

PRG1 (9): 多相型

脇田建

2016.10.31

findFirstString

```
def findFirstString(ss: Array[String], key: String): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

- ❖ findFirstString: 配列 (ss: Array[String]) のなかから、検索キーの文字列 (key: String) に合致する最初のデータの位置を返す。見つからない場合は -1 を返す。

findFirstNumber

```
def findFirstNumber(ss: Array[Int], key: Int): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

- ❖ findFirstNumber: 配列 (ss: Array[Int]) のなかから、検索キーの数 (key: Int) に合致する最初のデータの位置を返す。見つからない場合は -1 を返す。

findFirst{String vs Number}

```
def findFirstString(ss: Array[String], key: String): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

```
def findFirstNumber(ss: Array[Int], key: Int): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```


findFirst{String vs Number}

```
def findFirstString(ss: Array[String], key: String): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

```
def findFirstNumber(ss: Array[Int], key: Int): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

findFirstNumber vs findFirst

```
def findFirstNumber(ss: Array[Int], key: Int): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```

```
def findFirst[A](ss: Array[A], key: Int): Int = {  
  def loop(n: Int): Int =  
    if (n >= ss.length) -1  
    else if (ss(n) == key) n  
    else loop(n + 1)  
  
  loop(0)  
}
```


型変数 A

- ❖ 型変数の導入: `def findFirst[A] ...`
 - ❖ 新たな型変数 A を導入する。その変数 A の有効範囲は `def` の範囲。
 - ❖ 型変数 A の意味：「ある型があって、その名前をひとまず A としておこう」、(簡単に)「任意の型Aについて」
- ❖ 型構成子の型変数への適用: `Array[A]`
 - ❖ `Array[A]`: 任意の型 A に関して、型構成子 `Array` を型Aについて特殊化したもの。
 - ❖ `Array`の場合 A は`Array` が表す配列が要素とするデータの型なので、`Array[A]` は、A型のデータを要素とする配列の型と読める。

class Array[T]が提供する単相関数

- ❖ 型変数に依存しない関数群
- ❖ def isEmpty: Boolean
- ❖ def length: Int
- ❖ def size: Int

class Array[T]が提供する多相関数

- ❖ 出現する型変数がTだけで、返り値の型が単純なもの
- ❖ `def indexOf(T): Int`
- ❖ `def forall((T) ⇒ Boolean): Boolean` // \forall
- ❖ `exists((T) ⇒ Boolean): Boolean` // \exists
- ❖ `def indexOf(T): Int` // `Array(1, 2, 3).indexOf(3) => 2`
- ❖ `def count(p: (T) ⇒ Boolean): Int`
// `Range(1,..., 99).toArray.count(奇数) => 50`

class Array[T]が提供する多相関数

- ❖ 出現する型変数がTだけで、返り値の型がTを含むもの
- ❖ def head: T
- ❖ def last: T
- ❖ def init: Array[T] // Array(V^n , v) => Array(V^n)
- ❖ def tail: Array[T] // Array(v, V^n) => Array(V^n)
- ❖ def take(Int): Array[T] // Array(V^k , v, ...) => Array(V^n)
- ❖ def drop(Int): Array[T] // Array(V^k , v, ...) => Array(v, ...)

class Array[T]が提供する多相関数

- ❖ 型変数Tが引数にも返り値にも出現するもの
- ❖ `def filter((T) \Rightarrow Boolean): Array[T]`
// `Array(1, 2, 3, 4, 5).filter(奇数) => Array(1, 3, 5)`
// `Array(1, 2, 3, 4, 5).filter((n: Int) => n%2==1)`
- ❖ `def foldLeftB ((B, T) \Rightarrow B): B`

trait Set[T] が提供する関数

- ❖ isEmpty: Boolean
- ❖ empty: Set[A]
- ❖ contains(A): Boolean
- ❖ diff(GenSet[A]): Set[A]
- ❖ union(GenSet[A]): Set[A]
- ❖ map[B]((A) \Rightarrow B): Set[B] // Array(1, 2, 3).map((x: Int) => x.toString)
- ❖ subsets(): Iterator[Set[A]] // Set(1, 2, 3).subsets().foreach(println)

Scala の API マニュアル

- ❖ Macユーザへのお薦め: Dash の利用 (強力な検索能力)
- ❖ そうではない人は、本家のドキュメントをダウンロードして利用。
- ❖ <http://scala-lang.org/download/all.html> を開き、自分が利用している Scala のバージョン(2.11.8)のページを開き、API DocsのZipファイルをダウンロードしたあとで展開して利用する。

package/class/trait
の名前で検索

O, C をクリック

O: Companion Object

C: Class

scala-lang.org

scala api - Google 検索

Scala Standard Library

root package

package root

This is the documentation for the Scala standard library.

Package structure

The [scala](#) package contains core types like [Int](#), [Float](#), [Array](#), and all Scala compilation units without explicit qualification or import.

Notable packages include:

- [scala.collection](#) and its sub-packages contain Scala's core collections:
 - [scala.collection.immutable](#) - Immutable, sequential collections like [Vector](#), [List](#), [Range](#), [HashMap](#) or [HashSet](#)
 - [scala.collection.mutable](#) - Mutable, sequential collections like [ArrayBuffer](#), [StringBuilder](#), [HashMap](#) or [HashSet](#)
 - [scala.collection.concurrent](#) - Mutable, concurrent collections like [TrieMap](#)
 - [scala.collection.parallel.immutable](#) - Immutable, parallel collections like [ParVector](#), [ParRange](#), [ParHashMap](#) or [ParHashSet](#)
 - [scala.collection.parallel.mutable](#) - Mutable, parallel collections like [ParArray](#), [ParHashMap](#), [ParTrieMap](#) or [ParHashSet](#)
- [scala.concurrent](#) - Primitives for concurrent programming

クラスとCompanionオブジェクト (連れ合いのオブジェクト)

- ❖ class Int vs object Int
- ❖ class Array vs object Array
- ❖ class List vs object List
- ❖ trait Set vs object Set

class Int vs object Int

- ❖ 算術演算子: $+$, $-$, $*$, $/$
- ❖ 比較演算子: $>$, $<$, $==$
- ❖ ビット毎演算子
- ❖ `min`, `max`, `signum`
- ❖ `class Int`に関する情報
- ❖ `MaxValue`, `MinValue`
- ❖ `toString`

class Array vs object Array

- ❖ class Array についてすでに取り上げたので省略
- ❖ object Array
 - ❖ empty[T]: Array[T]
 - ❖ emptyIntArray: Array[Int]
 - ❖ fill[T](Int)(\Rightarrow T): Array[T]
// Array.fill(5)(3), Array.fill(100)(math.random)
 - ❖ ofDim[T](Int, Int): Array[Array[T]] // Array.ofDim[Int](3, 4)
 - ❖ tabulate[T](T, Int)((T) \Rightarrow T): Array[T]

object List

- ❖ `empty[A]: List[A]`
- ❖ `iterate[A](A, int)((A) => A) //`
`List.iterate(List.empty[Int], 4)((l: List[Int]) => 0::l)`
- ❖ `range[T](T, T) // List.range(0, 20, 3)`