

# TP 3

## Unit Converter

### À la fin de ce TP :

- Faire une archive contenant les projets Xcode des exercices
- Envoyer l'archive à [ahumiliere@captaintrain.com](mailto:ahumiliere@captaintrain.com) avec l'objet : [DANT] TP 3 – Prénom Nom
- Si le TP est fait à plusieurs, préciser les noms et adresses mail de chacun

*L'objectif de ce TP est de développer une application pour convertir une température en Celsius vers Fahrenheit.*

*Le TP se base sur un nouveau projet Xcode à créer, de type « Single View Application ».*

## Exercice 1

- Avec Interface Builder et la Librairie d'Objets (⌘L), ajouter un label pour la température convertie. Ajuster la taille et la police du label.
- Ajuster les contraintes du label pour qu'il soit positionné à une distance fixe du haut, et centré horizontalement.
- Ajouter une « Picker view » en bas de l'interface et ajuster les contraintes.
- Utiliser l'Assistant (⌘⌘) pour visualiser le rendu sur différentes tailles d'écrans.
- Lancer l'application (⌘R) et essayer d'utiliser le picker.

## Exercice 2

- La Picker View permet de faire un choix dans une liste de textes ou de dates. Elle utilise les design pattern *delegate* et *data source*.
- Définir ViewController comme le data source de la picker view. Pour cela, faire un clic-droit sur la picker view, et placer une connection de dataSource vers le View Controller.
- Lancer l'application, observer le crash et consulter l'affichage dans la console.
- ViewController n'implémente pas les méthodes du protocole UIPickerViewDataSource appelées sur le data source de la Picker View.
- Avec la documentation d'Xcode et de l'API (⌘0), explorer la référence du protocole UIPickerViewDataSource et les méthodes numberOfComponentsInPickerView: et pickerView:numberOfRowsInComponent:.
- Indiquer que ViewController implémente le protocole UIPickerViewDataSource.

```
class ViewController: UIViewController, UIPickerViewDataSource {
```

- Ouvrir l'Issue Navigator (⌘4) et observer les warnings qui indiquent les méthodes nécessaires pour se conformer au protocole UIPickerViewDataSource.
- Implémenter numberOfComponentsInPickerView: et pickerView:numberOfRowsInComponent:. Expérimenter différentes valeurs de retour pour chaque méthode.

```
func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
    return 1
}

func pickerView(pickerView: UIPickerView, numberOfRowsInComponent
    component: Int) -> Int {
    return 10
}
```

- Lancer l'application et observer le résultat sur le nombre d'éléments affichés et le nombre de colonnes.

## Exercice 3

- Sans delegate pour déterminer quoi afficher, la picker view n'affiche que des ? par défaut.
- Définir ViewController comme le delegate de la picker view. Pour cela, faire un clic-droit sur la picker view, et placer une connection de delegate vers le View Controller.
- Indiquer que ViewController implémente le protocole UIPickerViewDelegate.

```
class ViewController: UIViewController, UIPickerViewDataSource,
    UIPickerViewDelegate {
```

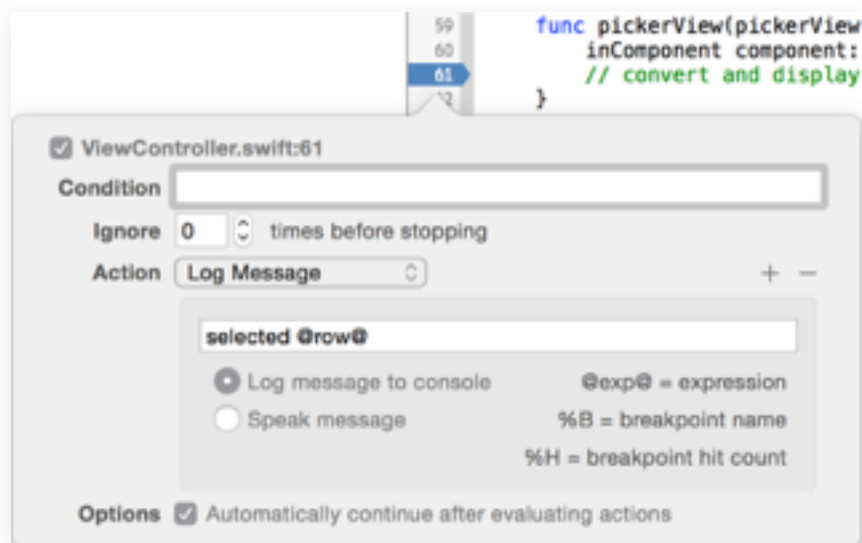
- Avec la documentation d'Xcode et de l'API, explorer la référence du protocole UIPickerViewDelegate et les méthodes pickerView:titleForRow:forComponent: et pickerView:didSelectRow:inComponent:.
- 
- Dans la classe ViewController, implémenter pickerView:titleForRow:forComponent:.

```
func pickerView(pickerView: UIPickerView, titleForRow row: Int,
    forComponent component: Int) -> String? {
    return "N°C"
}
```

- Dans la classe ViewController, implémenter pickerView:didSelectRow:inComponent:.

```
func pickerView(pickerView: UIPickerView, didSelectRow row: Int,
    inComponent component: Int) {
    // convert and display temperature
}
```

- Ajouter un point d'arrêt à pickerView:didSelectRow:inComponent:. Il générera un message dans les logs : selected: @row@.



- Lancer l'application et observer les valeurs affichées dans la picker view. Ouvrir la console et observer les messages qui s'affichent à chaque changement de ligne.

## Exercice 4

- On cherche à afficher une liste de températures positives et négatives dans le picker.
- Dans le controller, ajouter une property pour un Array contenant des valeurs de températures que le controller pourra fournir au picker pour l'affichage.

```
private var temperatureValues = [Int]()
```

- Implémenter une affectation (*arbitraire*) de temperatureValues dans viewDidLoad.

```
override func viewDidLoad() {
    super.viewDidLoad()
    temperatureValues = [1, 2, 3, 4, 5]
}
```

- Mettre à jour l'implémentation de pickerView:titleForRow:forComponent:.

```
func pickerView(pickerView: UIPickerView, titleForRow row: Int,
               forComponent component: Int) -> String? {
    let celsiusValue = temperatureValues[row]
    return "\(celsiusValue)°C"
}
```

- Lancer l'application, tester le results picker jusqu'à ce que l'application crashe. Observer le message dans la console.
- Mettre à jour pickerView:titleForRow:forComponent: en utilisant la taille de temperatureValues pour informer la results picker du nombre de lignes à afficher.
- Lancer l'application et observer le résultat.
- Plutôt que de créer un tableau statique contenant des valeurs de températures de -100 à 100, nous allons le créer dynamiquement. Modifier viewDidLoad pour créer ce tableau dynamiquement avec une boucle for.

```

override func viewDidLoad() {
    super.viewDidLoad()
    let lowerBound = -100
    let upperBound = 100
    for var index = lowerBound; index <= upperBound; ++index {
        temperatureValues.append(index)
    }
}

```

- Remplacer la boucle for par une boucle for-in renvoyant le même résultat.

## Exercice 5

- On cherche maintenant à convertir la température de la picker view en degrés Fahrenheit.
- Avec Interface Builder, créez une propriété IBOutlet vers le label.

```
@IBOutlet weak var temperatureLabel: UILabel!
```

- Mettre à jour la méthode `pickerView:didSelectRow:inComponent:`. Elle effectuera la conversion (*fahrenheit = 1,8 \* celsius + 32*) et entrera cette valeur dans le label : `temperatureLabel.text = ...`
- Lancer l'application et observer le résultat.
- Convertir la température en `Int` pour obtenir un affichage propre.

## Exercice 6 : MVC

- Le design pattern MVC, largement plébiscité sur iOS, permet de bien découper la logique du code de la gestion de l'affichage. Ici, nous allons chercher à créer un modèle prenant en charge la conversion des températures, pour extraire cette logique du View Controller.
- Ajouter une nouvelle classe Swift au projet pour le modèle `UnitConverter`.

```

import Foundation

class UnitConverter {

}

```

- Le contenu de la méthode `pickerView:didSelectRow:inComponent:` a sa place dans le modèle. Ajouter une méthode `degreesFahrenheit:` à la classe `UnitConverter`.

```

func degreesFahrenheit(degreesCelsius: Int) -> Int {
    return Int(1.8 * Float(degreesCelsius) + 32.0)
}

```


- Dans la classe `ViewController`, déclarer une nouvelle propriété pour un objet statique de type `UnitConverter` et lui affecter `UnitConverter()` par défaut.
- Mettre à jour `pickerView:didSelectRow:inComponent:` pour utiliser `UnitConverter.degreesFahrenheit:`

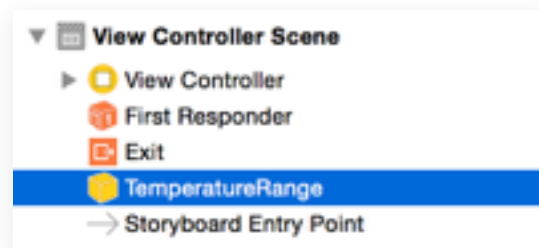
## Exercice 7 : MVC


- L'intervalle de températures n'a rien à voir avec la conversion et devrait être placé dans un autre modèle, indépendamment du View Controller et de `UnitConverter`. On utilise ici un « View Model » dont l'unique but est de servir la vue.
- Discuss the concept of a "view model": a model object whose sole purpose is to serve the view.
- Ajouter une classe Swift `TemperatureRange` au projet.

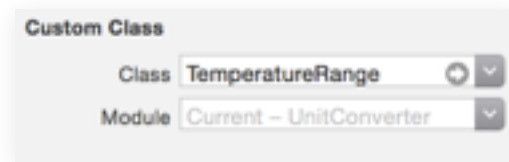
```
import Foundation

class TemperatureRange {
}
```

- Nous allons maintenant chercher à remplacer le View Controller par `TemperatureRange` comme data source de la picker view.
- De retour dans Interface Builder et la librairie d'objets, sélectionner un *Object* et l'ajouter à la *View Controller Scene* dans *Document Outline* () . Renommer l'objet en **TemperatureRange**.



- Avec l'objet `TemperatureRange` sélectionné, utiliser l'Identity Inspector () pour définir la classe comme étant `TemperatureRange`.



- Toujours avec InterfaceBuilder, changer le dataSource de la picker view pour le mettre sur `Temperature Range`.
- Lancer l'application et l'observer crasher. `TemperatureRange` doit implémenter le protocole `UIPickerViewDataSource`.

## Exercice 8 : MVC

- Changer le `import` de `TemperatureRange` pour lui donner accès à `UIPickerViewDataSource`.

```
import UIKit
```

- Mettre à jour `TemperatureRange` pour hériter de `NSObject` et adopter le protocole `UIPickerViewDataSource`.

```
class TemperatureRange: NSObject, UIPickerViewDataSource {
```

- Retirer la référence au protocole UIPickerViewDataSource de la classe ViewController.

```
class ViewController: UIViewController, UIPickerViewDelegate {
```

- Déplacer temperatureValues dans la classe TemperatureRange et raccourcir son nom en values. Ici, l'utilisation de map permet de simplifier la syntaxe. Chercher le comportement de la fonction map().

```
let values = map(-100...100) { $0 }
```

- Déplacer les méthodes du controller numberOfComponentsInPickerView: et pickerView:numberOfRowsInComponent: dans la classe TemperatureRange.

```
func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
    return 1
}

func pickerView(pickerView: UIPickerView,
    numberOfRowsInComponent component: Int) -> Int {
    return values.count
}
```

- La classe ViewController contient maintenant des erreurs.
- Avec Interface Builder et l'assistant, faire un drag & drop du clic droit pour créer une connexion de l'objet TemperatureRange vers le controller pour créer une nouvelle propriété..

```
@IBOutlet var temperatureRange: TemperatureRange!
```

- Mettre à jour les méthodes du controller pour utiliser TemperatureRange dans les méthodes pickerView:titleForRow:forComponent: et pickerView:didSelectRow:inComponent:.
- Lancer l'application et observer le résultat.
- Le code restant de la controller ne fait que gérer la communication entre la vue et le modèle. C'est le rôle du controller.

## Exercice 9

- Nous allons maintenant essayer d'améliorer l'expérience utilisateur de l'application. Par exemple en spécifiant une valeur par défaut pour la picker view.
- Créer un IBOutlet pour la picker view dans le view controller.

```
@IBOutlet weak var celsiusPicker: UIPickerView!
```

- Définir la température sélectionnée par défaut dans viewDidLoad. avec la méthode selectRow:inComponent:animated:.
- Lancer l'application. Si la valeur par défaut du picker est bonne, le label n'est pas mis à jour. Faire en sorte qu'il le soit en appelant la méthode adaptée dans viewDidLoad, de façon élégante et sans introduire de duplication de code.
- Lancer l'application et observer les changements.

## Exercice 10 : UserDefaults

- Lancer l'application et sélectionner une température. Mettre l'application en arrière-plan (⌘+H) et réouvrir l'application. Remarquer comment la température sélectionnée l'est toujours.
- Relancer l'application. La température sélectionnée auparavant ne l'est plus. Pour l'expérience utilisateur, il faudrait conserver en permanence la mémoire de la dernière température sélectionnée.
- Nous allons utiliser la classe `NSUserDefaults` qui permet de sauvegarder des informations de façon persistante dans l'application.
- Mettre à jour `pickerView:didSelectRow:inComponent:` pour enregistrer la ligne du picker sélectionnée.

```
let defaults = UserDefaults.standardUserDefaults()
defaults.setInteger(row, forKey: "defaultCelsiusPickerRow")
defaults.synchronize()
```

- La méthode `pickerView:didSelectRow:inComponent:` a maintenant deux responsabilités : afficher la température convertie et sauvegarder la dernière ligne sélectionnée. Extraire le code dans deux méthodes distinctes et les appeler dans `pickerView:didSelectRow:inComponent:`. Chaque méthode exécute une tâche précise.

```
func displayConvertedTemperatureForRow(row: Int)
func saveSelectedRow(row: Int)
```

- Dans un souci de clarté et pour minimiser les erreurs, extraire le string "defaultCelsiusPickerRow" dans une constante en haut de la classe et utiliser cette constante dans `saveSelectedRow:`.

```
let userDefaultsLastRowKey = "defaultCelsiusPickerRow"
```

- Lancer l'application, sélectionner la température. Relancer l'application, la température est sauvegardée mais pas restaurée.

## Exercice 11 : UserDefaults

- Réécrire `viewDidLoad` et utiliser une méthode (à créer) `initialPickerRow` pour déterminer la valeur initiale du picker.

```
override func viewDidLoad() {
    super.viewDidLoad()
    let row = initialPickerRow()
    celsiusPicker.selectRow(row, inComponent: 0, animated: false)
    pickerView(celsiusPicker, didSelectRow: row, inComponent: 0)
}

func initialPickerRow() -> Int {
    return celsiusPicker.numberOfRowsInComponent(0) / 2
}
```

- Dans la documentation, chercher et découvrir le fonctionnement de la méthode `NSUserDefaults integerForKey:`.
- Implémenter la méthode `initialPickerRow`.

```
func initialPickerRow() -> Int {
    let savedRow = UserDefaults.standardUserDefaults()
        .integerForKey(userDefaultsLastRowKey)
    if savedRow != 0 {
        return savedRow
    } else {
        return celsiusPicker.numberOfRowsInComponent(0) / 2
    }
}
```

- Lancer puis relancer l'application et observer le fonctionnement de la persistance. Que se passe-t-il si l'utilisateur sélectionne la première ligne du picker et redémarre l'application ?
- Dans la documentation, chercher et découvrir le fonctionnement de la méthode `NSUserDefaults objectForKey:`. Qu'est ce que le type `AnyObject` ? La méthode renvoie la valeur sauvegardée ou nil si la valeur n'est pas trouvée. C'est ce « ou nil » qui est matérialisé par le « ? ».
- Mettre à jour l'implémentation de `initialPickerRow`.

```
func initialPickerRow() -> Int {
    let savedRow = UserDefaults.standardUserDefaults()
        .objectForKey(userDefaultsLastRowKey) as? Int
    if let row = savedRow {
        return row
    } else {
        return celsiusPicker.numberOfRowsInComponent(0) / 2
    }
}
```

- Documentez-vous par vous même sur les « ? » (Optionals). On en parle au prochain cours !
- Lancer l'application. Enjoy !