクラス「TaskTableViewCell」に
キャストしてあげないと作ったCellが使えないので注意

ミニマム実装完了🎉

ここまでで最低限の実装は完了です！

他に部品を追加した場合は…

2-1～2-2みたいな形でアイテムを追加

4-4の箇所で処理を追加してあげるだけ。

TableViewで他にやりたいことがある場合は…

4-3～4-4でやったような手順で

Delegate/Datasourceメソッドをどんどん追加していくだけです！

できることは公式リファレンス や 以下記事を参考に

【参考】UITableViewのデリゲートメソッドまとめ

https://qiita.com/kagemiku/items/22b74010365723c5c4fe

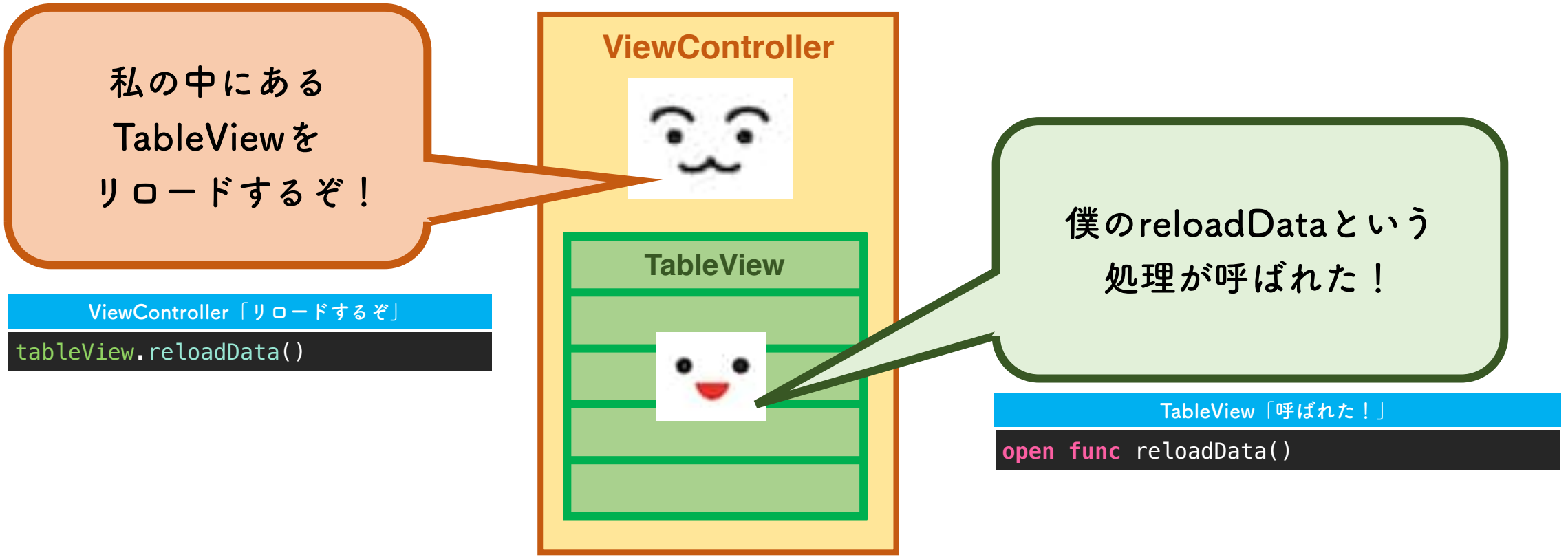
Delegateを理解する

Delegateをイメージで理解する①

ViewControllerはTableViewをプロパティとして持っている

```
@IBOutlet weak var tableView: UITableView!
```

だから直接TableViewの値を変更したり処理を呼ぶことができる



ViewController「リロードするぞ」

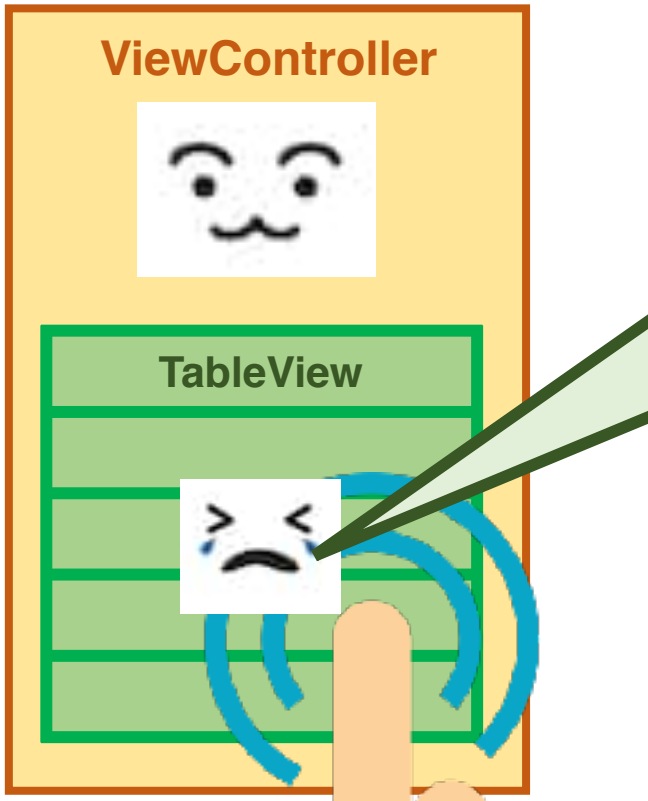
```
tableView.reloadData()
```

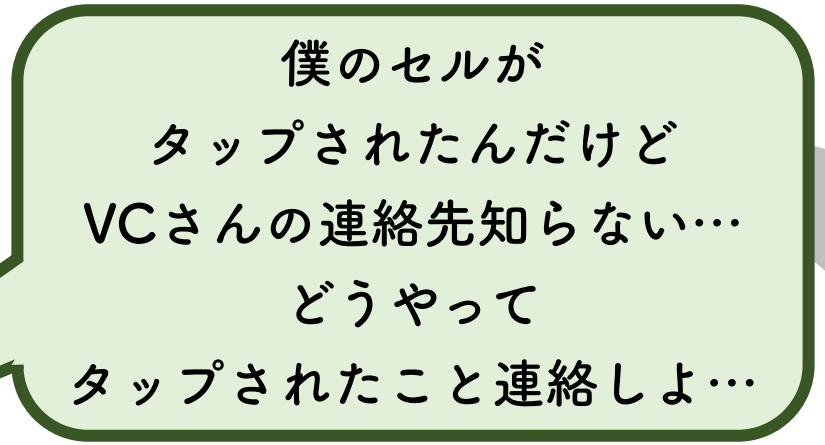
TableView「呼ばれた！」

```
open func reloadData()
```

Delegateをイメージで理解する②

TableViewはViewControllerをプロパティとして持っていない
（連絡先が分からない）ので、何もしないと連絡ができない

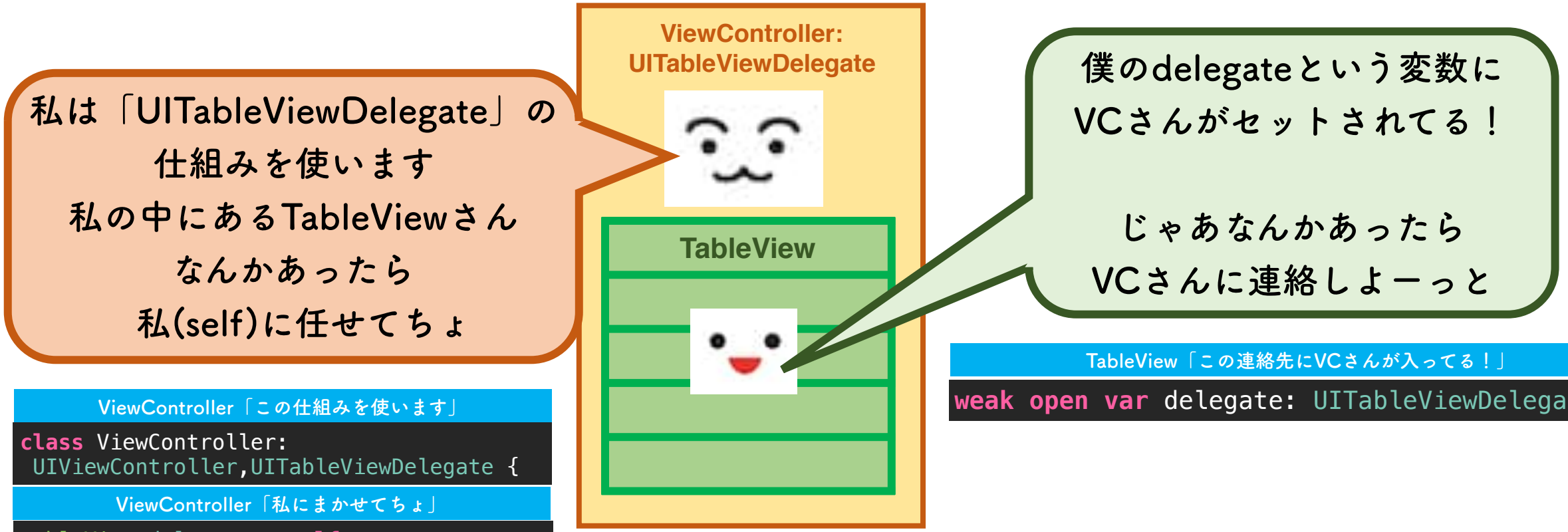




だから「Delegate」という仕組みを使って
連絡できるようにする必要がある

Delegateをイメージで理解する③

TableViewは「Delegate」の仕組みを使うことで
ViewControllerの連絡先を把握し、通知できるようになる



ViewController「この仕組みを使います」

```
class ViewController:
 UIViewController,UITableViewDelegate {
```

ViewController「私にまかせてちょ」

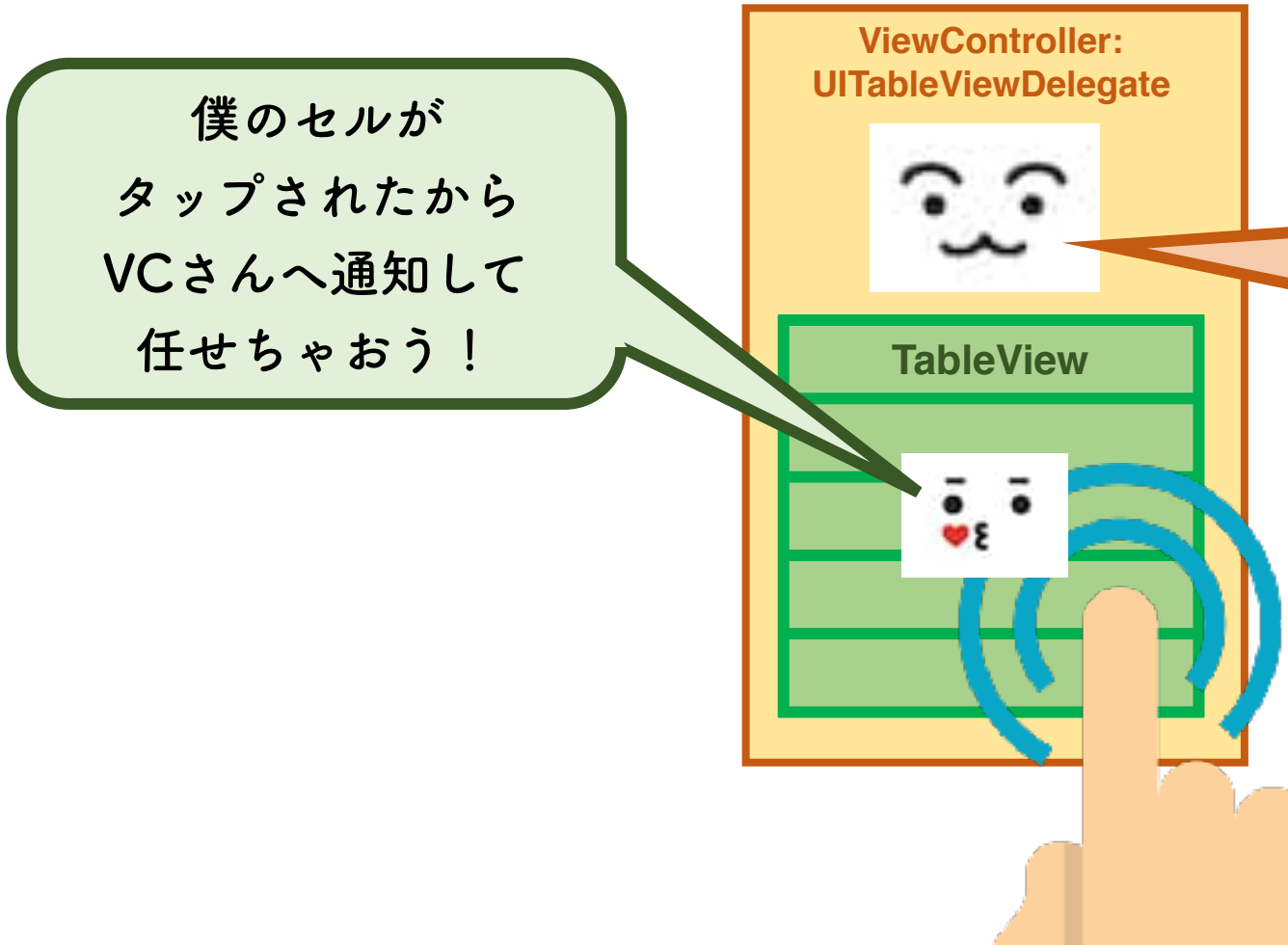
```
tableView.delegate = self
```

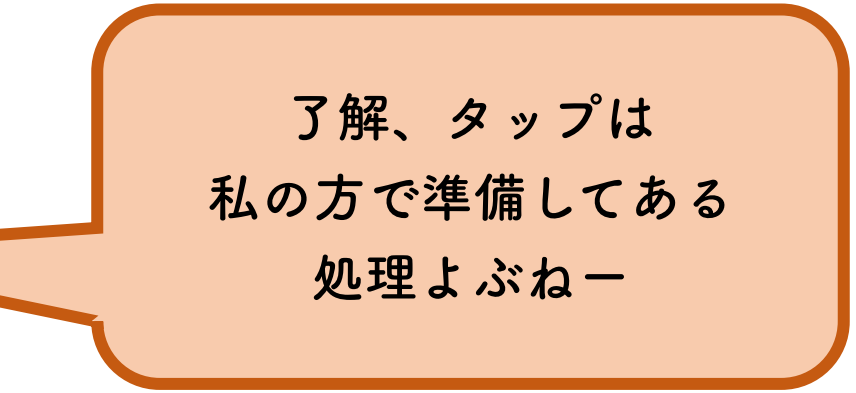
TableView「この連絡先にVCさんが入ってる！」

```
weak open var delegate: UITableViewDelegate?
```

Delegateをイメージで理解する④

TableViewから通知を受けて処理を任された
ViewControllerは通知された内容に合ったメソッドを呼び出す

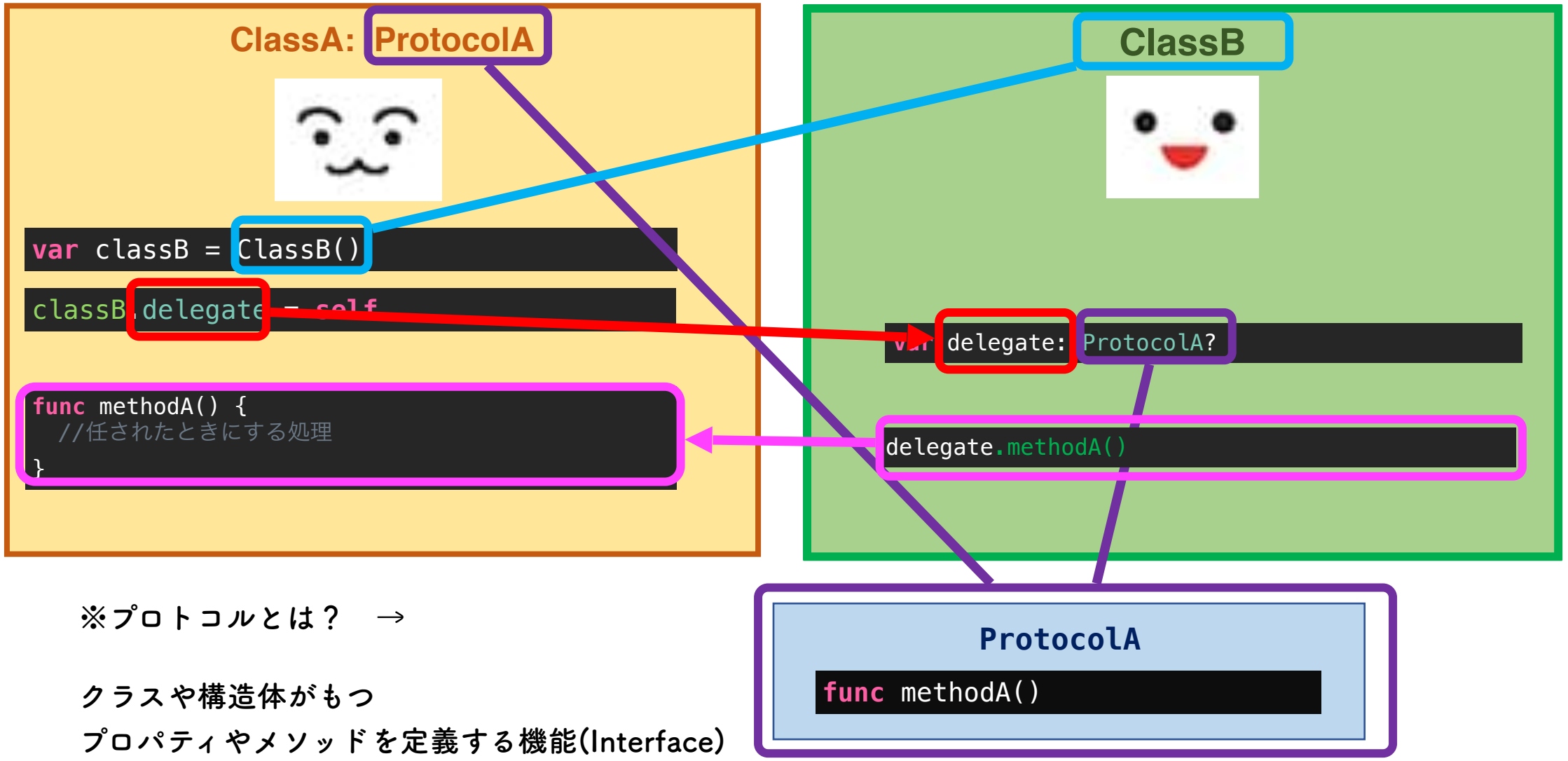




ViewController「私の方で準備してある処理呼ぶねー」

```
func tableView(_ tableView: UITableView,
               didSelectRowAt indexPath: IndexPath) {
  //タップしたときになにかする
}
```

補足：Delegateの全体像を把握する



※プロトコルとは？　→

クラスや構造体がもつ
プロパティやメソッドを定義する機能(Interface)

自分自身でDelegateを作る場合にはこの流れを理解しておく

ここまでのおさらい

こんなことを学びました

①TableView表示方法の選択肢

②Cellの見た目の作成方法の選択肢

③TableViewの作成手順

④Delegateの仕組み

ToDoアプリを通して
TableViewやDelegateに慣れる

【一緒にやってみよう】

登録機能・Delegateを作ってみよう

※完成PJは配布するので
ついてこれなくなったら
手を止めて見ることに集中！

今回やりたいこと

① 「Task」Classを作る

② Task配列を管理する「TaskCollection」Classを作る

③ ソースを「TaskCollection」に置き換える

④ AddVCとTaskCollectionを変更して追加処理を作成する

⑤ Delegateでリロード処理を作ってみる

①「Task」Classを作る

```
var taskList = ["aaaa","bbbb","cccc","dddd"]
```

今のやり方だと、Task自体が持つ情報が増えたり、

色々な場所でTaskを扱うことになったりしたときの影響が大きい💧

Taskの設計書みたいなもの(Class)を別で用意して、

Taskを扱う場合は、その設計書みたいなものを使用するようにする💡

```
class Task {
    var title: String
    var memo: String?

    init(title: String) {
        self.title = title
    }
}
```

② Task配列を管理する「TaskCollection」Classを作る

今回のアプリではこの配列データの操作がメインとなるので
別クラスに切り出しておく

また、外部からデータを操作できるように準備する

※ここはインスタンスを1つだけ生成できる書き方で書いています
　細部に興味があれば「シングルトン」などで調べてみてください

```
class TaskCollection {
        //初回アクセスのタイミングでインスタンスを生成(シングルトン)
        static var shared = TaskCollection()
        //外部からの初期化を禁止
        private init(){}
        //外部からは参照のみ許可
        private(set) var tasks: [Task] = []
}
```

③ソースを「TaskCollection」に置き換える

Tableの参照先をtaskListからtaskCollectionに置き換える

3-1.データソース自体を変更

```
var taskList = ["aaaa","bbbb","cccc","dddd"]
```

```
let taskCollection = TaskCollection.shared
```

3-2. numberOfRowsInSectionを修正

```
return taskList.count
```

```
return taskCollection.tasks.count
```

3-3. cellForRowAtを修正

```
cell.textLabel?.text = taskList[indexPath.row]
```

```
cell.textLabel?.text = taskCollection.tasks[indexPath.row].title
```

④ AddVCとTaskCollectionを変更して追加処理を作成する

AddVCでSaveボタンを押したときにTaskを追加する処理を実装

4-1.TaskCollectionに処理を追加

```
func addTask(_ task: Task) {
  tasks.append(task)
}
```

4-2.AddViewControllerに処理を追加

```
let task = Task(title: title)
task.memo = memoTextView.text
taskCollection.addTask(task)
```

これでCollectionへの追加自体はできましたが
TableViewが更新されませんね😂

色々方法はありますが、
今回はDelegateを使って更新を走らせてみましょう

先ほど行ったDelegateの例を
以下のように置き換えて考えましょう

＜伝える先＞
ViewController -> TaskTableViewController

＜伝える元＞
TableView -> TaskCollection

TaskCollectionさんの話を聞いてみると…

「なんか僕にTask追加されたんだけど、
TaskTableViewController さんにどうやって伝えよう…」

なんか見たことありますね😳

Delegateで解決できるパターンです💡

⑤ Delegateでリロード処理を作ってみる

P17の仕組みを自分で作ってみる

5-1.プロトコルを定義（他クラスに移譲する処理を定義）

```
protocol TaskCollectionDelegate: class {
    func reload()
}
```

5-2.TaskCollectionに連絡先を入れる変数を作る

```
weak var delegate: TaskCollectionDelegate? = nil
```

5-3. TaskTableViewControllerでこのDelegateの仕組み使うよーを定義

```
class TaskListTableViewController: UITableViewController, TaskCollectionDelegate {
```

⑤ Delegateでリロード処理を作ってみる

5-4.TaskTableViewControllerで連絡先を教えてあげる処理を作成

```
taskCollection.delegate = self
```

5-5.TaskCollectionでリロードさせる処理を追加

```
func addTask(_ task: Task) {
  tasks.append(task)
  save() //←ここを追加
}

func save(){   //新しく追加する処理 Delegateでフックを作成
  delegate?.reload()
}
```

【本日はまりそうなところを簡単に羅列した記事】

TableView と delegate を理解する（したい）

https://qiita.com/yamatatsu10969/items/0858b145a2a93a92af44

【次回までの課題】

ToDoアプリの編集・削除機能を作る

「そんなの余裕だよ」って人は
追加でUserDefaultsでTaskCollectionを保存する

それ以上の挑戦はなんでもOK

【次回の予告】

ToDoアプリにライブラリの機能を追加する

（ライブラリとは他の人が作った
便利な機能のようなものです！）

【準備すること】

CocoaPods を入れてくる！

以降のスライドに書いてあります！

CocoaPods（ライブラリ管理ツール）
導入

【CocoaPods導入】

以下の記事などを参考にしてインストールしてください

https://qiita.com/ShinokiRyosei/items/3090290cb72434852460

「pod setup」コマンドが完了したところまででOKです
（「ライブラリの導入」以降はやらなくてOK）

コマンドライン自体の基礎操作は知っておくと今後便利です
以下のコースを学習しておくことをオススメします（約1時間）

https://prog-8.com/languages/commandline