Cheatography

Scala Cheat Sheet

by Manon Gerrd (manongerard) via cheatography.com/195339/cs/44464/

Divers

{ ... }	optionnel : (lisibilité) groupe en blocs
";" ou " "	sépare des déclarations
indentation	2 espaces
a: B	a est de type A
object A: ... def main(args: Array[String]): Unit = ... ou @main def a(...) =	main
private def/val ..	méthode/variable privées
sealed class/trait	peut être étendu seulement dans le même fichier
type x = ...	alias ou synonyme de type

Chaque valeur est un objet
Chaque objet peut avoir 1 ou plus de membres
Module: objet dont le but principal est de donner à ses membres un espace de noms

Variables

val x = 1	variables immuables /!\ éviter var

Class

class C ...	classe
var c = new C(...)	crée un nouvel objet
object O ...	singleton = classe avec 1 seule instance
case class C ...	case classes sont des classes spéciales qui ont : • Une liste de paramètres qui agit comme constructeur. • Ne nécessitent pas le mot-clé new. • Tous les paramètres sont immuables et **publics**. • Les instances sont comparées par structure et non par référence.
abstract class C ...	classe abstraite

Class (cont)

class C extends D ...	classe héritée

"this" fait références à l'object
pour accéder a des "champs": this.champ ou c.champ

Méthodes

def a(b: C) *[: D]* = ...	méthode *A* avec comme argument *b* de type *C* qui retourne un objet de type *D* signature = définition
def a*[*T, U*]*(b: T, c: U) *[: D]* = ...	méthode polymorphique (= avec paramètre(s) de type)
*[*T <: U*]*	Borne supérieure T doit être un sous-type de U
*[*T >: U*]*	Borne inférieure T doit être un super-type de U
def a(b: C*) = ...	fonctions variadiques : 0 ou plusieurs arguments b: C* = b est un objet séquence (Seq) d'éléments de type C
(x, y) => x + y ou _ + _ ou ...	fonctions anonymes ou littérales = sans nom
@annotation.tailrec def ...	optimise une fonction récursive terminale

\textcolor{red}{\warning} Scala accepte toute définition de fonction récursive syntaxiquement correcte, même si la fonction ne peut jamais se terminer. C'est au développeur de savoir si, pour un domaine donné, le calcul se terminera toujours.

\textcolor{red}{\warning} ne pas utiliser "\texttt{return}" : la valeur de la dernière instruction est renvoyée
le type de retour n'est pas obligatoire si assez clair, mais le prof le préfère
les fonctions sont typés donc ça permet d'avoir des fonctions d'ordre supérieur

Fonctions d'ordre supérieur

```
def partial[A, B, C](a: A, f: (A, B) => C): B => C =
(b) => f(a, b)
val plus = (x: Int, y:Int) => x + y
val plus1 = partial(1, plus)
plus1(5) // returns 6
```

fonction prenant en argument une autre fonction ou renvoyant une fonction comme résultat

Currying

```
def mul(x: Int, y: Int): Int = x * y
def curriedMul(x: Int)(y: Int): Int = x * y
```

convertir une fonction prenant en entrée plusieurs arguments en une séquence de fonctions prenant chacune un seul argument

Comparaison

a equals b a.equals(b)	égalité par rapport à la valeur erreur si null
a == b	dépend de equals de *a* gère correctement la valeur null
a eq b	égalité des références

Stratégies d'évaluation

val a = 1	évaluée lorsqu'elle est définie
lazy val a = 1	Appel par nécessité: 1 seule fois quand utilisé puis mémoïsé
def method(a: Int, b: => Int) ...	fonction: évalué quand elle est appelée a: appel par valeur: évalué lors d'un appel et résultats sont copiés b: Appel par nom: évalué à chaque utilisation

Méthodes importantes

foldRightB(f: (A, B) => B): B	Applique f à chaque élément de droite à gauche en commençant avec z
foldLeftB(f: (B, A) => B): B	Applique f à chaque élément de gauche à droite en commençant avec z

Méthodes importantes (cont)

take(n: Int): ...[A]	Sélectionne les n premiers éléments
takeWhile	
forall	
exists	
scanLeft	
scanRight	
mapB: ...[B]	Crée un ... en appliquant f à chaque élément
flatMapB: ...[B]	Crée un ... en appliquant f à chaque élément et en utilisant les éléments des collections résultantes
groupMap	
apply	
filter(p: A => Boolean): ...[A]	Sélectionne tous les éléments qui satisfont un prédicat

Gestion des exceptions

Option[+A]	Le type de retour reflète la possibilité qu'un résultat ne soit pas défini
Some(A)	Résultat défini
None	Résultat indéfini
getOrElse*[B>:A]*(default: => B): B	Renvoie la valeur du Some, sinon defaut
orElse*[B>:A]*(op: => Option[B]): Option[B]	Renvoie le Some, sinon op
Either*[+E, +A]*	Permet de stocker une valeur pour les exceptions
Left*[+E]*(value: E)	erreur/exception
Right*[+A]*(value: A)	correcte/succès
getOrElseB >: A: B	Renvoie la valeur de Right, sinon defaut
orElseEE >: E, AA >: A: Either[EE, AA]	Renvoie le Right, sinon op

Scala Cheat Sheet

by Manon Gerrd (manongerard) via cheatography.com/195339/cs/44464/

Collection immuable

(1, a)	Tuple éléments peuvent être de différent type
a._nb a(nb)	accéder à un élément en fct de l'index (._ commence à 1, et () à 0)
val l: List[Int] = List(1, 2) val l: List[(Int, String)] = List(...)	Liste les éléments doivent être du "type" mis entre []
Nil equivalent à List[Nothing]	liste vide
1 :: 2 :: 3 :: Nil equivalent à List(1, 2, 3)	cons operator head :: tail (list)
val lz = LazyList(1, "a")	Liste paresseuse
1 #:: (1+1) #:: lz	cons qui retarder le calcul
List.fill(n)(method(a))	crée une liste de *n* items contenant le résultat de *n* évaluations de *method(a)*
l.unzip	divise une liste de paires en une paire de listes
l.reduce((a, b) => a.anonMethod(b))	réduire les éléments d'une collection à 1élément Nécessite une fonction qui implémente une opération binaire (commutative et associative)
l.groupBy(_.champ)	crée un dictionnaire des champ (clés) vers une liste du type de *l*
val m = Map("a" -> 1, "b" -> 2)	dictionnary qui map des clés a des valeurs
m.values	itérable contenant chaque valeur associée à une clé
m.keys	itérable contenant chaque clé
c.toList	transforme une collection en une liste
val s = Seq ???	séquence = liste ordonnée

Collection immuable (cont)

l.range(1,10[,1])	crée une collection allant de start à end (non compris) avec éventuellement un saut

Nothing est un sous-type de toutes les classes => Nil équivalent à List[Int], List[String], ...

Type

Unit	tuple vide/void en tant qu'objet seule valeur : ()
Boolean	true, false

Int/Double/String

Ne pas utiliser la classe StringBuilder car elle n'est pas référentiellement transparente

Sucre syntactique

sucre syntactique	équivalent à
_.champ	(x) => x.champ
_ method/opérateur _	(x, y) => x method y (x, y) => x.method(y)
List[?]	Liste de n'importe quel type wildward argument
s*	opérateur splat : adapte une séquence pour qu'elle puisse être utilisée comme un nombre variable d'arguments

L'ordre des arguments est important, il n'y a pas moyen de l'inverser

Import/packages

import package.method import package.{method1, method2}	Importe des méthode(s) venant du package
import package.*	Importe toutes les méthodes venant du package

Quand une méthode est importée, elle peut être utilisée de manière non qualifiée, sans préciser le package d'où elle vient.

Cheatography

Scala Cheat Sheet

by Manon Gerrd (manongerard) via cheatography.com/195339/cs/44464/

Comments

// single line comment

/* Multiline
comment*/

/** Documentation
*comment
*/

Paramètres de type

def a[T, U]...	2 paramètres de types T et U
def a[T <: U]...	Borne supérieure T doit être un sous-type de U
def a[T :> U]...	Borne inférieure T doit être un super-type de U

Variance

List[+T]	covariant	List[S] est le sous-type de List[T]
List[-T]	contravariant	List[T] est le sous-type de List[S]
List[T]	Invariant par défaut	List[T] et List[S] ne sont pas liés

Supposons que nous ayons une classe générique qui a un paramètre de type T. Nous avons 2 types de liste, List[L] et List[S] où S est un sous-type de T. Grâce à cela, nous pouvons définir les 3 types de variances pour le paramètre T de la liste :

Structure de contrôle

Structure de contrôle (cont)

filtrage par motif
cible match cas
possibilité d'ajouter un if dans le case = garde de motif

peut ajouter () autour des conditions/gardes

Type algébrique de données

enum C[+A]: case D(...) case E(...)	déclarer un type de données avec un ou plusieurs constructeurs de données Doit être importé
object C: ...	object compagnon = object avec le même nom qu'un type de données déclaré avec enum
trait C[+A]: interfaces et champs partagés	déclarer un type de données avec des interfaces et champs partagés
case class D(...) extends C[A] ...	on peut avoir accès a leurs champs

?

object T: extension (t: T[String])...	étend l'objet ou la classe avec des champs supplémentaires, tels que des fonctions.
object T: given StringT: T[String] with ...	instance given
object T: def fA(using m: T[A]) ...	paramètre de contexte : transmis "automatiquement" Scala cherche une instance given pour chacun

Si un paramètre de contexte peut être ambigu, il faudra préciser lequel utiliser

if/then/else

if cond then expr1
else if cond2 then expr2
else expr3

if cond expr1
équivalent à if cond expr1
else ()

can add an end if at the end of each expression

boucle **for** : avec des effets de bord

for generator(s) do expr

generateur: p <- e où e est une collection, ou qqch to qqch \textit{[by qqch]}\newline peut avoir un if dedans (= garde)\newline ou (clé, valeur) <- m où m est une map

for expressions: retourne une valeur

for generateur(s)
yield expr
équivalent à val list = (x).map(p => expr)
si un garde est présent, replacer x par x.filter(IFexpr)

boucle **while**

while cond do expr

try/catch/finally

try expr
catch
case ...
finally

match expressions

x match
case 0 => "zero"
case _ => "autre" // attrape tout

By **Manon Gerrd**
(manongerard)
cheatography.com/manongerard/

Not published yet.
Last updated 30th November, 2024.