1. For the language with which you are most familiar (this may or may not be the first that you learned), list the three things that you wish had been differently designed. Why do you think they were designed the way they were? How would you fix them if you had the chance to do it over? Would there be any negative consequences – for example, in terms of compiler complexity or program execution speed?

2. Read “The Semicolon Wars” (see online syllabus). Argue why “camelcase” and “sulkingcamelcase” makes sense to use when writing a program in your favorite programming language (e.g. as a type name, macro name, identifier name, etc.).

3. Find out more about “Hungarian notation” and give a list of at least 16 examples of naming conventions for types suggested for names in C/C++.

4. List the person(s) who were instrumental to the development of Python, Pascal, C, C++, Java, Smalltalk-80. In what year(s) were these languages initially developed?

5. Explain the difference between a functional and a special form in Scheme.

6. Why is Scheme called “homoiconic”?

7. What is NOR (normal order reduction/evaluation) and AOR (applicative order reduction/evaluation) in lambda calculus (Chapter 11.7)? Show an example.

8. Which Scheme construct(s) will cause a Scheme program to depart from a purely functional programming model, i.e. are not considered “pure”?

9. We can implement a binary tree data structure by using lists with three elements: (value left-tree right-tree). A leaf is an empty list ‘()'.
   a. Given the tree:

```
     3
    / \
   1   5
    /   \
   4
```
b. For this example, give the expression in Scheme to obtain the value “5” from the list representation of the tree (use \texttt{car} and \texttt{cdr}).

10. Define a Scheme function “\texttt{allpos}” that takes a list of integers, applies a function that converts an integer to true if the integer is positive and to false otherwise, and reduces the resulting list to a single true/false by conjunction. To keep our code compact, use higher-order Scheme functions whenever possible. See lecture slide 27. The result should be: \texttt{(allpos '(1 2 3)) => #t} and \texttt{(allpos '(0 1 2)) => #f}