複数の戻り値あり

複数の戻り値を返すこともできます。

戻り値の型のところを()をつけて複数の型を用意します。

```
func 関数名(引数:引数の型,...) -> (戻り値の型, 戻り値の型,...) {
        // 行う仕事
        return 戻り値
}
```

例

```
//-- 関数の書き方：複数の戻り値あり
func myFunc() -> (Int, String, Bool) {
    println("複数の戻り値を返す")
    return (100, "Hello", true)
}
//-- 関数の呼び出し方
var ans = myFunc()                    // 複数の値が返ってきます
ans.0   // 0番目の戻り値
ans.1   // 1番目の戻り値
ans.2   // 2番目の戻り値
```

```
"複数の戻り値を返す"
(.0 100, .1 "Hello", .2 true)

(.0 100, .1 "Hello", .2 true)
100
"Hello"
true
```

Tips

メソッドの引き数が2つ以上ある場合 Xcode 6.1

少し条件が複雑ですが、「メソッドの引き数の数が2つ以上あって、ラベルを使っていない場合」には、そのメソッドを呼び出すとき、ラベル名も指定する必要が出てきます。

1つ目の引き数にはラベル指定をせず、そのまま値を書くのですが、2つ目以降の引き数には変数名をラベルとして指定します。

例 引き数が3つあって、ラベルを使っていない場合

```
func myFunc(val1:Int, val2:Int, val3:Int)->Int {
        return val1 * val2 * val3
}
```

2つ目以降の引き数には変数名をラベルとして指定します。

```
var ans = myFunc(1, val2: 2, val3: 3)
```

もしラベルを書かなかったとしても、Xcodeが⭕付きの警告で「この書き方は足りないところがありますよ」と教えてくれます。さらに、その⭕をクリックしていくと、Xcodeが正しい書き方に修正してくれます。

はてな？

関数ってどんな数？

「関数」には「数」とついていますが、計算結果を出さずに、仕事をするだけの場合もあります。昔のコンピュータは、仕事としては計算結果を出すことだったので、そのなごりです。

はてな？

メソッドと関数

クラスのメソッドも関数の一種です。

ほとんど同じですが、メソッドはクラスが持っている関数です。クラスの外から見たとき、そのクラスにできる仕事として見えるものがメソッドです。

使い方 Web上の画像を表示させる方法

❶ URLオブジェクトを作る：NSURL(string: String)

URLを指す文字列から、URLオブジェクト(NSURL)を作ります。

```
var URLオブジェクト = NSURL(string: "URLの文字列")
```

例 URLを指す文字列からURLオブジェクトを作る

```
var myURL = NSURL(string: "http://www.ymori.com/itest/test.jpg")
```

❷ ファイルデータを作る：NSData(contentsOfURL: NSURL) Xcode 6.1

次に、指定したURLオブジェクトからデータを読み込んで、素のファイルデータ(NSData)を作ります。

```
var ファイルデータ = NSData(contentsOfURL: URLオブジェクト!)
```

例 URLオブジェクトからファイルデータを作る

```
var myData = NSData(contentsOfURL: myURL!)
```

❸ イメージデータを作る：UIImage(data: NSData!) Xcode 6.1

素のファイルデータのままでは画像にならないので、ここからイメージデータ(UIImage)に変換します。

```
var イメージデータ = UIImage(data: ファイルデータ!)
```

例 ファイルデータからイメージデータを作る

```
var myImage = UIImage(data:myData!)
```

❹ イメージビューに表示する：var image: UIImageView

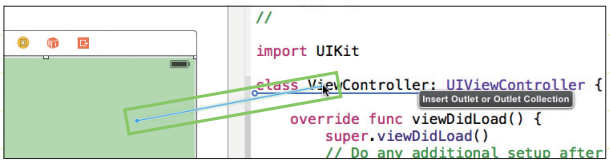
できたイメージデータを、イメージビューに設定します。すると、Web画像が表示されます。

```
イメージビュー.image = イメージデータ
```

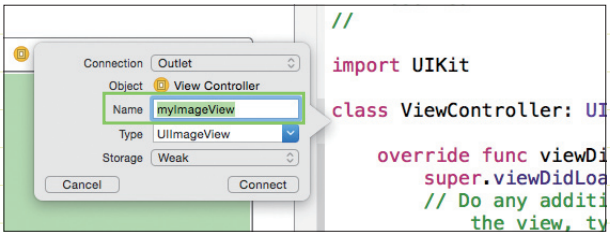
例 イメージデータをイメージビューに設定する

```
myImageView.image = myImage
```

Image Viewをプログラムにつないで
IBOutletの名前をつけます。

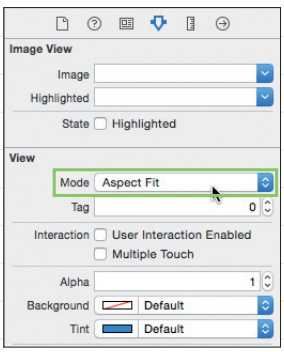


Name：IBOutlet名（例：myImageView）



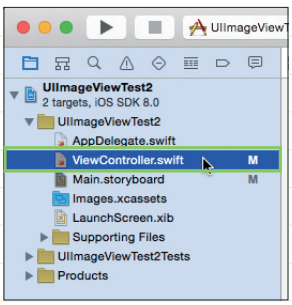
❸ 画像の比率を設定する

アトリビュート・インスペクタの「View＞Mode」で「Aspect Fit」を選択します。



❹ プログラムを作る

「ViewController.swift」を選択します。



プログラムを書きます。 Xcode 6.1

```
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        var myURL = NSURL(string: "http://www.ymori.com/itest/sample.jpg")
        var myData = NSData(contentsOfURL: myURL!)
        var myImage = UIImage(data:myData!)
        myImageView.image = myImage
    }
```

使い方

ウェブビューの使い方

Webページを表示する方法

❶ UIWebViewにオブジェクト名を設定

アシスタントエディターで、UIWebViewをプログラムにつないでOutletの名前をつけます。

❷ URLオブジェクトを作る：NSURL( )

URLを指す文字列から、URLオブジェクト(NSURL)を作ります。

```
var URLオブジェクト = NSURL(string: "URLの文字列")
```

例 URLを指す文字列からURLオブジェクトを作る

```
var myURL = NSURL(string: "http://www.apple.com/jp")
```

❸ URLリクエストを作る：NSURLRequest( ) Xcode 6.1

次にURLオブジェクトから、URLリクエスト(NSURLRequest)を作ります。

```
var URLリクエスト = NSURLRequest(URL: URLオブジェクト!)
```

例 URLオブジェクトから、URLリクエストを作る

```
var myURLReq = NSURLRequest(URL: myURL!)
```

❹ ウェブビューに表示：func loadRequest(NSURLRequest!)

作ったURLリクエストを使って、ウェブビューのloadRequest:メソッドを実行すると、指定したページが表示されます。

```
ウェブビュー.loadRequest(URLリクエスト)
```

例 URLリクエストを使って、ウェブビューで実行する

```
myWebView.loadRequest(myURLReq)
```

ローディング中にインジケータを表示する方法

❶ UIWebViewにオブジェクト名を設定

アシスタントエディターで、UIWebViewをプログラムにつないでOutletの名前をつけます。

❷ プロトコルを追加する：UIWebViewDelegate

ViewController.swiftに、プロトコル(UIWebViewDelegate)を追加して、UIWebViewを使う準備をします。

```
class ViewController: UIViewController, UIWebViewDelegate {
```

❸ デリゲート先を設定する：var delegate: UIWebViewDelegate!

デリゲート先をself(ViewController)に設定することで、ウェブビューの状態によって、ViewController内のwebViewDidStartLoadメソッドや、webViewDidFinishLoadメソッドが呼ばれるようになります。
ViewController.swiftに書きます。

```
ウェブビュー.delegate = UIWevViewの状態が変わったときの通知先
```

例

```
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        myWebView.delegate = self
    }
```

❹ URLリクエストを作って、ウェブビューに表示する　Xcode 6.1

URLオブジェクト(NSURL)からURLリクエスト(NSURLRequest)を作り、ウェブビューのloadRequestメソッドを実行します。

例

```
        var myURL = NSURL(string: "http://www.apple.com/jp")
        var myURLReq = NSURLRequest(URL: myURL!)
        myWebView.loadRequest(myURLReq)
```

❺ ページ読込開始時の処理：func webViewDidStartLoad(webView: UIWebView!)

ページ読込開始時に、インジケータを表示します。

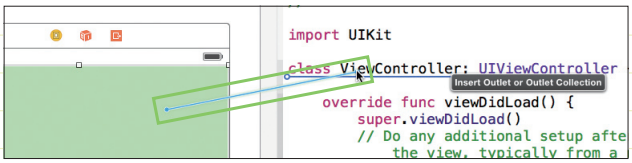
```
    func webViewDidStartLoad(webView: UIWebView!) {
        UIApplication.sharedApplication().networkActivityIndicatorVisible = true
    }
```

❻ ページ読込完了時の処理：func webViewDidFinishLoad(webView: UIWebView!)

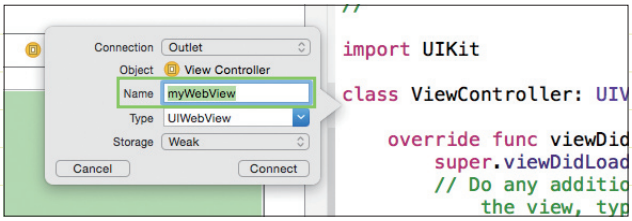
ページ読込完了時に、インジケータを停止します。

```
    func webViewDidFinishLoad(webView: UIWebView!) {
        UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    }
```

WebViewをプログラムにつないでIBOutletの名前をつけます。

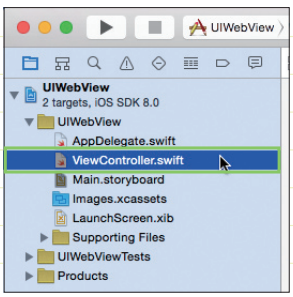


Name：IBOutlet名（例：myWebView）



❹ プログラムを作る　Xcode 6.1

「ViewController.swift」を選択します。ViewControllerに、プロトコル（UIWebViewDelegate）を追加します。



WebView表示のプログラムを書きます。読み込み開始時にステータスバーにインジケータを表示、終了時にインジケータを非表示にします。

```
import UIKit

class ViewController: UIViewController, UIWebViewDelegate {
    @IBOutlet weak var myWebView: UIWebView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // デリゲート先を設定する
        myWebView.delegate = self
        // URLリクエストを作って
        var myURL = NSURL(string: "http://www.apple.com/jp")
        var myURLReq = NSURLRequest(URL: myURL!)
        // ウェブビューに表示する
        myWebView.loadRequest(myURLReq)
    }
    // ページ読込開始時に、インジケータを表示する
    func webViewDidStartLoad(webView: UIWebView!) {
        UIApplication.sharedApplication().networkActivityIndicatorVisible = true
    }
    // ページ読込完了時に、インジケータを非表示にする
    func webViewDidFinishLoad(webView: UIWebView!) {
        UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

使い方

JSONデータの読み込み方

1 ネット上のJSONデータを読み込む方法

❶ JSONのURLからデータを読み込む

ネット上のJSONデータのURLからNSURLRequestを作り、NSURLConnectionのsendSynchronousRequest:メソッドで、データを読み込みます。

```
var url = NSURL(string: "JSONファイルのURLt")
        var request = NSURLRequest(URL: url)
        var jsondata = NSURLConnection.sendSynchronousRequest(request, returningResponse:
nil, error: nil)
```

❷ 読み込んだデータを変換して、配列や辞書データに変換する Xcode 6.1

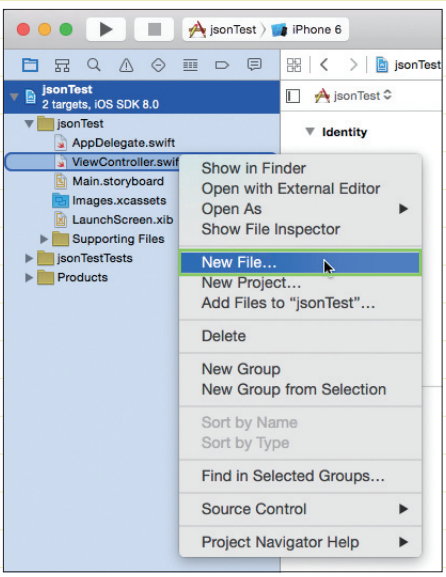
```
let jsonArray = NSJSONSerialization.JSONObjectWithData(jsondata!, options: nil, error:
  nil) as NSArray
```

2 プロジェクトの中のJSONデータを読み込む方法

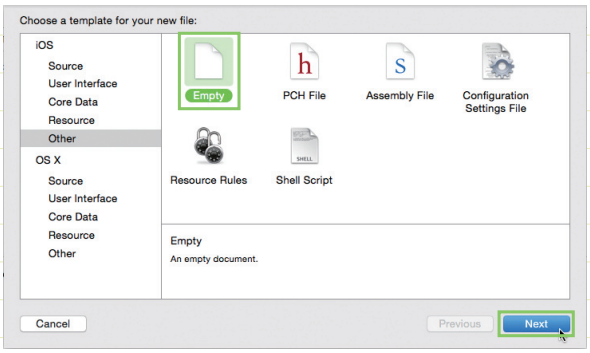
ネットを使わず、プロジェクトの中にJSONデータを作って読み込ませることもできます。

❶ プロジェクト内にJSONファイルを作る

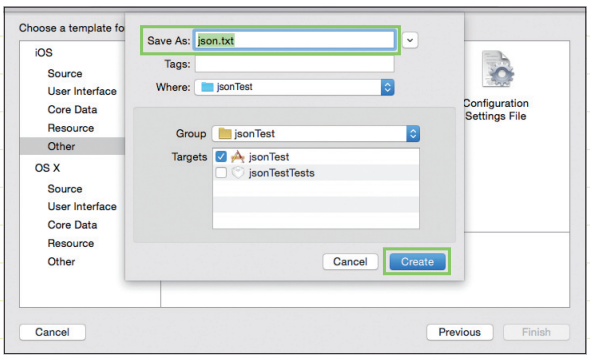
ナビゲータエリアを右クリック(control＋クリック)して、「New File...」を選択します。



Other>Emptyを選択して「Next」ボタンを選択します。



テキストファイル名をつけて「Create」ボタンを選択します(例：json.txt)。



ナビゲータエリアでテキストファイルを選択して、JSONデータを記述します。

❷ プロジェクト内のJSONデータを読み込むプログラムを作る

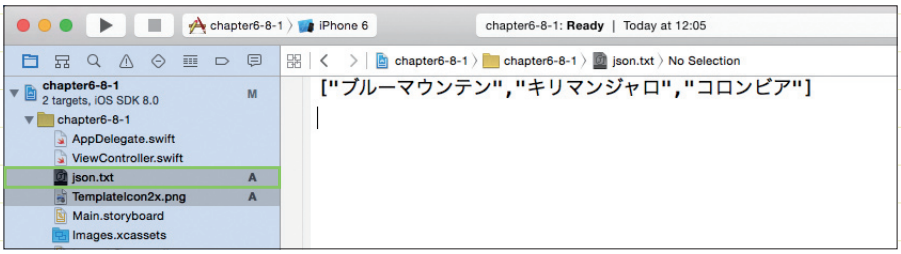
パスを作って、NSDataとして取り出します。

```
var path = NSBundle.mainBundle().pathForResource("ファイル名", ofType: "拡張子")
var jsondata = NSData(contentsOfFile: path)
```

❸ データを変換して、配列や辞書データに変換します。 Xcode 6.1

```
let jsonArray = NSJSONSerialization.JSONObjectWithData(jsondata!, options: nil, error: nil)
as NSArray
```

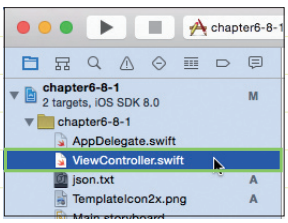
ナビゲータエリアでテキストファイルを選択して、JSONデータを記述します。




```
01 ["ブルーマウンテン","キリマンジャロ","コロンビア"]
```

② プログラムを作る

「ViewController.swift」を選択します。

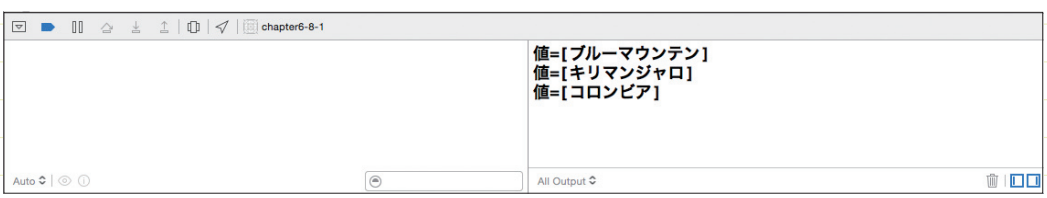


画面を表示したときに、JSONデータを読み込んで変換し、表示します。 Xcode 6.1

```
    // 画面が表示されるときに
    override func viewWillAppear(animated: Bool) {
        //-- json.txtファイルを読み込んで
        var path = NSBundle.mainBundle().pathForResource("json", ofType: "txt")
        var jsondata = NSData(contentsOfFile: path!)
        //-- 配列データに変換して
        let jsonArray = NSJSONSerialization.JSONObjectWithData(
            jsondata!, options: nil, error: nil) as NSArray
        //--  配列の個数だけ繰り返して表示する
        for dat in jsonArray {
            println("値=[\(dat)]")
        }
    }
```

※この例では、実行するとデバッグエリアに値を表示します。

結果

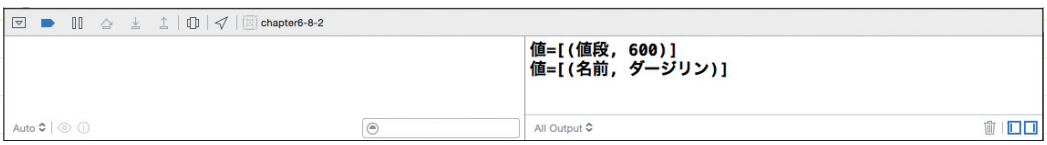


画面を表示したときに、JSONデータを読み込んで変換し、表示します。 Xcode 6.1

```
    // 画面が表示されるときに
    override func viewWillAppear(animated: Bool) {
        // json.txtファイルを読み込んで
        var path = NSBundle.mainBundle().pathForResource("json", ofType: "txt")
        var jsondata = NSData(contentsOfFile: path!)
        // 配列データに変換して
        let jsonDictionary = NSJSONSerialization.JSONObjectWithData(
            jsondata!, options: nil, error: nil) as NSDictionary
        // 配列の個数だけ繰り返して表示する
        for dat in jsonDictionary {
            println("値=[\(dat)]")
        }
    }
```

※この例では、実行するとデバッグエリアに値を表示します。

結果

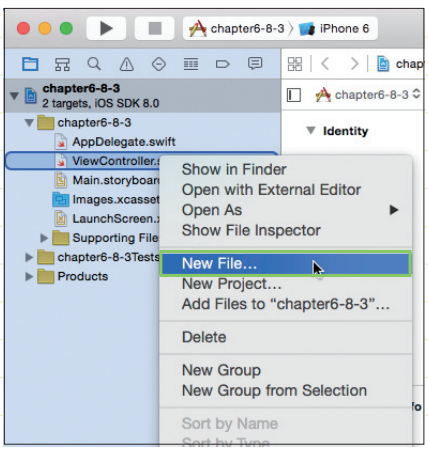


やってみよう 「辞書の配列」の読み込み

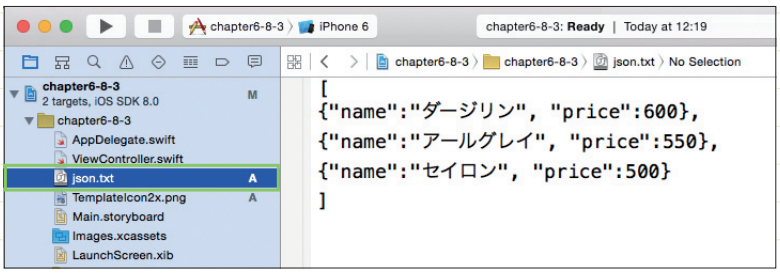
プロジェクト内のJSONデータ(辞書の配列)の値を、デバッグエリアに表示させてみましょう。
「辞書をデータを配列にしたデータ」は、まず配列データに変換し、各要素をそれぞれ辞書データとしてアクセスします。

❶ プロジェクト内に、辞書の配列データのJSONデータファイルを作る

・ナビゲータエリアを右クリック(control＋クリック)して、「New File...」を選択します。
・Other>Emptyを選択して「Next」ボタンを選択します。
・テキストファイル名をつけて「Create」ボタンを選択します(例：json.txt)。

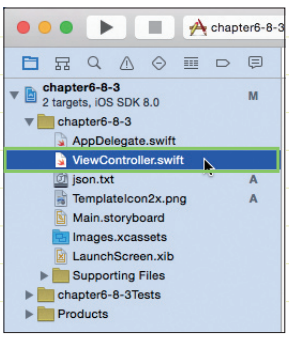


ナビゲータエリアでテキストファイルを選択して、JSONデータを記述します。



❷ プログラムを作る

「ViewController.swift」を選択します。

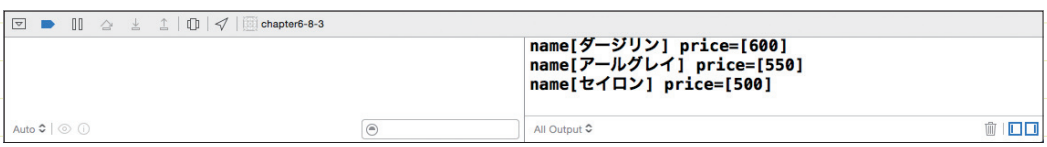


画面を表示したときに、JSONデータを読み込んで変換し、表示します。 Xcode 6.1

```
    override func viewWillAppear(animated: Bool) {
        // json.txtファイルを読み込んで
        var path = NSBundle.mainBundle().pathForResource("json", ofType: "txt")
        var jsondata = NSData(contentsOfFile: path!)
        // 辞書データに変換して
        let jsonArray = NSJSONSerialization.JSONObjectWithData(
            jsondata!, options: nil, error: nil) as NSArray
        // 配列の個数だけ繰り返して表示する
        for dat in jsonArray {
            var d1 = dat["name"] as String
            var d2 = dat["price"] as Int
            println("name[\(d1)] price=[\(d2)]")
        }
    }
```

※この例では、実行するとデバッグエリアに値を表示します。

結果

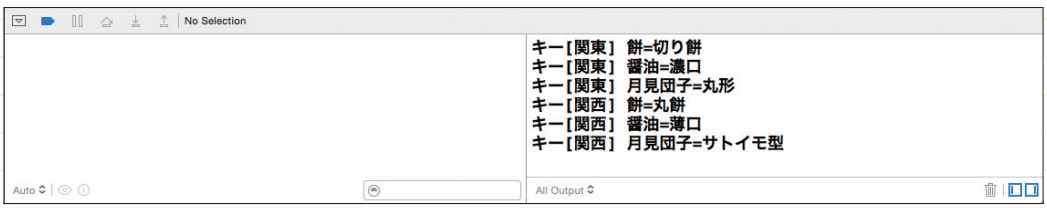


画面を表示したときに、JSONデータを読み込んで変換し、表示します。 Xcode 6.1

```
// 画面が表示されるときに
override func viewWillAppear(animated: Bool) {
    // json.txtファイルを読み込んで
    var path = NSBundle.mainBundle().pathForResource("json", ofType: "txt")
    var jsondata = NSData(contentsOfFile: path!)
    // 辞書データに変換して
    let jsonDictionary = NSJSONSerialization.JSONObjectWithData(
        jsondata!, options: nil, error: nil) as NSDictionary
    // 辞書データの個数だけ繰り返して表示する
    for (key, data) in jsonDictionary {
        var d1 = data["餅"] as String
        var d2 = data["醤油"]  as String
        var d3 = data["月見団子"] as String
        println("キー[\(key)] 餅=\(d1)")
        println("キー[\(key)] 醤油=\(d2)")
        println("キー[\(key)] 月見団子=\(d3)")
    }
}
```

※この例では、実行するとデバッグエリアに値を表示します。

結果



❷VViewController.swiftに、２つのプロトコル(UITableViewDataSource、UITableViewDelegate)を追加して、テーブルビューを使う準備をします。

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

❸テーブルビューの表示をメソッドでコントロールします。

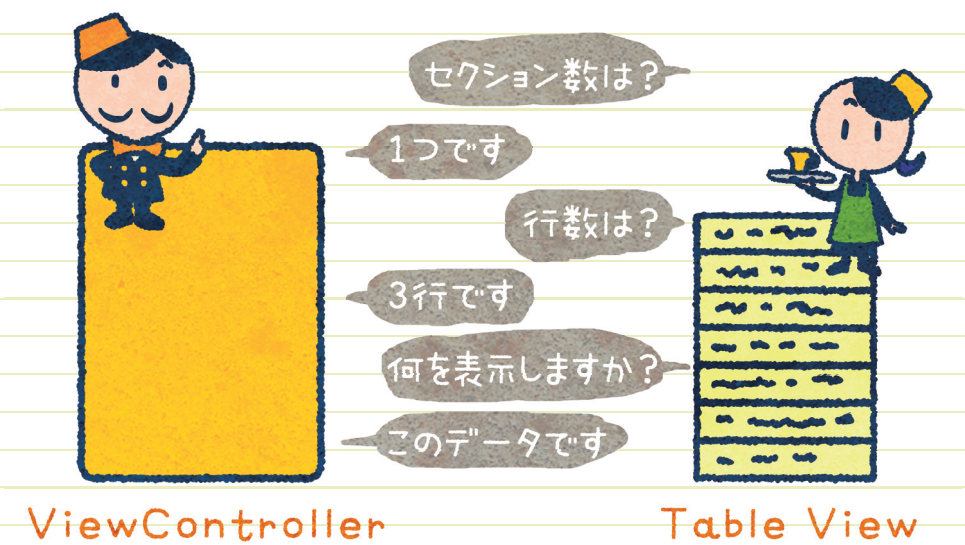
表示に必ず必要なメソッド

行数

```
func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 行数
}
```

セルの内容 Xcode 6.1

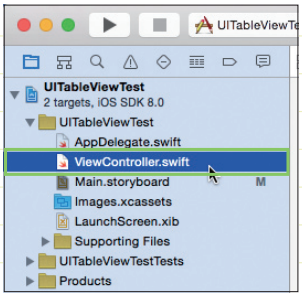
```
func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
UITableViewCell {
        var cell = UITableViewCell(style: .Default, reuseIdentifier: "myCell")
        cell.textLabel.text = "文字列"
        return cell
}
```



❷ プログラムを作る

「ViewController.swift」を選択します。

テーブルビューを使う準備をします。



テーブルビューを表示するプログラムを修正します。 Xcode 6.1

```
import UIKit

class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    // 行数
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 10
    }
    // 表示するセルの中身
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
        UITableViewCell {
        var cell = UITableViewCell(style: .Default, reuseIdentifier: "myCell")
        cell.textLabel.text = "\(indexPath.row)行目"

        return cell
    }
    // 選択されたときに行う処理
    func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath: NSIndexPath!) {
        println("\(indexPath.row)行目を選択")
    }
    // ステータスバーを非表示にする
    override func prefersStatusBarHidden() -> Bool {
        return true
    }

}
```

cellForRowAtIndexPathで、表示する行のラベルの値を設定します。 Xcode 6.1

```
    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NS
        UITableViewCell {
        let cell = tableView.dequeueReusableCellWithIdentifier("Cell", forIndexPath: in
            UITableViewCell

        //  表示する行のラベルに、配列の値を設定する
        var myStr = objects[indexPath.row] as String
        cell.textLabel.text = myStr

        return cell
    }
```

❹ 行が選択されたときのプログラムを作る

行が選択されたときは、prepareForSegueメソッドが呼ばれてから、画面が切り替わります。

prepareForSegueにデータをディテール画面へ伝えるプログラムを書きます。

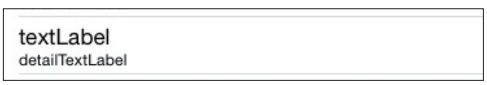
```
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        if segue.identifier == "showDetail" {
            // 配列の選択した行を調べてその値を受け渡す
            if let indexPath = self.tableView.indexPathForSelectedRow() {
                let object = objects[indexPath.row] as String
                let controller = segue.destinationViewController as DetailViewController
                controller.detailItem = object
            }
        }
    }
```

デフォルトでは、destinationViewController（ディテール画面）に、setDetailItem:メソッドで日付データ（NSDate）を渡しています。detailItemは、AnyObject型と言ってオブジェクトであればどんな型のデータでも扱えるので、文字列を渡すように修正しました。

ディテール画面（DetailViewController.swift）を見ると、受け取ったデータをラベルに表示させるしくみができているので、そのまま表示されます。

```
    func configureView() {
        // Update the user interface for the detail item.
        if let detail: AnyObject = self.detailItem {
            if let label = self.detailDescriptionLabel {
                label.text = detail.description
            }
        }
    }
```

```
var cell = UITableViewCell(style: セルの種類, reuseIdentifier: セルID)
```

UITableViewCellStyle.Default または、 .Default	 textLabelが左に。detailTextLabelは非表示
UITableViewCellStyle.Value1 または、 .Value1	 textLabelが左に。detailTextLabelが右に薄く
UITableViewCellStyle.Value2 または、 .Value2	 textLabelが左に青く。detailTextLabelが中央に
UITableViewCellStyle.Subtitle または、 .Subtitle	 textLabelが上に。detailTextLabelが下に

例 サブタイトルの表示にする

```
var cell = UITableViewCell(style: UITableViewCellStyle.Subtitle,
    reuseIdentifier: "myCell")
```

セルの高さを設定する：var rowHeight: CGFloat

セルの高さを設定するときは、セルではなく、セルを表示しているテーブルビューの方の、rowHeightプロパティに高さを数値で設定します。

```
テーブルビュー.rowHeight = 高さ
```

例 セルの高さを100にする

```
tableView.rowHeight = 100
```

文字内容を設定する：var textLabel, detailTextLabel: UILabel! Xcode 6.1

通常のテキストを設定するときは、テキストラベル(textLabel)の、textプロパティを設定します。
サブテキストを設定するときは、詳細テキストラベル(detailTextLabel?)の、textプロパティを設定します。

```
セル.textLabel.text = "文字列"
```

```
セル.detailTextLabel?.text = "文字列"
```

textLabelと違い、detailTextLabelの後に？がついているのは、セルの種類によってはdetailTextLabelがなくてnilの場合もありうるからです。簡単に言うならこの「?」は、「セルの種類によってはこの部品はないかもね？」という意味です。
（Xcode 6.0ではセル内の全ての部品に「?」をつける仕様でしたが、6.1でdetailTextLabelだけにつける仕様に変わりました。今後は仕様変更がないことを期待します。）

例 詳細テキストに「サブテキスト」と表示する

```
        cell.detailTextLabel?.text = "サブテキスト"
```

セルに画像を設定する：var imageView.image: UIImage! Xcode 6.1

セルの画像を設定するときは、imageView.imageプロパティに画像をUIImageで設定します。

```
セル.imageView.image = イメージ
```

例 セルに画像を設定する

```
        cell.imageView.image = UIImage(named: "berry.png")
```

文字の色を設定する：var textColor: UIColor! Xcode 6.1

セルの文字色を設定するときは、通常テキストラベル(textLabel)か詳細テキストラベル(detailTextLabel?)の、textColorプロパティに色をUIColorで設定します。

```
セル.textLabel.textColor = 色
```

```
セル.detailTextLabel?.textColor = 色
```

例 テキストの文字の色を青にする

```
        cell.textLabel.textColor = UIColor.blueColor()
```

セルの背景色を設定する：var backgroundColor: UIColor?

backgroundColorプロパティで、背景色を設定します。

```
セル.backgroundColor = 色
```

例 セルの背景色を黒にする

```
        cell.backgroundColor = UIColor.blackColor()
```

フォントやサイズを設定する：var textLabel.font: UIFont!

Xcode 6.1

ラベルのフォントやサイズを設定するときは、通常テキストラベル(textLabel)か詳細テキストラベル(detailTextLabel?)の、fontプロパティにフォントをUIFontで設定します。

```
セル.textLabel.font = フォント
```

```
セル.detailTextLabel?.font = フォント
```

例 システムフォントでサイズ24で表示する

```
        cell.textLabel.font = UIFont.systemFontOfSize(24)
```

セルのアクセサリを設定する：UITableViewCellAccessoryType

セルの右端に詳細ボタンやチェックマークを表示するときは、accessoryTypeプロパティにアクセサリタイプを設定します。

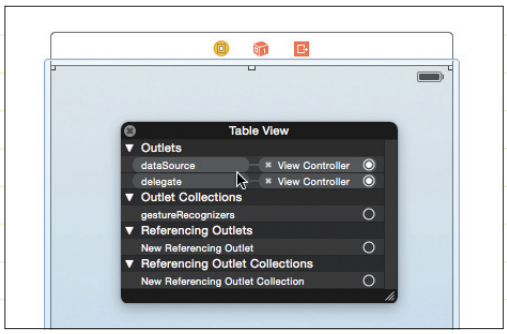
```
セル.accessoryType = アクセサリタイプ;
```

UITableViewCellAccessoryType.None または、 .None	 アクセサリなし
UITableViewCellAccessoryType.DisclosureIndicator または、 .DisclosureIndicator	 右向き矢印
UITableViewCellAccessoryType.DetailDisclosureButton または、 .DetailDisclosureButton	 詳細ボタン
UITableViewCellAccessoryType.Checkmark または、 .Checkmark	 チェックマーク

例 セルの右端にチェックマークを表示する

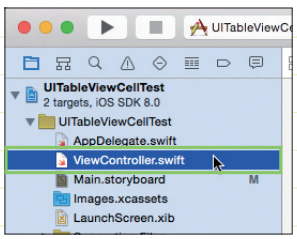
```
        cell.accessoryType = UITableViewCellAccessoryType.Checkmark
```

テーブルビューを右クリックすると接続を確認できます。



❷ プログラムを作る

「ViewController.swift」を選択します。
テーブルビューを使う準備をします。



テーブルビューを表示するプログラムを修正します。 Xcode 6.1

```
import UIKit

class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    // 行数
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 10
    }
    // 表示するセルの中身
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
        UITableViewCell {
        var cell = UITableViewCell(style: .Default, reuseIdentifier: "myCell")
        cell.textLabel.text = "\(indexPath.row)行目"
        // 文字を茶色にする
        cell.textLabel.textColor = UIColor.brownColor()
        // 文字サイズを20にする
        cell.textLabel.font = UIFont.systemFontOfSize(20)
        // チェックマークをつける
        cell.accessoryType = .Checkmark
        return cell
    }
    // ステータスバーを非表示にする
    override func prefersStatusBarHidden() -> Bool {
        return true
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

テーブルビューを表示するプログラムを修正します。 Xcode 6.1

```
import UIKit

class ViewController: UIViewController ,UITableViewDataSource, UITableViewDelegate{
    // フォント名用の配列を用意する
    var fontName_array:[String] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        // フォントファミリー名を調べて、配列に追加する
        for fontFamilyName in UIFont.familyNames()  {
            for fontName in UIFont.fontNamesForFamilyName(fontFamilyName as String)  {
                fontName_array.append(fontName as String)
            }
        }
    }
    // 行数
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return fontName_array.count
    }
    // 表示するセルの中身
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
        UITableViewCell {
        var cell = UITableViewCell(style: .Subtitle, reuseIdentifier: "myCell")
        // 配列から、このセルに表示するフォント名を取得して、フォントを設定する
        var fontname = fontName_array[indexPath.row]
        cell.textLabel.text = "ABCDE abcde 012345 あいうえお"
        cell.textLabel.font = UIFont(name: fontname, size: 18)
        // サブテキストにフォント名を表示する
        cell.detailTextLabel?.textColor = UIColor.brownColor()
        cell.detailTextLabel?.text = fontname
        return cell
    }
    // ステータスバーを非表示にする
    override func prefersStatusBarHidden() -> Bool {
        return true
    }
```

※iPhoneで使えるフォントのリストをそのフォントで表示しています。

結果