159.272 Programming Paradigms

Tutorial 1 - Working with Scala lists

There are seven exercises in this introductory tutorial . Try to complete them all. You can use the file **`tutorial1.scala`** as a template to demonstrate your results. This file provides you with correct type annotations for the functions you are asked to define. Upload your completed **`tutorial1.scala`** file to Stream.

Given a list object

```
val myList = "apple" :: "grape" :: pear" :: "plum" :: Nil
```

recall the results of following first-order **`List`** methods

- **`myList.isEmpty`** returns **`False`**
- **`myList.head`** returns **`"apple"`**
- **`myList.tail`** returns **`"grape" :: pear" :: "plum" :: Nil`**
- **`myList.last`** returns **`"plum"`**
- **`myList.init`** returns **`"apple" :: "grape" :: pear" :: Nil`**
- **`myList.length`** returns **`4`**
- **`myList.reverse`** returns **`"plum :: "pear :: "grape :: apple" :: Nil`**

1. Using these methods, define a function **`second`**, that returns the second element in a list. Your function only has to work for lists with at least two elements.

   ```
   def second[T](xs:List[T]): T = {

     }
   ```

   For example:

   ```
   second(List(1,2,3,4))
   2
   ```

2. Using these methods, define a function **`middle`**, that returns a list minus its first and last elements. If the list has two or fewer elements, your function should return the empty list.

   ```
   def middle[T](xs:List[T]): List[T] = {

     }
   ```

   For example:

   ```
   middle(List(1,2,3,4))
   List(2, 3)
   ```

3. Write a function that takes three lists of integers as its arguments and returns a single list that contains the first element of each of the lists. Your function only has to work for lists with at least one element.

   ```
   def heads(xs:List[Int, ys:List[Int], zs:List[Int]): List[Int] = {

     }
   ```

For example:

```
heads(List(1,2,3,4), List(5,6,7), List(1,1,1,1))
List(1, 5, 1)
```

4. Using list pattern matching and recursion, as shown in lectures, write a function **sum** that returns the sum of a list of integers.

```
def sum(xs:List[Int]): Int =
  xs match {
    case ?
    case ?
  }
```

5. Using the same approach, write definitions for
   - a **product** function that multiplies a list of integers together and returns the result.

```
def product(xs:List[Int]): Int = {

  }
```

   - a function **joinLists** that takes a list of lists and returns a single list consisting of all the lists concatenated together.

```
def joinLists[T](xss:List[List[T]]): List[T] = {

  }
```

6. Define a function **compress** that eliminates consecutive duplicates of list elements. If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed. Use list pattern matching and recursion to solve this problem.

```
def compress[T](xs:List[T]: List[T] = {

  }
```

For example:

```
compress ['a','a','a','a','b','c','c','a','a','d','e','e','e','e']
['a','b','c','a','d','e']
```