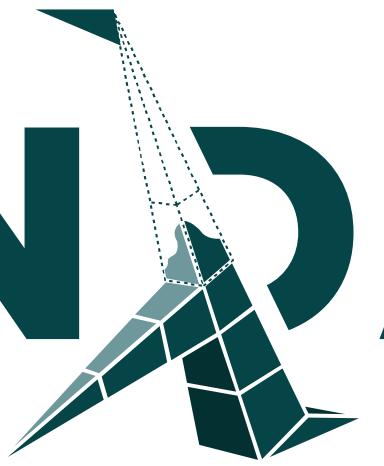
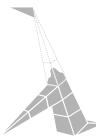


# FOUNDATION



# Val vs Def functions



# Functions

# Val function (Lambda)

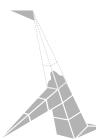
```
val replicate: (Int, String) => String =  
(n: Int, text: String) => ...
```

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

# Defunction (Method)

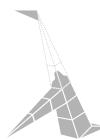
```
def replicate(n: Int, text: String): String  
{  
    ...  
}
```

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



# Val function (Lambda or anonymous function)

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



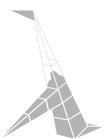
# Val functions are ordinary objects

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

```
3
```

```
"Hello World!"
```

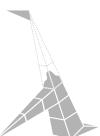
```
User("John Doe", 27)
```



# 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

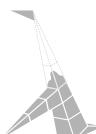
```
val replicate = (n: Int, text: String) => List.fill(n)(text).mkString
```

```
val counter = 3

val message = "Hello World!"

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

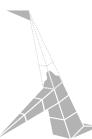
```
val repeat = replicate
```



# 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>)
```

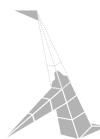
```
functions(0)(10)
// res12: Int = 11

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



# Val function desugared

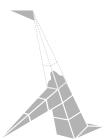
```
val replicate: (Int, String) => String = (n: Int, text: String) => List.fill(n)(text).mkString
```



# Val function desugared

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

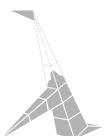
```
val replicate: Function2[Int, String, String] = (n: Int, text: String) => List.fill(n)(text).mk
```



# Val function desugared

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

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

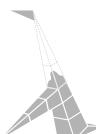


# Val function desugared

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

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

```
replicate.apply(3, "Hello ")  
// res19: String = "Hello Hello Hello "
```



# 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  
}
```

```
replicate.apply(3, "Hello ")  
// res19: String = "Hello Hello Hello "
```

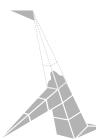
```
replicate(3, "Hello ")  
// res20: String = "Hello Hello Hello "
```



# Def function (Method)

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

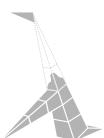
```
createDate(2020, 1, 5)  
// res21: LocalDate = 2020-01-05
```



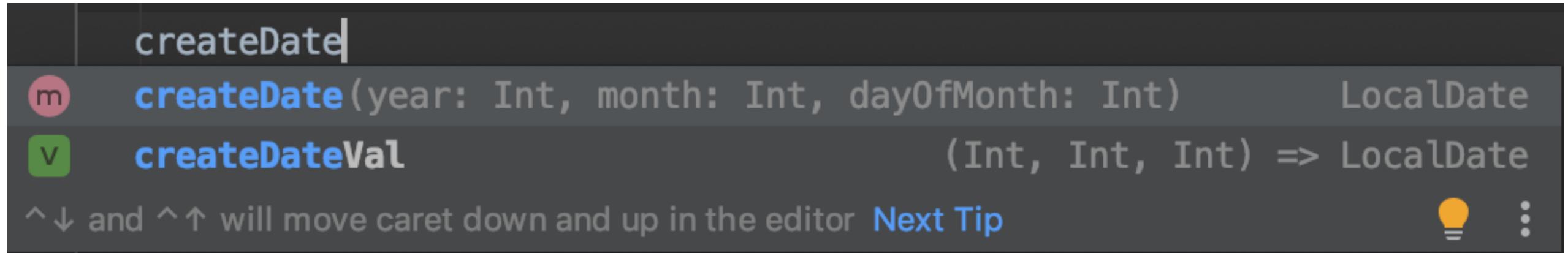
# Function arguments

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

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

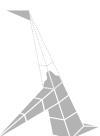


# IDE



# Javadoc

```
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)  
// res23: LocalDate = 2020-01-05
```

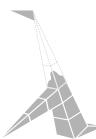
```
createDate(dayOfMonth = 5, month = 1, year = 2020)  
// res24: LocalDate = 2020-01-05
```



# Def functions are not data

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

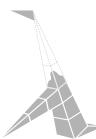
```
List(createDate)  
// error: missing argument list for method createDate in class App10  
// Unapplied methods are only converted to functions when a function type is expected.  
// You can make this conversion explicit by writing `createDate _` or `createDate(_,_,_)` instead  
// List(createDate)  
//          ^^^^^^^^^^
```



# Def functions are not data

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

```
List(createDate _)  
// res26: List[Int, Int, Int] => LocalDate] = List(<function3>)
```

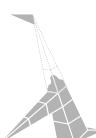


# Def functions are not data

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

```
List(createDate _)  
// res26: List[Int, Int, Int] => LocalDate] = List(<function3>)
```

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

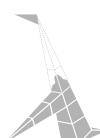


# Def functions are not data

```
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 an ordinary objects
- Use def functions for API
- Easy to convert def to val

