VAL VS DEF FUNCTIONS

Val function (Lambda)

```
val replicate: (Int, String) => String =
  (n: Int, text: String) => ...
```

```
replicate(3, "Hello ")
// res1: String = "Hello Hello Hello "
```

Def function (Method)

```
def replicate(n: Int, text: String): String =
  ...
```

```
replicate(3, "Hello ")
// res3: String = "Hello Hello Hello "
```

Val function (Lambda or anonymous function)

```
(n: Int, text: String) => List.fill(n)(text).mkString
```

Val functions are ordinary objects

```
(n: Int, text: String) => List.fill(n)(text).mkString
```

```
3

"Hello World!"

User("John Doe", 27)
```

Val functions are ordinary objects

```
val replicate = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val counter = 3

val message = "Hello World!"

val john    = User("John Doe", 27)
```

Val functions are ordinary objects

```
val replicate = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val counter = 3

val message = "Hello World!"

val john    = User("John Doe", 27)
```

```
val repeat = replicate
```

Val functions are ordinary objects

```
val numbers   = List(1,2,3)
// numbers: List[Int] = List(1, 2, 3)

val functions = List((x: Int) => x + 1, (x: Int) => x - 1, (x: Int) => x * 2)
// functions: List[Int => Int] = List(<function1>, <function1>, <function1>)
```

Val functions are ordinary objects

```
val numbers   = List(1,2,3)
// numbers: List[Int] = List(1, 2, 3)

val functions = List((x: Int) => x + 1, (x: Int) => x - 1, (x: Int) => x * 2)
// functions: List[Int => Int] = List(<function1>, <function1>, <function1>)
```

```
functions(0)(10)
// res12: Int = 11

functions(2)(10)
// res13: Int = 20
```

Val function desugared

```
val replicate: (Int, String) => String = (n: Int, text: String) => List.fill(n)(text).mkString
```

Val function desugared

```
val replicate: (Int, String) => String        = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val replicate: Function2[Int, String, String] = (n: Int, text: String) => List.fill(n)(text).mkString
```

Val function desugared

```
val replicate: (Int, String) => String        = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val replicate: Function2[Int, String, String] = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val bigFunction: (Int, Int, Int, Int, Int,
                  Int, Int, Int, Int, Int,
                  Int, Int, Int, Int, Int,
                  Int, Int, Int, Int, Int,
                  Int, Int, Int
                 ) => String = _ => ???

// error: function values may not have more
//        than 22 parameters, but 23 given
```

Val function desugared

```
val replicate: (Int, String) => String        = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val replicate: Function2[Int, String, String] = new Function2[Int, String, String] {
  def apply(n: Int, text: String): String =
     List.fill(n)(text).mkString
}
```

SAM (Single Abstract Method)

```
trait Printer {
  def print(message: String): Unit
}
```

SAM syntax

```
val console: Printer =
  (message: String) => println(message)
```

Standard syntax

```
val console: Printer = new Printer {
  def print(message: String): Unit =
    println(message)
}
```

SAM (Single Abstract Method)

```
trait Printer {
  def print(message: String): Unit
}
```

Standard syntax

```
val console: Printer = new Printer {
  def print(message: String): Unit =
    println(message)
}
```

SAM syntax

```
val console: Printer =
  (message: String) => println(message)
```

Val function desugared

SAM syntax

```
val replicate: (Int, String) => String =
  (n: Int, text: String) =>
    List.fill(n)(text).mkString
```

Standard syntax

```
val replicate: (Int, String) => String =
  new Function2[Int, String, String] {
    def apply(n: Int, text: String): String =
       List.fill(n)(text).mkString
  }
```

Val function desugared

SAM syntax

```
val replicate: (Int, String) => String =
  (n: Int, text: String) =>
    List.fill(n)(text).mkString
```

```
replicate(3, "Hello ")
// res24: String = "Hello Hello Hello "

replicate.apply(3, "Hello ")
// res25: String = "Hello Hello Hello "
```

Standard syntax

```
val replicate: (Int, String) => String =
  new Function2[Int, String, String] {
    def apply(n: Int, text: String): String =
       List.fill(n)(text).mkString
  }
```

Apply syntax

```scala
trait Printer {
  def apply(message: String): Unit
}

val console: Printer = new Printer {
  def apply(message: String) = println(message)
}
```

```scala
console("Hello World!")
// Hello World!
```

Def function (Method)

```
import java.time.LocalDate

def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
createDate(2020, 1, 5)
// res28: LocalDate = 2020-01-05
```

Arguments

```
import java.time.LocalDate

def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
val createDateVal: (Int, Int, Int) => LocalDate =
  (year, month, dayOfMonth) => ...
```

IDE

```
createDate
m  createDate(year: Int, month: Int, dayOfMonth: Int)        LocalDate
v  createDateVal                          (Int, Int, Int) => LocalDate
^↓ and ^↑ will move caret down and up in the editor  Next Tip
```

Scaladoc

```
def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate
```

```
val createDateVal: (Int, Int, Int) => LocalDate
```

Named arguments

```
import java.time.LocalDate

def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
createDate(2020, 1, 5)
// res30: LocalDate = 2020-01-05
```

```
createDate(dayOfMonth = 5, month = 1, year = 2020)
// res31: LocalDate = 2020-01-05
```

Def functions are not data

```
def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
List(createDate)
// error: missing argument list for method createDate in class App16
// Unapplied methods are only converted to functions when a function type is expected.
// You can make this conversion explicit by writing `createDate _` or `createDate(_,_,_)` inste
// List(createDate)
//      ^^^^^^^^^^
```

Def functions are not data

```
def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
List(createDate _)
// res33: List[(Int, Int, Int) => LocalDate] = List(<function3>)
```

Def functions are not data

```
def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
List(createDate _)
// res33: List[(Int, Int, Int) => LocalDate] = List(<function3>)
```

```
val createDateVal = createDate _
// createDateVal: (Int, Int, Int) => LocalDate = <function3>
```

Def functions are not data

```
def createDate(year: Int, month: Int, dayOfMonth: Int): LocalDate =
  ...
```

```
List(createDate): List[(Int, Int, Int) => LocalDate]
```

```
val createDateVal: (Int, Int, Int) => LocalDate = createDate
```

Summary

- Val functions are ordinary objects
- Use def functions for API
- Easy to convert def to val