**Évaluation formative**

**Commentaire du code Scala** avec IntelliJ Idea

**On a commenté 05 projets**

Code Expression

-ProjetDeCours

-Classes

-FirstApp

-Fonctions.

**1/Commentaires du projet Code Expression**

Une image contenant texte

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

**Détail du code :**

//Statements  
val myVariable = 10  
*println*("Hello World!")

2.13.10.jar;C:\Users\admin\AppData\Local\Coursier\cache\v1\https\repo1.maven.org\maven2\org\scala-lang\scala-reflect\2.13.10\scala-reflect-2.13.10.jar main

Hello World!

* Dans ce cas, le code exécuté est celui qui sera affiché après la commande print. Et non pas le statement. Qui a une la valeur déclarée.
* Statement est une instruction que le programme va exécuter. Cela inclut l’assignement du statement et la commande print.

//Expressions  
val myVar = 5  
  
val calculus = {val PI=3.14; PI\*myVar}  
*println*("Calculus = " + calculus)

Calculus = 15.700000000000001

* Expressions est une combinaison de variables, de valeurs et d’opérateurs qui vont être représentés par une valeur unique de sortie.

val number = 2  
val divideByTwo = if(number % 2 == 0) {*println*("Divisible by two!")} else {None}  
*println*(divideByTwo)  
val result = 3 \* 2 \* {if(number %2 == 0) {number \* 2} else {0}}  
*println*("result = " + result)

Divisible by two!

()

result = 24

//Match Expressions  
val laptop = "PC"  
val laptopType = laptop match {  
 case "Mac" => "hello apple!"  
 case "PC" => "Hellow PC!"  
 case nothingMatches => "Are you sure you're a laptop? :)"  
}  
*println*("laptop = " + laptopType)

laptop = Hellow PC!

* Match expressions on utilise un nombre de case statements qui peuvent matcher les valeurs possibles. Cela peut aussi faire des matchs entre valeurs numériques et de type string.
* On utilise aussi dans ce cas le ‘case \_ ‘ pour mentionner toutes les autres valeurs qui ne font pas le match.

val productType = "PC"  
val productTypeMatch = productType match {  
 case "Mac" | "iPod" | "iPad" => "Hello Apple!"  
 case "PC" | "Surface" | "WindowsPhone" => "Hello Microsoft!"  
 case nothingMatches => "Sorry, don't know what you're talking about..."  
}  
*println*("product type = " + productTypeMatch)

product type = Hello Microsoft!

//Pattern Guards  
val productType2 = "Linux"  
val productTypeMatch2 = productType2 match {  
 case "Mac" | "iPod" | "iPad" => "Hello Apple!"  
 case "PC" | "Surface" | "WindowsPhone" => "Hello Microsoft!"  
 case nothingMatches if nothingMatches == "Linux" => "Great Choice!"  
 case nothingMatches => "Sorry, don't know what you're talking about..."  
}  
*println*("product type = " + productTypeMatch2)

product type = Great Choice!

* Dans le cas du Pattern Guard, on veut s’assurer que certaines conditions peuvent se réaliser, a l’intérieur de la clause Case.

//Match-All  
val matchAll = "Blaah"  
val matchEmAll = matchAll match {  
 case "Mac" => "hello apple!"  
 case "PC" => "Hellow PC!"  
 case nothingMatches => s"What do you mean by **$**nothingMatches?"  
}  
*println*("Match All = " + matchEmAll)

Match All = What do you mean by Blaah?

//wildcards  
val wildCard = "Blaah"  
val wildCardVal = wildCard match {  
 case "Mac" => "hello apple!"  
 case "PC" => "Hellow PC!"  
 case \_ => s"I don't think that your product is a PC or Apple, it's **$**wildCard"  
}  
*println*("Wildcard = " + wildCardVal)

Wildcard = I don't think that your product is a PC or Apple, it's Blaah

* C’est une situation de match avec l’utilisation des Cases, On utilise dans ce cas le ‘case \_ ‘ pour mentionner toutes les autres valeurs qui ne font pas le match.

//Down casting - pattern variables  
val typedVariable:Any = 2.3  
val typeOfVariable = typedVariable match {  
 case typedVariable:Int => "integer, I can do my calculations."  
 case typedVariable:String => "string, cannot do calculations but I can use the value for text."  
 case typedVariable:Double => "Double, let's go!"  
 case \_ => "Any"  
}  
*println*("The type of my variable is: " + typeOfVariable)

The type of my variable is: Double, let's go!

* Le downcasting est plus utilisé en Java pour faire un cast du type au niveau de l’enfant (Child) est plus utilisé que le upcasting qui effectue un cast du type d’objet au niveau du parent. Il s’agit d’une conversion d’objet d’u type a un autre.

//Closures  
val y = 2  
val addYVariable = (x:Int) => x + y  
*println*("Adding an outside variable to my function: " + addYVariable(3))  
var extras = 10  
val fullPrice = (price:Int) => price + extras  
*println*("Full price with extras = " + fullPrice(40))  
  
*println*("################################ ")  
extras = 20  
*println*("Full price with extras = " + fullPrice(40))

Adding an outside variable to my function: 5

Full price with extras = 50

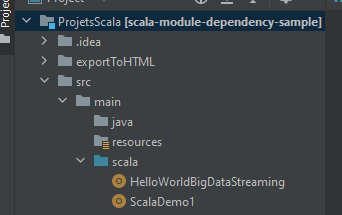
################################

Full price with extras = 60

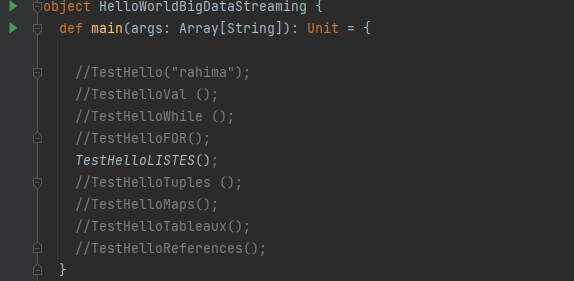
* Closure est une fonction qui permet d’utiliser des variables en dehors de la fonction pour effectuer des calculs. Dans notre cas, extras, fait référence a une variable qui se trouve en dehors de la fonction.

**2/Commentaire du projet -ProjetDeCours**

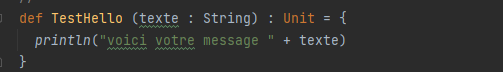
**1/présentation du projet**



On a défini une fonction main qui contient toutes les fonctions ci-dessus avec “def “qui nécessite plusieurs lignes c'est pour ça on a utilisé les accolades et les points virgules.



La fonction “TestHello” contient les paramètre texte du type string et retourne un message saisi par l’utilisateur.

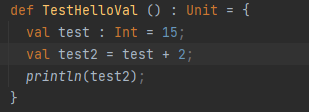




On définit une fonction qui s'appelle TestHelloVal qui retourne la somme entre deux valeurs immuables.

Déclaration d'une valeur immuable avec val test.

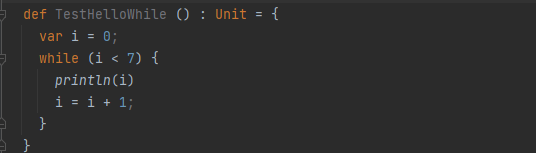
Déclaration d'une valeur immuable avec val test2.

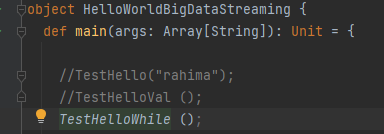


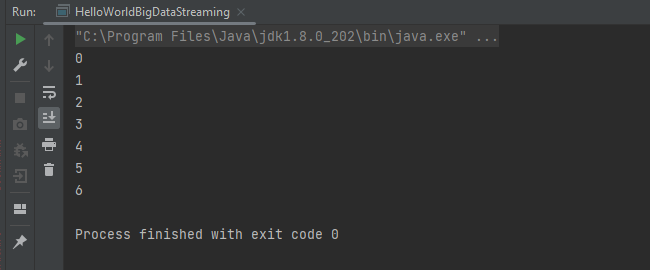


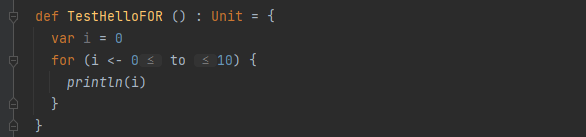


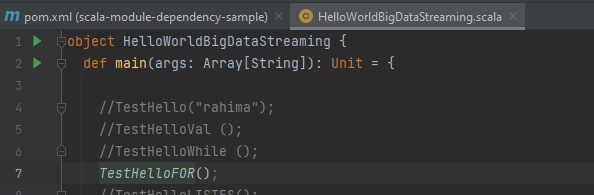
On définit une fonction TestHelloWhile qui sert à utiliser la boucle while suite à la déclaration d'une nouvelle variable en utilisant le mot clé var qui a été initie à 0 et la condition de la boucle qu'elle soit inferieur a 7 non compris.

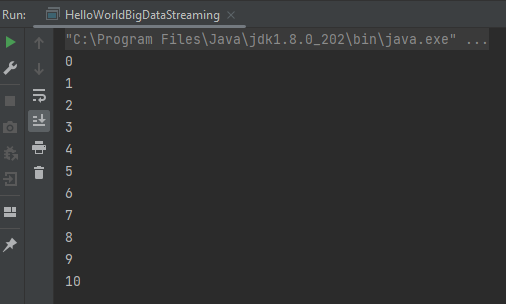






On définit la fonction TestHelloFOR qui utilise la boucle for, suite à la déclaration d'une nouvelle variable en utilisant le mot clé var qui a été initie à 0 et la condition de la boucle est que les valeurs soient entre 0 et 10 compris.  
  




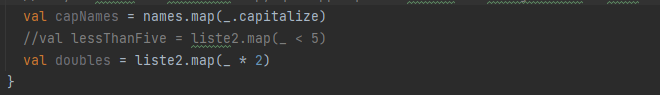


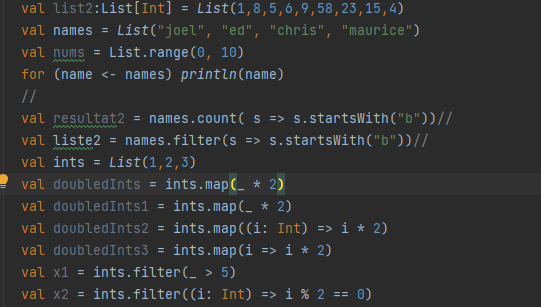
Les listes qui sont une collection immutable et ordonné de données de même type  
Déclaration de la liste en utilisant val names ensuite effectuer des opérations sur cette liste et aussi vérifier s'il y a des noms qui commencent par "b" et aussi filtrer les noms qui commencent par "b", ainsi multiplier la liste par deux en utilisant .map, aussi faire appel au filtre.

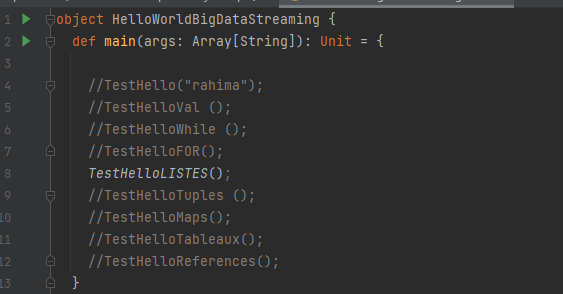
On utilise foreach pour parcourir pour chaque objet de la liste jusque à la fin des éléments.

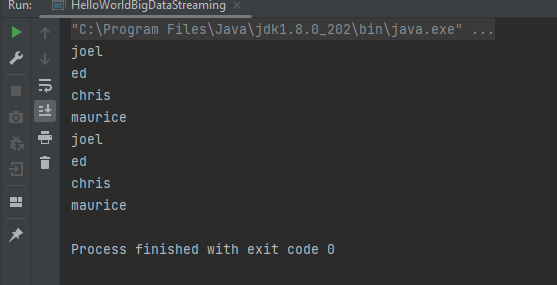


On utilise aussi la fonction map qui applique le calcul ou l’algorithme a tous les éléments de la liste









Les tuples sont des ensembles ordonnés de valeurs. Ce sont des enregistrements immuables qui peuvent être référencés par leur position. La référence dans les tuples commence à 1, et non 0 comme dans les tableaux et listes)

Déclaration d'une variable immuable

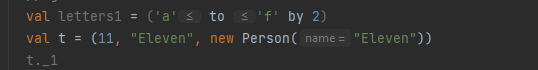


Création d’un classe Person avec un paramètre de type string.

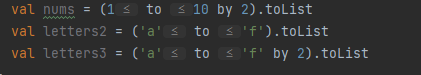


Déclaration d’une variable constante allant de la lettre “a” jusqu'au ‘f” avec un intervalle de 2.

Déclaration d’une variable immuable qui prend en paramètre nouveau objet de la classe Person.



Conversion des tuples en liste en utilisant toList

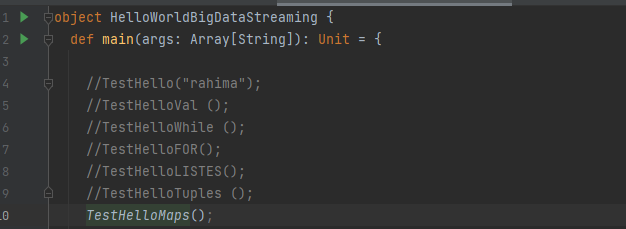


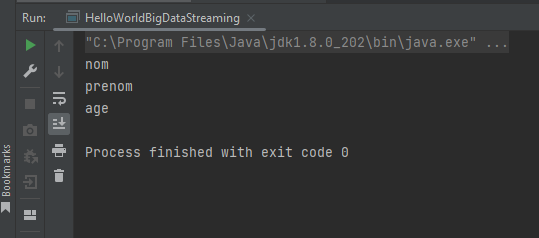
On peut créer un tableau de clé et valeur en utilisant le Map.



Si on veut extraire juste les clés de ce tableau en utilisant foreach pour chaque objet.







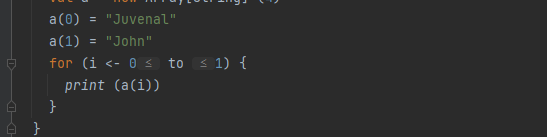
Déclaration d’un tableau de type string avec chargement des valeurs du même type



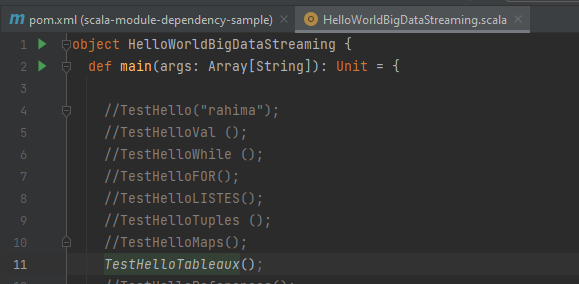
Déclaration d’une valeur immuable et l’affectation d’une valeur de type tableau string avec une taille de 4

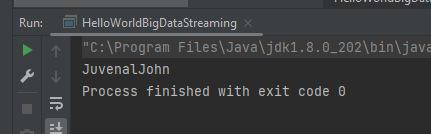


Affectation à la première case du tableau la variable “a” une valeur bien définie de type string même chose pour le deuxième "a” ensuite parcourir le tableau en utilisant la boucle for pour imprimer les valeurs du tableau.

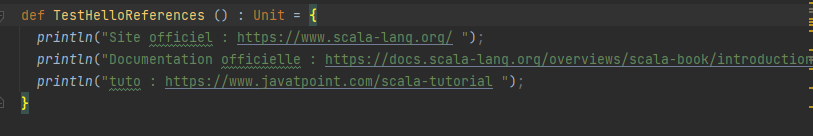


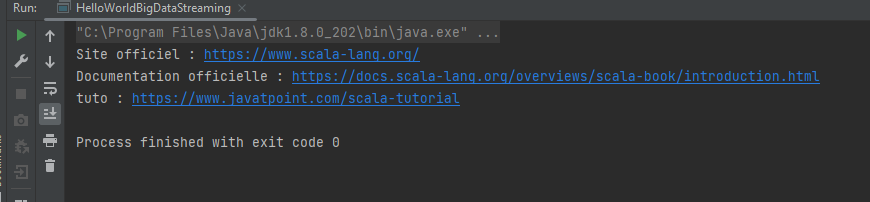
Le résultat du tableau



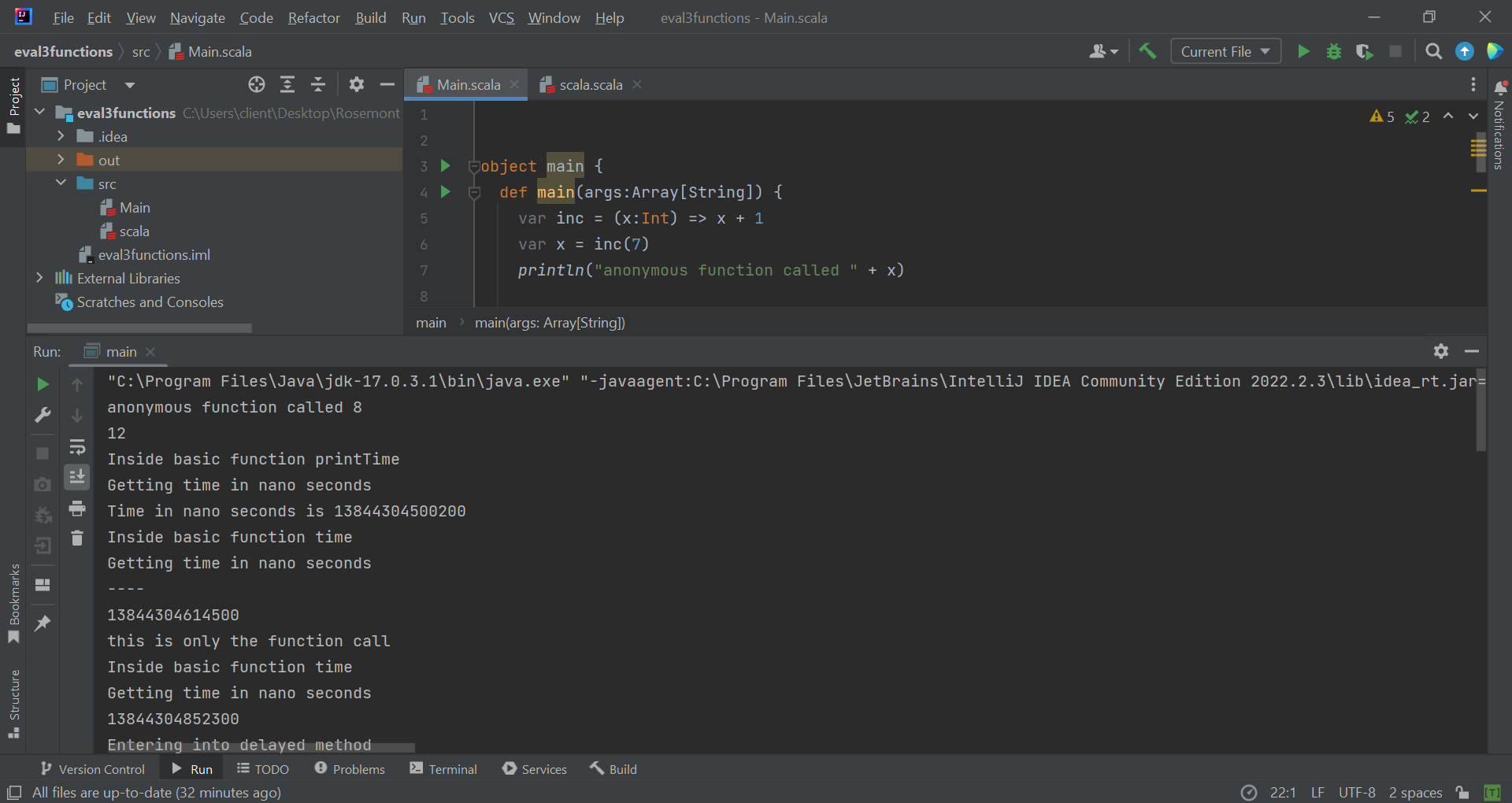


Déclaration d’une méthode qui permet l'affichage des liens hyper texte qui veut dire des liens des pages web accessibles sur le net.

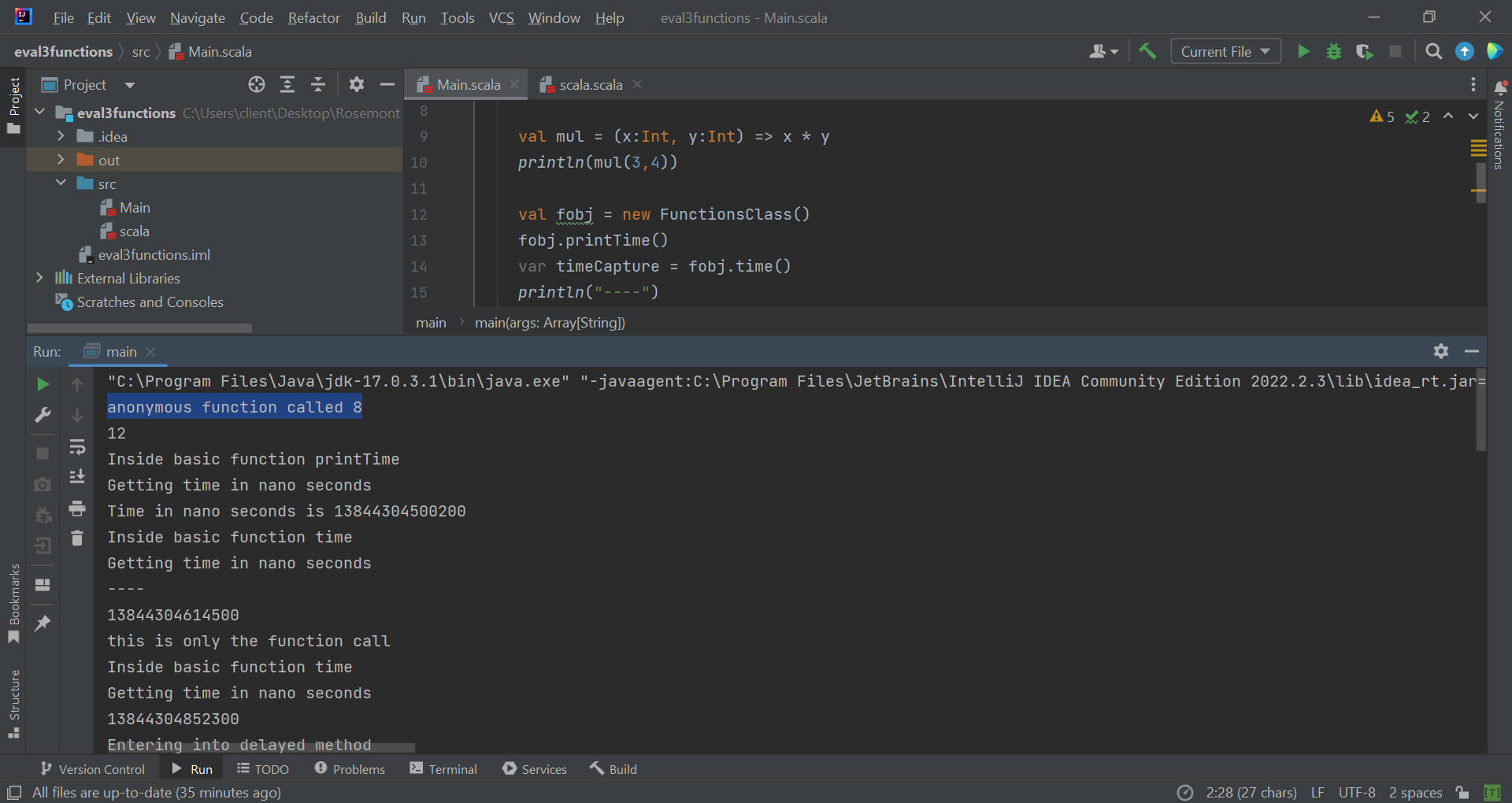




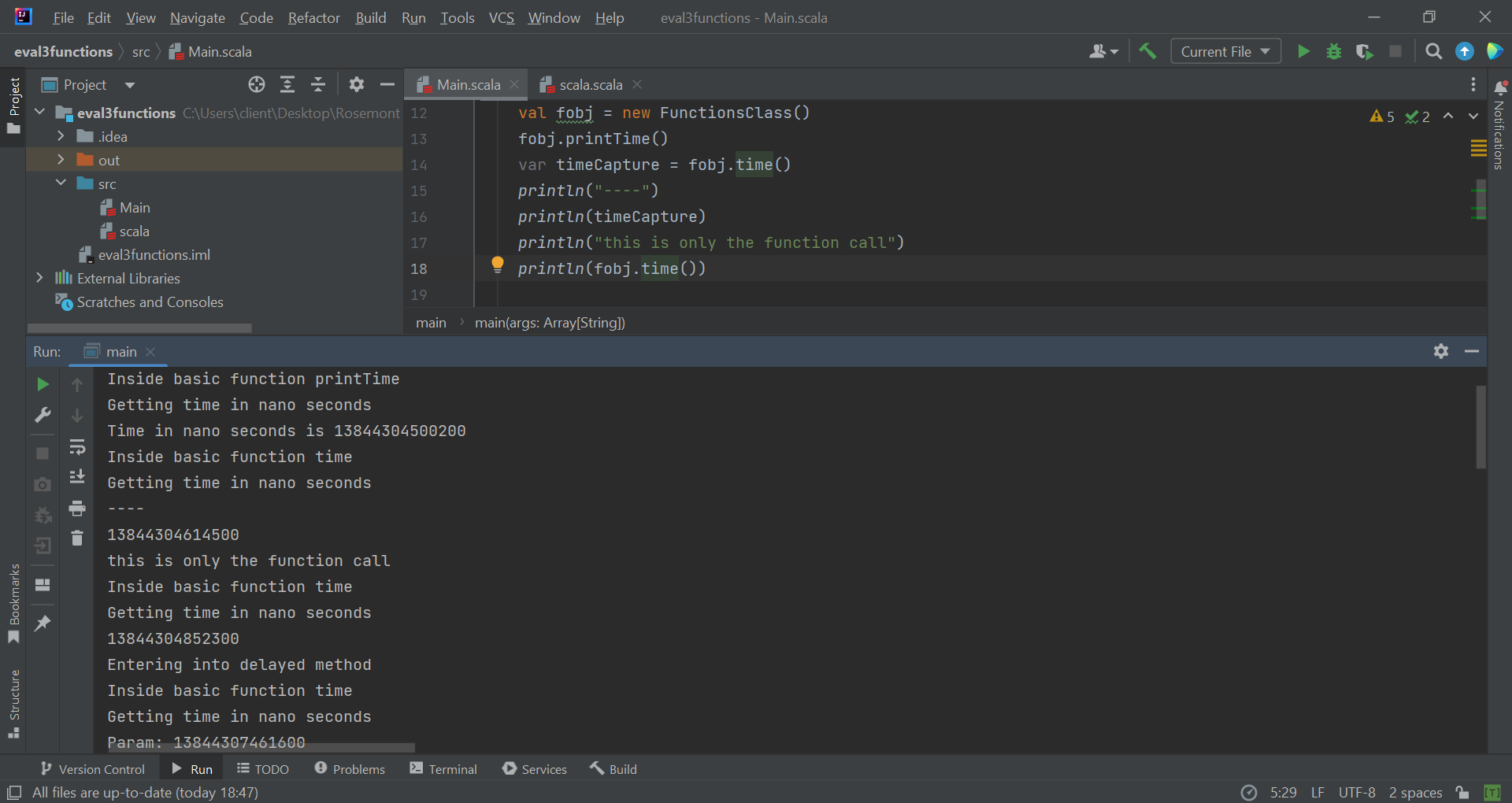
**3/Les commentaires du projet les fonctions**



Pour le chiffre 7 : le résultat est d’écrire anonymous function called et faire 7+1= 8

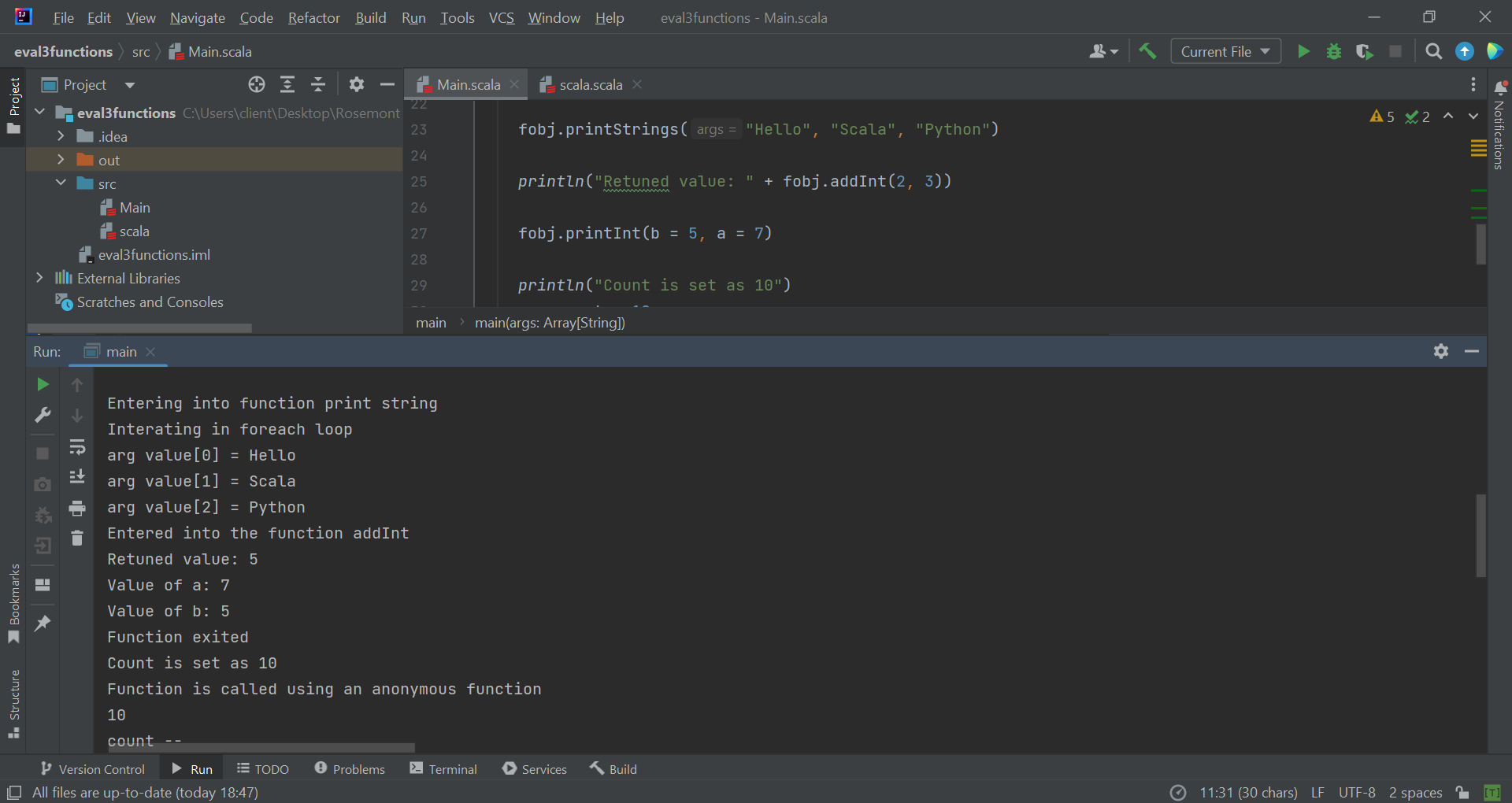


Pour la deuxième fonction il fait 3\*4=12



Il prend le temps de tout de suite et le transforme en nanosecondes.

Après il a mis ---- et a pris le temps de tout suite en nanosecondes et a écrit après this is only the function call



Il sort les valeurs de la matrice (1,3)

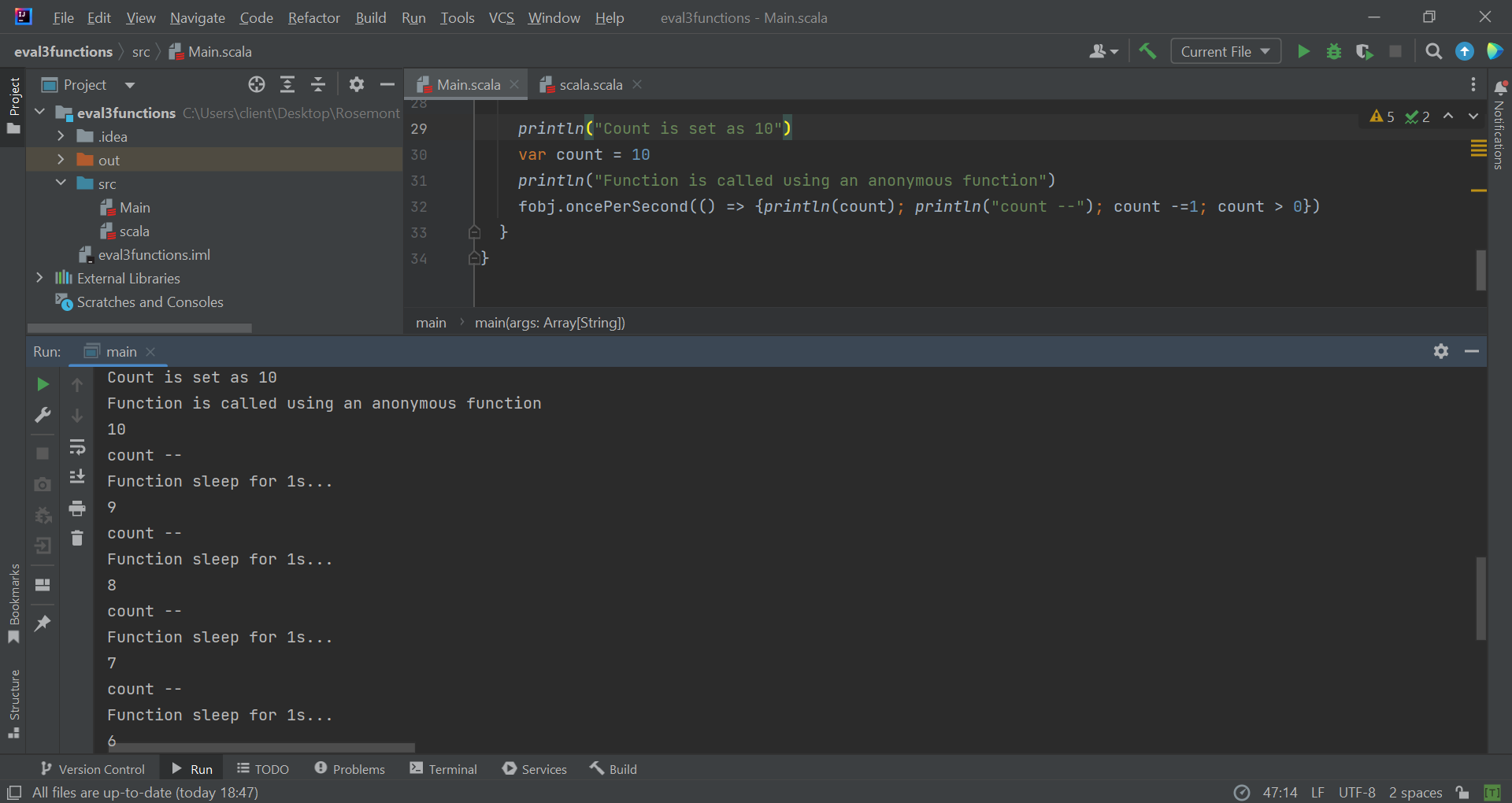
arg value[0] = Hello

arg value[1] = Scala

arg value[2] = Python

après il a fait la fonction addition 2+3=5

puis après il a affiche a=7 et ensuite b=5



ensuite il a affiche Function is called using an anonymous function

et il a compte de 10 a 1 en affichant a chaque fois

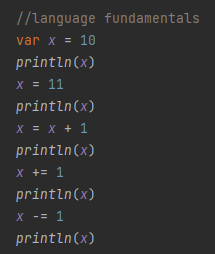
le chiffre compté

count --

le chiffre compté – 1.

**4/ Les commentaires pour le projet 1-m1\_firstapp**

Déclaration des variables : val et var

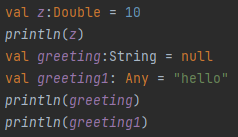


La différence entre val et var est que val rend une variable immuable et var rend une variable mutable

* println() pour afficher le résultat
* les opérations x = x+1 (x +=1) sont semblables au language Python

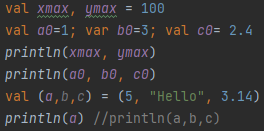


Déclaration du type de variable



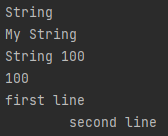
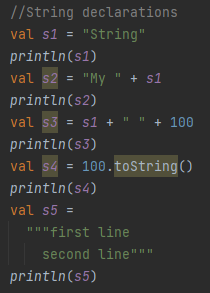
* Double : comme float
* String : lettres
* Any : super type

Déclaration de plusieurs variables



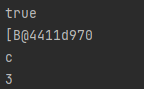
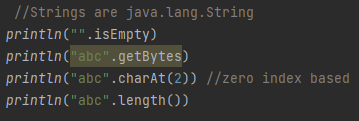
On a plusieurs façons de déclarer des variables

Déclaration des strings



Plusieurs façons de déclarer un String

Des fonctions avec Strings

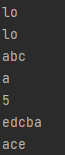
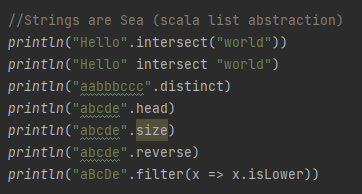


isEmpty : check si String est vide

chartAt : la position de la lettre dans String

length() : longueur String

Des fonctions avec Strings



intersect : prendre les éléments communs entre deux Strings

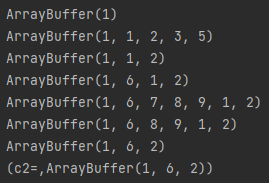
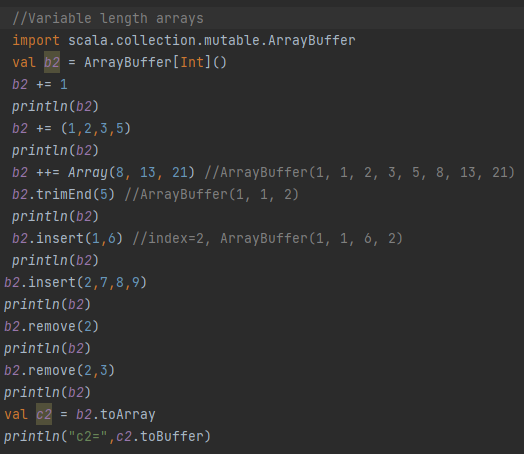
distinct : prendre des éléments disticts du String

head : la premiere lettre du String

size : la longueur du String

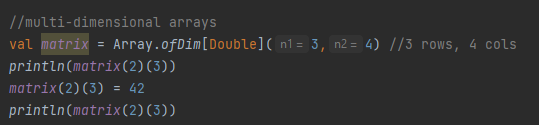
reverse : afficher le String à l’ordre inversé

Déclaration d’un Array et des fonctions avec Array



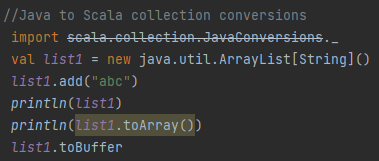
Array dans scala est homogène et modifiable, c'est-à-dire qu'il contient des éléments du même type de données et que ses éléments peuvent changer mais que la taille de Array ne peut pas changer. Pour créer une séquence mutable et indexée dont la taille peut changer, la classe ArrayBuffer est utilisée. Pour utiliser, ArrayBuffer, la classe **scala.collection.mutable.ArrayBuffer** est importée, une instance de ArrayBuffer est créée.

Array multi dimensionnel



Un array multidimensionnel est un array stockant des données au format matriciel. Nous pouvons créer un array multidimensionnel en utilisant la méthode Array.ofDim et Array of Array.

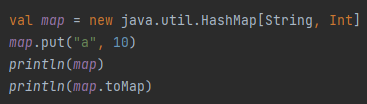
Conversion to ArrayList



ArrayList fait partie du framework de collection Java et c'est une classe du package java.util.

list.toArray : convert java list to array in scala

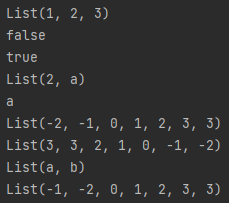
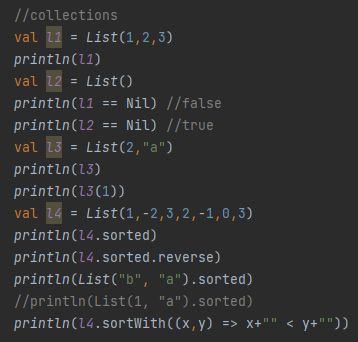
HashMap



Un HashMap est une combinaison de paires de clés et de valeurs qui sont stockées à l'aide d'une structure de données de table de hachage. Il fournit l'implémentation de base de Map. Il est utilisé pour stocker l'élément et renvoyer une carte.

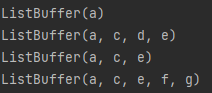
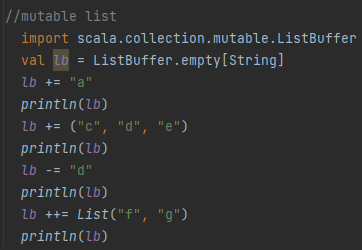
La méthode toMap() est utilisée pour retourner une carte composée de tous les éléments de l'ensemble

Operations avec List : sorted, sorted.reverse, sortWith()



L == Nil : check si la liste est nulle

Liste mutable

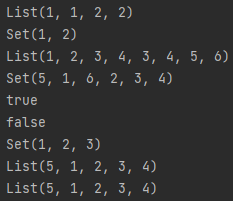
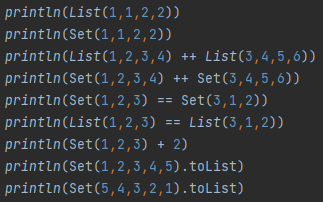


Une liste est immuable, si nous devons créer une liste qui change constamment, l'approche préférée consiste à utiliser un ListBuffer.

Pour cela, il faut importer **scala.collection.mutable.ListBuffer**

Et puis on peut concaténer cette liste avec des opérations

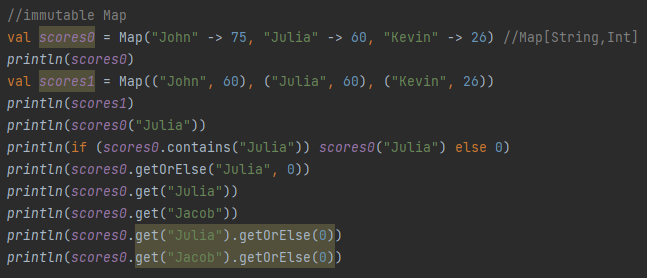
Différences entre List et Set

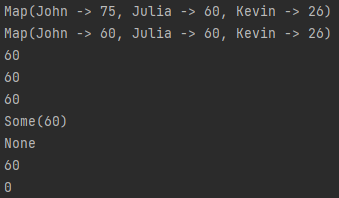


Lists sont ordonnés et peuvent avoir des éléments répétés.

Set n'est pas ordonné et ne peut pas avoir d'éléments en double

Map immutable: contains, get, getOrElse



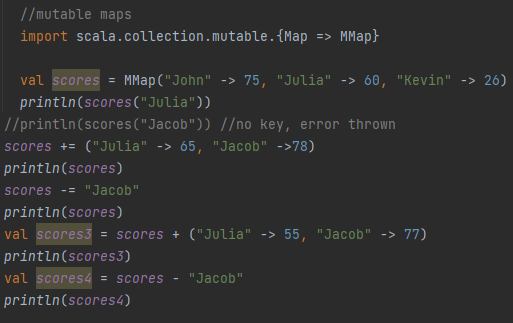


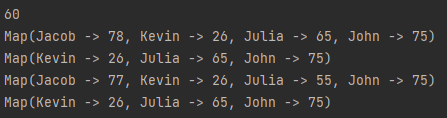
Map est une collection de paires clé-valeur. En d'autres termes, il est similaire au dictionnaire. Les clés sont toujours uniques alors que les valeurs n'ont pas besoin d'être uniques. Les paires clé-valeur peuvent avoir n'importe quel type de données. Cependant, le type de données une fois utilisé pour une clé et une valeur doit être cohérent d'un bout à l'autre. Les cartes sont classées en deux types : modifiables et immuables.

Pour les Maps immutables :

* Map(key) : Les valeurs sont accessibles à l'aide du nom et de la clé de la variable Map.
* contains(key) : Cette fonction recherche la clé dans la carte. Si la clé est présente alors elle renvoie true, false sinon
* get(key) (getOrElse( )) : Une autre approche consiste à utiliser la méthode, get ou getOrElse lors de la recherche d'une valeur. Il renvoie la valeur par défaut que vous spécifiez si la clé n'est pas trouvée :

Maps mutables





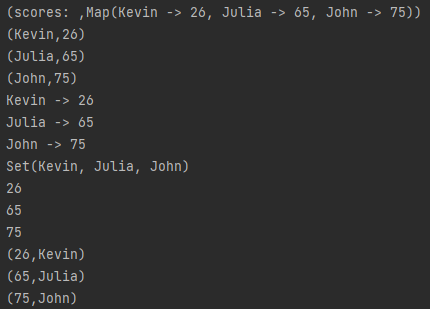
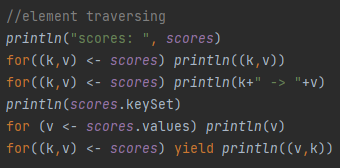
Par défaut, Scala utilise Map immuable. Pour utiliser Mutable Map, nous devons importer explicitement la classe **scala.collection.mutable.Map**.

Ajout d'une nouvelle paire clé-valeur : Nous pouvons insérer de nouvelles paires clé-valeur dans une carte mutable en utilisant l'opérateur += suivi de nouvelles paires à ajouter ou à mettre à jour.

Suppression d'une paire clé-valeur :

La suppression d'une paire clé-valeur est similaire à l'ajout d'une nouvelle entrée. La différence est qu'au lieu de +=, nous utilisons l'opérateur -= suivi des clés à supprimer.

Accéder au cle et valeur du Maps :



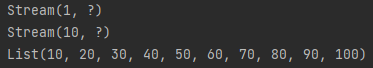
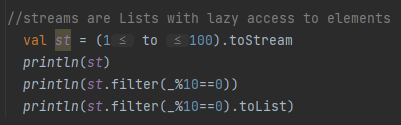
Pour chaque valeur (k,v) du Map, afficher la valeur (k,v) ou (k -> v)

Afin d'obtenir toutes les clés d'une carte Scala, nous devons utiliser soit la méthode keySet (pour obtenir toutes les clés en tant qu'ensemble)

Pour chaque valeur v du Map.values, afficher v

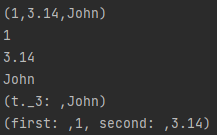
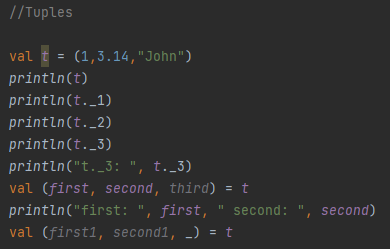
Yield est un mot-clé dans scala qui est utilisé à la fin de la boucle. C’est comme return in python, pour sauvegarder le résultat sous le même type que la collection d’origine

Stream



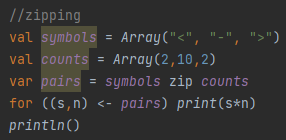
Le Stream est une **liste paresseuse** où les éléments ne sont évalués que lorsqu'ils sont nécessaires. Il s'agit d'une fonction scala. Scala prend en charge le calcul paresseux. Cela augmente les performances de notre programme. Les flux ont les mêmes caractéristiques de performances que les listes.

Tuples



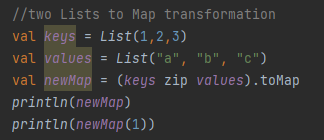
Tuple est une collection d'éléments. Les tuples sont des structures de données hétérogènes, c'est-à-dire qu'ils peuvent stocker des éléments de différents types de données. Un tuple est immuable, contrairement à un Array en scala qui est mutable.

Zipping



La méthode zip prend une autre collection comme paramètre et fusionnera ses éléments avec les éléments de la collection actuelle pour créer une nouvelle collection composée de paires ou d'éléments Tuple2 des deux collections

Transformation des List

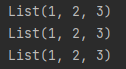
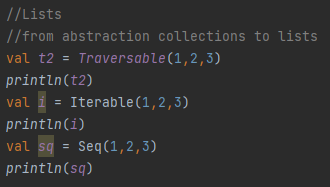


Pour convertir une list en Map dans Scala, nous utilisons la méthode toMap. Nous devons nous rappeler qu'un Map contient une paire de valeurs, c'est-à-dire une paire clé-valeur, alors qu'une liste ne contient que des valeurs uniques. Donc nous devons utiliser fonctions zip pour fusionner/connecter les 2 listes.

Voici quelques points à garder à l'esprit lors de la fusion/connection de deux listes :

* La liste que nous utilisons comme clé doit avoir des éléments uniques. Si les éléments sont uniques, le dernier élément sera pris en compte.
* Les deux listes doivent avoir un nombre égal d'éléments ; une valeur doit être présente pour chaque clé. Si ce n'est pas le cas, les éléments excédentaires de la liste seront ignorés.

Tranversable vs Iterable vs Seq

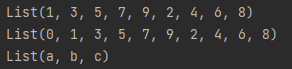
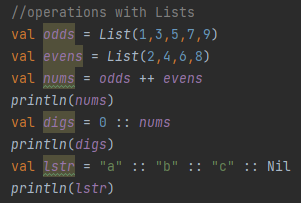


Un Traversable a une méthode abstraite : foreach. Lorsque vous appelez foreach, la collection alimente la fonction passée tous les éléments qu'elle conserve, les uns après les autres.

D'autre part, un Iterable a comme méthode abstraite iterator, qui renvoie un Iterator. Vous pouvez appeler next sur un Iterator pour obtenir l'élément suivant au moment de votre choix. Jusqu'à ce que vous le fassiez, il doit garder une trace de l'endroit où il se trouvait dans la collection et de la suite

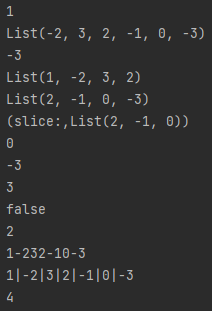
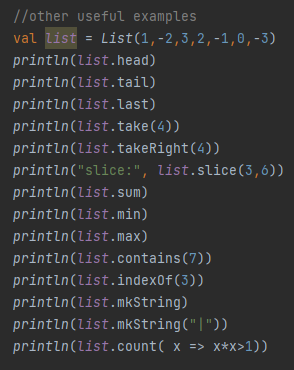
Seq est un trait pour représenter des séquences immuables. Cette structure fournit un accès basé sur un index et diverses méthodes utilitaires pour trouver des éléments, leurs occurrences et leurs sous-séquences. Un Seq maintient l'ordre d'insertion

Operations avec List



On peut utiliser ++ pour fusionner les 2 listes, :: pour ajouter un élément à la liste

D’autres fonctions avec liste

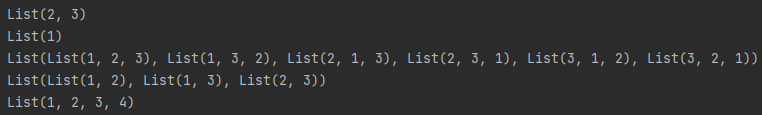
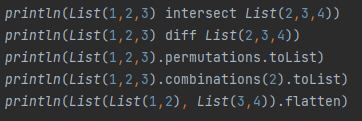


head : Le premier élément d'une liste retournée par la méthode head

tail : cette méthode renvoie une liste composée de tous les éléments sauf le premier.

…

mkString : La méthode mkString() est utilisée pour afficher tous les éléments de la liste dans une chaîne avec ou sans un séparateur



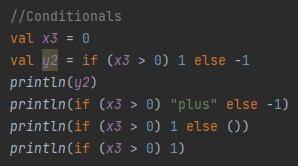
Intersect : Afficher les éléments communs de la liste avec d’autre liste

Diff : pour afficher les éléments distincts de la liste avec d’autre liste

Permutations : pour permuter les éléments de la liste

Combinations : pour combiner les éléments de la liste avec d’autre valeur ou liste

Condition

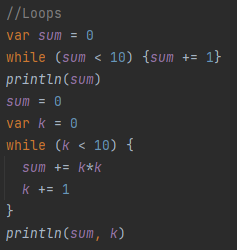


Comme beaucoup d'autres applications et langages de programmation, Scala a également une prise de décision conditionnelle if-else. Le bloc conditionnel de l'instruction if est exécuté si la condition est vraie, sinon le bloc conditionnel else est implémenté (seulement si, l'instruction else est présente)

Ici x3 > 0 y2 = 1 sinon y2 = -1

Pareillement pour la suite

Loops While

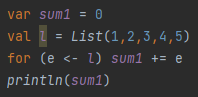


Pendant la programmation, il peut y avoir une situation que nous devons répéter jusqu'à ce qu'une condition soit remplie. Dans ces cas, la boucle while est utilisée. Une boucle while prend généralement une condition entre parenthèses. Si la condition est vraie, le code dans le corps de la boucle while est exécuté. Une boucle while est utilisée lorsque nous ne connaissons pas le nombre de fois que nous voulons que la boucle soit exécutée, mais nous connaissons la condition de terminaison de la boucle. La condition à laquelle la boucle s'arrête, est appelée condition de rupture.

Ici, pour le premier code : la programme va ajouter à sum chaque fois 1, jusqu’à ce que sum n’est plus inférieure à 10. Donc sum = 10

Pareillement, sum = 1\*1 + 2\*2 + … +10\*10 =285

Loops For

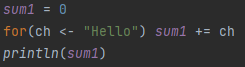


Dans Scala, la boucle for est également connue sous le nom de for-comprehensions. Une boucle for est une structure de contrôle de répétition qui nous permet d'écrire une boucle qui est exécutée un certain nombre de fois. La boucle nous permet d'effectuer n nombre d'étapes ensemble sur une seule ligne.

Chaue fois, on va ajouter à sum chaque valeur de e de la List(1,2,3,4,5),



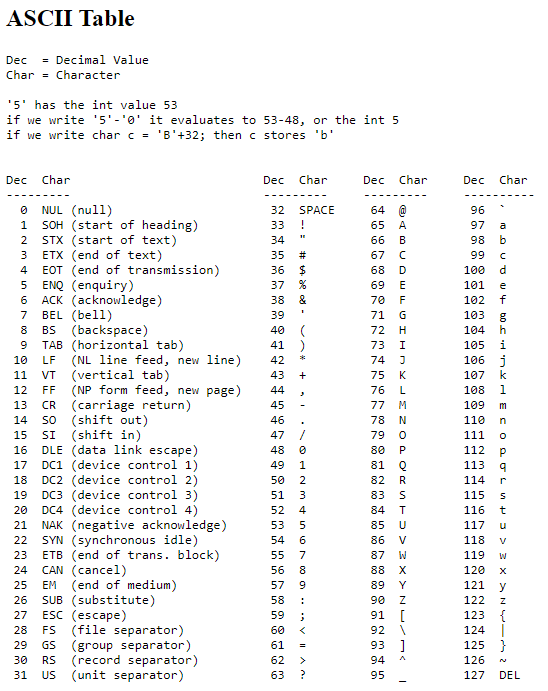
Chaque fois on va ajouter à sum une valeur de i de 1 à 10  sum =  55

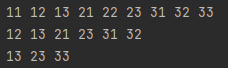
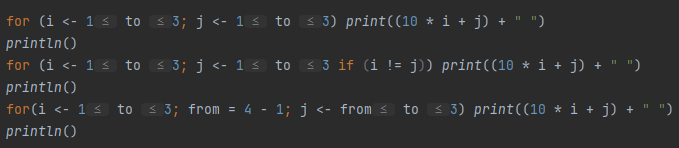


Pour chaque lettre dans « Hello », le programme va cherche la valeur dans la table ASCII :

H = 72, e = 101, l = 108, o = 111

Donc Hello = 72 + 101 + 108\*2 + 111 = 500



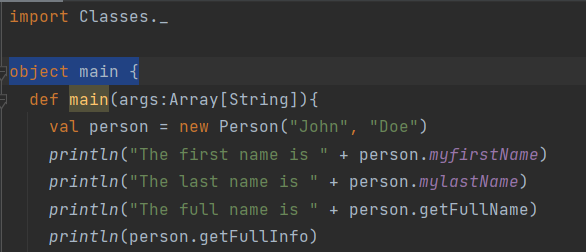


Chaque fois, i de 1 à 3, et j de 1 à 3, on va calculer 10\*i + j // 10\*1 + 1 // 10\*1 +2 // 10\*1 +3 //10\*2+1//…

Pareillement mais on calcule seulement si i différent de j

Pour la troisieme, from = 4-1 = 3, donc j accepte seulement la valeur de 3, on va faire le calcul avec i = 1, 2 , 3 avec 10 et plus j = 3

**5/Les commentaires du projet Classes version01**

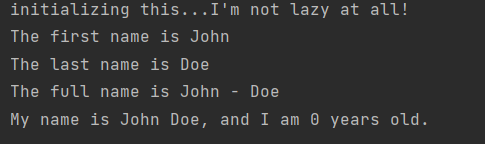
****

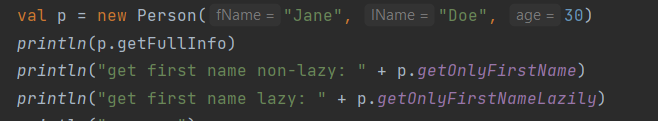
Déclarer Object main

Définir la méthode main

Main prend un paramètre d'entrée nommé args :Array[String] qui est un tableau de chaînes

**val** pour déclarer une variable immutable contrairement a **var** qui déclare une variable mutable

****

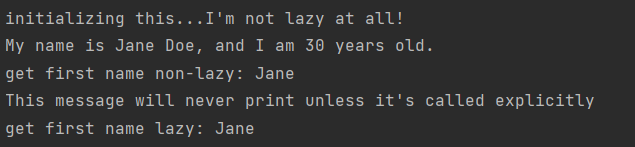
****

déclarer une variable p

Appeler la fonction getFullInfo

Appeler la fonction getOnlyFirstName

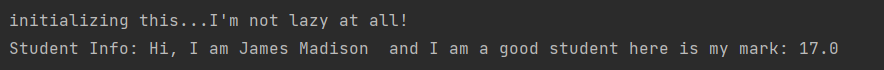
Appeler la fonction getOnlyFirstNAmeLazily

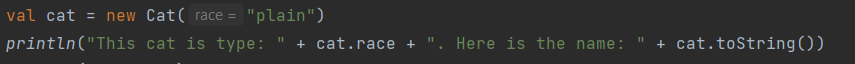
****

****

déclarer une variable student

Appeler la fonction studentInfo

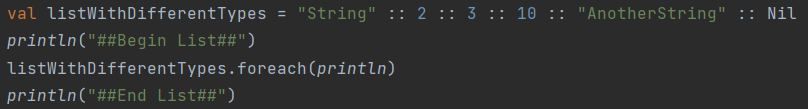
****

****

Déclarer  la variable cat

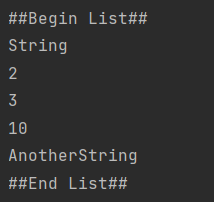
Appeler la fonction toString

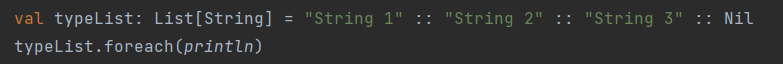
****

****

Déclarer une variable listWithDifferentTypes de type liste

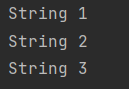
foreach pour itérer les éléments d’une liste et imprimer le contenu

****

****

Déclarer une variable typeList

foreach pour itérer les éléments d’une liste et imprimer le contenu

****

****

Définir la fonction getHead

Méthode head pour accéder au premier élément de la liste

****

**Les commentaires du projet Classes version02**

Présentation de la question 5 dans Intellij IDEA

Une image contenant texte

Description générée automatiquement

Output de l’exécution du code

Une image contenant texte

Description générée automatiquement

**Explication du code du fichier « Classes » et du fichier « main »**

Les deux fichiers sont interreliés et permettre de faire des opérations avec des « class » en utilisant les trois éléments suivants :

Def – qui sert à définir une fonction. Une fonction qui définit une méthode.

Val – qui sert à définir des valeurs fixes qui ne peuvent pas être modifiées.

Var – qui sert à définir des variables qui peuvent être modifiées.

En particulier, l’objet nommé « Classes » intègre les « class » Person, Student, Cat, WebProgrammeur. L’objet nommé « Classes » intègre aussi deux « class abstraite » Pet et Engineer.

Les « class abstraites » permettent d’afficher les fonctionnalités essentielles aux utilisateurs en masquant les implémentations et détails des traitements internes. Cela permet de constater ce que font les objets et non sur la manière dont ils le font.

Par exemple, l’interrelation entre la « class Cat » et la « class abstraite Pet » permet de simplifier le code dans le fichier « main » au moment de l’appel de l’ajout d’un élément dans la « class Cat ». À première vue, on pourrait être septique en regardant le code mais dans la réalité de la programmation cela fait du sens et devient un élément de bonne pratique de programmation.

Interrelation dans le fichier « Classes »

abstract class Pet(val name:String) {  
 override def toString = s"the pet of the family, they call me: **$**name"  
 def animalRace:String  
}  
  
class Cat(val race:String, name:String="Miaouuw") extends Pet(name) {  
 def animalRace = "Cat"  
}

Appel dans le fichier « main »

//Pet  
val cat = new Cat("plain")  
*println*("This cat is type: " + cat.race + ". Here is the name: " + cat.toString())

Cela est la même chose pour le cas de la « class WebProgrammer » et la « class abstraite Engineer».

Ensuite, on peut remarquer un élément de « lazy val » un « val paresseux » dans le code du fichier « Classes ». En particulier, cela permet au compilateur d’évaluer la variable seulement au premier accès et ne fait pas l’évaluation de l’expression du « lazy val » immédiatement. Le résultat de l’évaluation de la variable au premier accès est stocké et réutilisé lors des accès ultérieurs.

Ensuite, on remarque la présence de l’élément de « Polymorphism » dans le code du fichier « main ». Cela est un des éléments concepts importants de la programmation orientée objet. En bref, cela est la capacité des données d’être traitées sous plusieurs formes.

Les autres éléments de programmation des deux fichiers semblent relativement standard dans ce type de traitement. Les résultats obtenus de l’exécution des fichiers sont ceux que l’on s’attendait d’obtenir. Les interrelations entre les deux fichiers fonctionnent très bien.