Higher Order Functions Scala vs Python

By Biswas Gauli, Freddy Perez, Ashley Weaver

What are Higher Order Functions

- Higher Order functions are function that take other functions as parameters or if the function returns another function
- Basically functions that use other functions inside them are called Higher Order Functions
- In its core higher order functions are functions that use the principle: Functions are Values
- They are used in functional programming rather than imperative functions
- Many languages support higher order functions, for example: Scala, Python, Javascript, Go, etc.

Properties of Higher Order Functions

- A function is an instance of the Object type.
- You can store the function in a variable.
- You can pass the function as a parameter to another function.
- You can return the function from a function.
- You can store them in data structures such as hash tables, lists,

Example

Here is a simple example of a higher order function

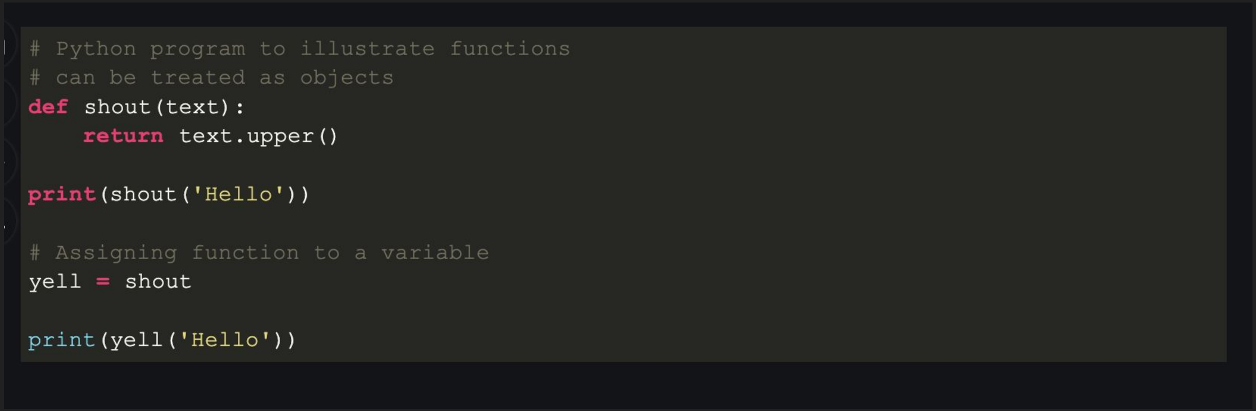
```scala
def arith(x: Int, y: Int, f: (Int, Int) => Int): Int = f(x,y)

def main(args: Array[String]) {
    val result = arith(10, 2, (x,y) => x*y)
    val result2 = arith(10, 2, (x,y) => x+y)
    println(result) //returns 20
    println(result2) //returns 12
}
```

Higher order methods example

Here is an example of using higher order methods

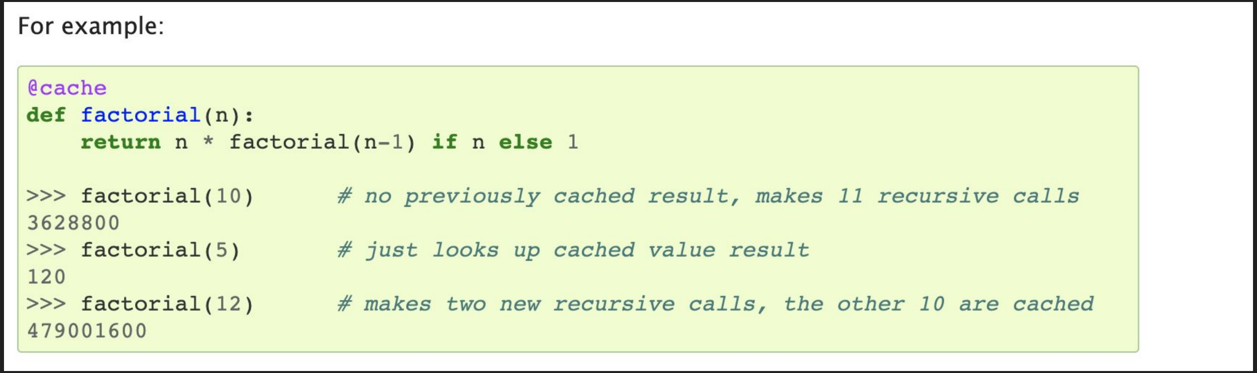
```
Val mylist = List(1,2,3,4,5,6)
def main(args: Array[String]) {
   println(mylist.map(x => x*2))
   //returns List[2,4,6,8,10,12]
   println(mylist.map(x => x + "foo"))
   //returns List(1foo,2foo,3foo,4foo,5foo,6foo)
}
```

Python Higher Order Function



```
# Python program to illustrate functions
# can be treated as objects
def shout(text):
    return text.upper()

print(shout('Hello'))

# Assigning function to a variable
yell = shout

print(yell('Hello'))
```

Python Higher Order Function

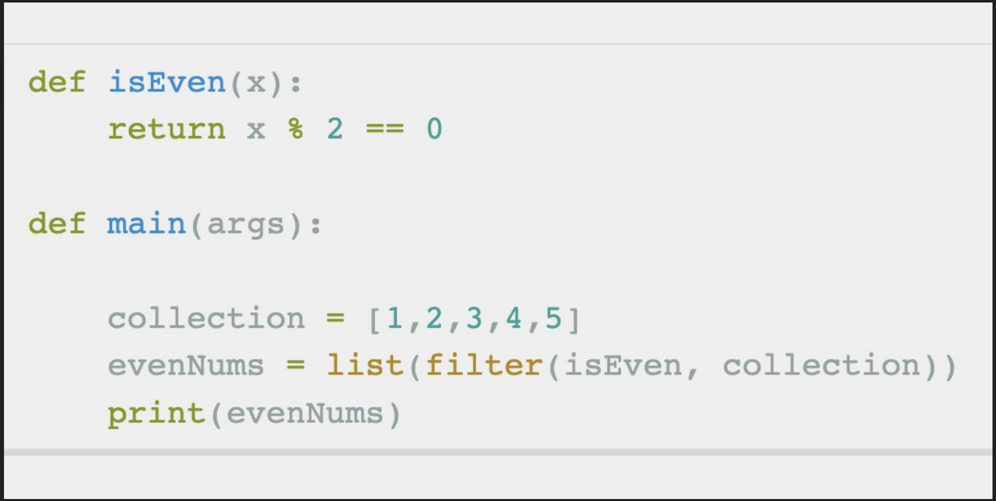


For example:

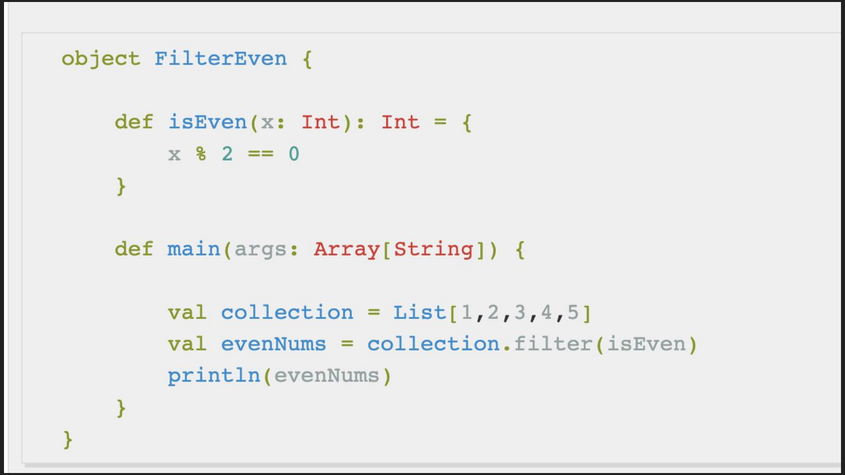
```
@cache
def factorial(n):
    return n * factorial(n-1) if n else 1

>>> factorial(10)      # no previously cached result, makes 11 recursive calls
3628800
>>> factorial(5)       # just looks up cached value result
120
>>> factorial(12)      # makes two new recursive calls, the other 10 are cached
479001600
```

Filter Even

Python



```python
def isEven(x):
    return x % 2 == 0

def main(args):

    collection = [1,2,3,4,5]
    evenNums = list(filter(isEven, collection))
    print(evenNums)
```



```scala
object FilterEven {

    def isEven(x: Int): Int = {
        x % 2 == 0
    }

    def main(args: Array[String]) {

        val collection = List[1,2,3,4,5]
        val evenNums = collection.filter(isEven)
        println(evenNums)
    }
}
```

Scala

Reduce

Python

```python
def main(args):

    from functools import reduce

    collection = [1,3,5,2,4]
    totalSum = reduce(lambda x,y: x + y, collection)
    print(totalSum)
```

Scala

```scala
object SumNumbers {

    def main(args: Array[String]) {

        val collection = List[1,3,5,2,4]
        val totalSum = collection.reduce((x, y) => x + y)
        println(totalSum)
    }
}
```

```
[1,3,5,2,4].reduceLeft((x, y) => x + y)    // initialize var acc
(((1 + 3) + 5) + 2) + 4    // take first value, acc = 1
((4 + 5) + 2) + 4    // acc = 1 + 3 = 5
(9 + 2) + 4  // acc = 4 + 5 = 9
11 + 4    // acc = 9 + 2 = 11
15    // acc = 11 + 4 = 15 returned upon end of collection
```