



UNIVERSIDAD  
COMPLUTENSE  
DE MADRID



**ntic**  
master  
**School**

# Scala

Valores Options

Noviembre de 2023



# 1 Valores Option



# Option



```
scala> val languages = Map("s"->"Scala", "p"->"Python")
languages: scala.collection.immutable.Map[String,String] = Map(s -> Scala, p -> Python)

scala> languages.get("s")
res11: Option[String] = Some(Scala)

scala> languages.get("c")
res12: Option[String] = None
```

- Valores Option mecanismo de control para evitar acceso a valores vacíos.
- Este mecanismo es provisto por la librería Standard.
- Es un [ADT](#) que nos ofrece dos posibles tipos:
  - *Some*: case class como envoltura de un valor.
  - *None*: singleton object.



## Instanciar Option

```
scala> Some("Scala")
res13: Some[String] = Some(Scala)

scala> Option(null)
res14: Option[Null] = None

scala> Option("Scala")
res15: Option[String] = Some(Scala)
```

- Se usa *Some* y *None* directamente.
- Se usa el método *apply* del companion object *Option* para envolver los valores, vacíos (*null*) y obtener un *None*, distintos a vacíos para *Some*.

## Pattern Matching y Option

```
def languageByFirst(s: String): String =  
  languages.get(s) match {  
    case Some(language) => language  
    case None => s"No language starting with $s!"  
  }
```

```
scala> languageByFirst("s")  
res16: String = Scala  
  
scala> languageByFirst("c")  
res17: String = No language starting with c!
```

- Al implementar un *Option* en un match (Pattern Matching), el sistema de tipado nos obliga a manejar los valores opcional correctamente.
- El compilador dará error si no se contemplan ambos casos: *Some* y *None*.

# Funciones de orden superior y Option



```
scala> Option("Scala").map(_.reverse)
res20: Option[String] = Some(alacS)

scala> for {
  | language <- Some("Scala")
  | behavior <- Some("mola")
  | } yield s"$language $behavior"
res21: Option[String] = Some(Scala mola)
```

- Option ofrece las mismas funciones de orden superior que las colecciones.
- El uso de las funciones de orden superior (map y flatMap) sobre Option permite anidar llamadas que gestionan los valores perdidos fácilmente y con seguridad.



## foreach y Option



```
scala> val languages = Map("s" -> "Scala", "p" -> "Python")
languages: scala.collection.immutable.Map[String,String] = Map(s -> Scala, p -> Python)

scala> languages.get("s").foreach(println)
Scala

scala> languages.get("c").foreach(println)

scala> █
```

- Las llamadas a foreach invocan a la función solo si hay un valor presente.
- Igual que con las colecciones, foreach ejecuta la función para efectos secundarios (ejemplo, escribir por consola), si no hay valor devuelto se descarta la invocación.



# Extracción del valor



```
def languageByFirst(s: String): String =  
  languages.get(s).getOrElse(s"No language starting with $s")
```

- *getOrElse* extrae el valor que contenga o devuelve un valor por defecto.





## Ejercicio: Usa Option



- Añade el método `timeAt` a `Train`:
  - Añade un parámetro de tipo `Station`
  - devuelve un `Option` de `Time`:
    - `Some` de `Time` si el `Train` se detiene en la estación,
    - si no, devuelve `None`

