TP 2
Swift, Clean code

À la fin de ce TP :

- Faire une archive contenant les projets Xcode des exercices
- Envoyer l'archive à ahumiliere@captaintrain.com avec l'objet : `[DANT] TP 2 – Prénom Nom`
- Si le TP est fait à plusieurs, préciser les noms et adresses mail de chacun

Exercice 1

- Ouvrir **main.swift**.
- Lancer le programme (⌘**R**) et observer l'affichage dans la console (⇧⌘**C**).
- Declarer deux variables.

```
var numberOfPlanets: Int = 8
var diameterOfEarth: Float = 24859.82 // In miles, from pole to pole.
```

- Retirer l'impression de `Hello World!` et ajouter vos propres appels à `print` sous les déclarations de variables.

```
println("Welcome to our solar system!")
println("There are \(numberOfPlanets) planets to explore.")
println("You are currently on Earth, which has a circumference of
   \(diameterOfEarth) miles.")
```

- Lancer le programme et observer l'affichage dans la console.
- Retirer les annotations de types des deux déclarations de variables.

```
var numberOfPlanets = 8
var diameterOfEarth = 24859.82 // In miles, from pole to pole.
```

- Lancer le programme et s'assurer qu'il tourne toujours.
- Les valeurs des deux variables ne changent pas pendant l'execution. Les transformer en constantes.

```
let numberOfPlanets = 8
let diameterOfEarth = 24859.82 // In miles, from pole to pole.
```

- Lancer le programme et s'assurer qu'il tourne toujours.

En Swift, c'est une bonne pratique de toujours déclarer une constante/variable en let *et de ne la passer en* var *que si nécéssaire.*

Exercice 2

- On cherche à récupérer le nom de l'utilisateur, pour le stocker et l'afficher dans la console.
- Implémenter le code permettant de récupérer l'input de la console, avec une fonction utilitaire fournie, `getln`, qui permet de récupérer le texte entré par l'utilisateur dans le terminal.

```
print("What is your name?")
let name = getln()
print("Nice to meet you, \(name). My name is Eliza, I'm an old
   friend of Siri.")
```

- La fonction `getln` n'est pas fournie avec la librairie Standard de Swift. C'est une fonction fournie avec le sujet du TP, visible et/ou modifiable dans **HelperFunctions.swift**.
- Avant de lancer l'application, Xcode compile tous les fichiers .swift présents.
- Lancer le programme et interagir avec la console.
- On cherche à suggérer à l'utilisateur une aventure et à lui demander si il ou elle souhaite que le programme choisisse pour lui une destination aléatoire.

```
print("Let's go on an adventure!")
print("Shall I randomly choose a planet for you to visit? (Y or
   N)")
let decision = getln()
```

- Le programme va prendre une décision en fonction du chois de l'utilisateur. Implémenter cette prise de décision avec un `if/else`.

```
if decision == "Y" {
   print("Ok! Traveling to...")
   // TODO: travel to random planet
} else {
   print("Ok, name the planet you would like to visit...")
   // TODO: let the user select a planet to visit
}
```

- Lancer le programme, interagir avec la console et observer le résultat.

Exercice 3

- On cherche maintenant à savoir si l'utilisateur veut *vraiment* que l'on choisisse pour lui *ou pas* une destination. Pour cela, nous allons réitérer la question « jusqu'à ce que l'utilisateur réponde Y ou N ». Modifier l'user input en ce sens.

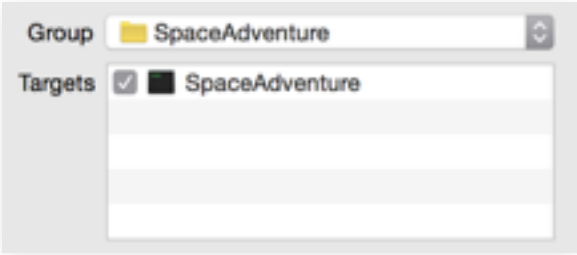
```
var decision = "" // Start with empty String
while !(decision == "Y" || decision == "N") {
   decision = getln()
   if decision == "Y" {
      print("Ok! Traveling to...")
      // TOOD: travel to random planet
   } else if decision == "N" {
      print("Ok, name the planet you would like to visit...")
      // TODO: let the user select a planet to visit
   } else {
      print("Sorry, I didn't get that.")
   }
}
```

- `decision` est maintenant une variable.
- Lancer le programme, interagir avec la console et mettre à l'épreuve l'input.

Exercice 4

On va maintenant chercher à structurer le code en objets.

- Ajouter un nouveau fichier Swift (⌘**N**) appelé **SpaceAdventure.swift**. S'assurer que le groupe SpaceAdventure est sélectionné, et que le target SpaceAdventure est coché.



- Par convention et dans un soucis de clarté, on utilise toujours un fichier pour une seule classe.Le nom de ce fichier (**SpaceAdventure.swift**) doit refléter le nom de la classe qu'il contient.
- A terme le **main.swift** ne devra servir qu'à deux choses : créer un objet `SpaceAdventure` et démarrer cette aventure.
- Au dessus du code existant dans **main.swift**, instancier un objet `SpaceAdventure`.

```
import Foundation

let adventure = SpaceAdventure()

let numberOfPlanets = 8
...
```

- Observer les erreurs dans Xcode. La classe `SpaceAdventure` n'a pas encore été définie, on ne peut donc pas créer un objet `SpaceAdventure`.
- Ouvrir **SpaceAdventure.swift** et implémenter une classe minimale.

```
class SpaceAdventure {

}
```

- Retourner sur **main.swift**, les erreurs ont disparu.
- On chercher maintenant à appeler une méthode sur `SpaceAdventure` pour lancer l'aventure. Ajouter cet appel sur l'instance de `SpaceAdventure`.

```
adventure.start()
```

- Observer les erreurs dans Xcode.

Exercice 5

Les objets de type `SpaceAdventure` ne savent pas comment prendre en charge les appels de méthode `start`.

- Ajouter une implémentation vide de la méthode `start` dans la classe `SpaceAdventure`.

```
class SpaceAdventure {

    func start() {
    }

}
```

- Retourner sur **main.swift**, les erreurs ont disparu.
- Notre objet sait maintenant répondre à la méthode `start`, mais elle ne fera pas grand chose puisque son implémentation est vide. Couper-coller le code existant dans **main.swift** dans l'implémentation de la méthode `start`.

```
func start() {
    let numberOfPlanets = 8
    ...
```

- Lancer le programme et interagir avec la console pour vérifier que les fonctionnalités préexistantes sont toujours là.
- **main.swift** est maintenant plus concis, lisible et expressif.

Exercice 6

- La méthode `start` semble faire 3 choses bien distinctes : afficher une introduction, souhaiter la bienvenue à l'utilisateur et déterminer sur quelle planète il veut voyager.
- Extraire les premières lignes de `start` dans une nouvelle méthode privée et remplacer le code extrait par un appel à la méthode.

```
private func displayIntroduction() {
    let numberOfPlanets = 8
    let diameterOfEarth = 24859.82 // In miles, from pole to pole.
    print("Welcome to our solar system!")
    print("There are \(numberOfPlanets) planets to explore.")
    print("Your are currently on Earth, which has a circumference
        of \(diameterOfEarth) miles.")
}
```

- La méthode `displayIntroduction` sera seulement appelée par la méthode `start`. Elle est marquée comme privée pour que seul le code issu du même fichier puisse l'appeler.

- La méthode `start` utilise à deux reprises la paire `print` et `getln` pour afficher un message et attendre une réponse. Encapsuler ce travail dans une méthode séparée `private func responseToPrompt(prompt: String) -> String`. Cette méthode va afficher le `String` passé en paramètre et retourner ce que l'utilisateur aura entré dans la console.

- Extraire et remplacer le code restant dans start par deux méthodes `greetAdventurer` et `determineDestination`.
- La méthode `start` doit maintenant être de cette forme :

```
func start {
   displayIntroduction()
   greetAdventurer()
   print("Let's go on an adventure!")
   determineDestination()
}
```

- Lancer le programme, interagir avec la console pour vérifier que rien n'a changé.

Exercice 7

- Nous allons maintenant nous attacher à créer une collection de planète (destinations), en utilisant une classe `PlanetarySystem`.
- Ajouter un nouveau fichier Swift au projet, appelé **PlanetarySystem.swift**.
- Ajouter une implémentation basique de `PlanetarySystem`.

```
class PlanetarySystem {

}
```

- Ajouter la déclaration d'une propriété pour représenter le nom du système.

```
class PlanetarySystem {

   let name: String

}
```

- On utilise ici une constante car le nom n'est pas amené à changer. L'annotation de type est ajoutée car la propriété n'a pas de valeur par défaut permettant de déduire son type.
- Xcode affiche des erreurs. Son mécontentement vient du fait que la propriété `name` n'est pas initialisée. Swift requiert que toutes les constantes soient assignées pendant l'initialisation.
- Ajouter un initialiseur à la classe `PlanetarySystem`.

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

- Nous devons maintenant faire le lien entre la classe `SpaceAdventure` et `PlanetarySystem`, pour que l'aventure puisse savoir dans quel système elle se déroule.
- Ajouter une propriété de type `PlanetarySystem` à la classe `SpaceAdventure`. et lui attribuer une valeur par défaut (`"Solar System"`).

```
class SpaceAdventure {

   let planetarySystem = PlanetarySystem(name: "Solar System")
...
```

- Mettre à jour `displayIntroduction` pour supprimer du code statique et utiliser `PlanetarySystem` `name` pour afficher le message d'introduction.

```
private func displayIntroduction() {
    let numberOfPlanets = 8
    print("Welcome to the \(planetarySystem.name)!")
    print("There are \(numberOfPlanets) planets to explore.")
}
```

- Lancer le programme, manipuler la console et observer les changements effectués.