What is the result of the following expression written in Scala:

List(3,5,9,10).foldLeft(0)((x,y)=>x+y)

27

List(27)

List(3,5,9)

3

Write function forAllExist that receives as input 3 parameters which are a list of integers, and 2 predicates. The function returns true when all elements of the argument list satisfy the first predicate and there is at least one element satisfies the second predicate.

def forAllExist(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(f1) && lst.exists(f2)

def forAllExist(lst:List[Int],f1:Boolean,f2:Boolean) = lst.forall(f1) && lst.exists(f2)

def forAllExist(lst:List[Int],f1:Int,f2:Int) = lst.forall(f1) && lst.exists(f2)

def forAllExist(lst:List[Int],f1:Boolean=>Boolean,f2:Boolean=>Boolean) = lst.forall(f1) && lst.exists(f2)

Select the correct method invocation of filter to achieve the list of even number from the argument list? List(2,3,4,5,6).\_\_\_\_\_\_\_\_\_\_\_  => List(2,4,6)

filter(x => x % 2)

filter(x % 2 == 0)

filter(x => x % 2 == 0)

filter((x,y)=> x % 2 == y)

Write function doubleCheck that receives 3 input parameters which are a list of integers and 2 predicates. This function return true when the argument list has at least one element satisfying the first predicate and at least one element  satisfying the second predicate.

def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.exists(x => f1(x) && f2(x))

def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.exists(f1) && lst.exists(f2))

def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(x => f1(x) && f2(x))

def doubleCheck(lst:List[Int],f1:Int=>Boolean,f2:Int=>Boolean) = lst.forall(f1) && lst.forall(f2))

Given the list List(4,2,6,9), which high-order function should be used to create a list of squares of elements in the given list, i.e. List(16,4,36,81)?

foldLeft

map

filter

forall

Correct, when the number of element in the parameter list is the same as the number of elements in the result list, map is the most appropriate solution

Let reverse be a function to reverse a list (for example List(1,4,2).reverse => List(2,4,1)) and :: be used to concat an element into the beginning of a list (a :: List(b,c,d) => List(a,b,c,d)). Let lst keep a list of integers, which express works as reverse?

lst.map(x => x :: List())

lst.foldLeft(List[Int]())((x,y)=>x::y)

lst.forAll(x => x::List())

lst.foldLeft(List[Int]())((x,y) => y::x)

Let a function be declared as follows:

def increaseClosures(n:Int)(x:Float) = x + n

A variable inc3 is initialized as follows:

val inc3 = increaseClosures(3) \_

What does variable inc3 keep?

Nothing because the function call increaseClosures is wrong as there are not enough arguments

3

x + 3

A function:  Float => Float

Correct, this function is  (x:Float) => x + 3

Let listA be a list containing 3 elements {a, b, c} and  listB be a list containing 3 elements{d,e,f}, if listA.append(listB) makes listA contain 6 elements which are 3 elements from listA and 3 elements from listB, what's the kind of append?

Immutable

Mutable

Correct

Assume that Scala is a pure functional programming language and there is the following statement in Scala:

while (a < 1) { ... }

The body of the while is a black box. What can you discuss about this loop instruction?

Cannot execute any iteration

this statement will execute like the while statement in other programming language (Java)

Always loop forever

Either loop forever or no loop

Correct, in pure functional programming language, a variable represents an unknown value not a storage, so expression a < 1 is always true or false so the while will loop forever (when a < 1 true) or no iterate (when a < 1 false).