Programming Paradigms Exercise 2 - Ruby

Johann Gamper   Radityo Eko Prasojo   Thomas Tschager
1st Semester 2018/19

1. Assume time series data stored in an array \( a \). Write a Ruby function \( \text{maxtrend}(a) \) that returns the length of the longest positive trend (i.e., number of strictly increasing consecutive values). For example, for \( t = [3, 1, 4, 5, 9, 8] \) the return value is 4.

2. Write a class \( \text{Vehicle} \) which has a field \( \text{year} \) and a method \( \text{printStats} \) which prints the value of \( \text{year} \). The value of \( \text{year} \) must be given as an argument to the constructor of \( \text{Vehicle} \).

Next, write a class \( \text{Car} \) which extends \( \text{Vehicle} \) and has two fields \( \text{model} \) and \( \text{brand} \). The constructor of \( \text{car} \) must get three arguments, assign values to \( \text{model} \) and \( \text{brand} \) and call the constructor of the superclass \( \text{Vehicle} \) in initialize \( \text{year} \). Also, create a method \( \text{printStats} \) which prints \( \text{model} \) and \( \text{brand} \), and also calls the parent method in \( \text{Vehicle} \) to print \( \text{year} \).

Finally, create a module \( \text{OldtimerCheck} \) with a method \( \text{isOldtimer?} \), that checks if a car is older than 30 years, and include it in the mixin of the \( \text{Car} \).

3. Define a module \( \text{Debug} \) with a method \( \text{printIdentifier} \) that prints the class name and the object it is called from. (\( \text{ Hint: } \) You can get the class name of an object using \( \text{self.class.name} \).) Add the module to the mixins of other classes to test its functionality.

4. Write a method \( \text{profile} \) that takes a string \( \text{name} \) and a proc \( \text{codesnippet} \) as input, measures the execution time of \( \text{codesnippet} \), and finally prints the \( \text{name} \) and the execution time of \( \text{codesnippet} \). (\( \text{Hint: } \) Use \( \text{Time.now} \) to get the current time.)

5. Extend the Ruby class \( \text{Fixnum} \) with a method \( \text{square_root_times} \) that, if called for a number \( n \) and a code snippet, executes the code snippet \( \left\lceil \sqrt{n} \right\rceil \) times. For example, \( 5.\text{square_root_times}{ \text{puts 'hello world!' } } \) produces

\[
\text{hello world!}
\text{hello world!}
\text{hello world!}
\]

You are not allowed to use a built-in Ruby function to compute the square root of numbers.
6. Write a module **Inject** that will be included to the mixins of the class **Array** to sum up all numbers in an array. The module should contain

   (a) a method **inject** that is called with a initial value $n$ and a code snippet. The method executes the code snippet for each element of the array, adds the result to $n$, and finally returns $n$.

   (b) Moreover, the module should contain a method **mysum** that calls the method **inject** with $n = 0$ and a code block that sums up all elements of the array.

Add the module to the mixins of the class **Array**, so that $[1,2,3,4,5]$.**mysum** returns 15.

Now, add a third method **myprod** to the module **Inject** that calls **inject** with $n = 1$ and a code snippet that takes the product of all elements of the array. Check, if $[1,2,3,4,5]$.**myprod** results 120.

Finally, try to add the module to the mixins of the class **Range** and check its functionality.