Why should we be more interest in functional programming than we are?

Computation is the evaluation of math and logic functions.

State is not changed – no side effects (no assignment statement)

Developed from lambda calculus

Referential transparency- execution results are the same, regardless of time that is, results only depend on argument values loops are discarded in favor of recursive calls

Anonymous subroutines

Lazy evaluation (sometimes manually via above)

Procedures are first class objects (as is data)

What is a type system and what benefits does it provide?

A grouping of values into types syntax and grammar of available types & portions of parser handling type declarations & portions of compiler checking types of parameters enables coders to describe data effectively and to prevent operations that make no sense during run-time Types of operands must be checked before op is applied

Statically typed language (Java, Haskell): all variables and expressions have fixed type types are checked at compile-time

Dynamically typed language (Scheme, Python): variables and expressions do not have fixed type types are checked at run-time

How are call-by-value, call-by-reference, call-by-value-result, call-byname implemented?

Call-by-value: A copy of the value of the variable is made and passed to the corresponding formal parameter - variable’s value is unchanged after the procedure returns

Call-by-reference: The location of the variable is passed to the corresponding formal parameter - the variable may be updated several times in the body of the procedure

Call-by-value-result: Same as call-by-value except that the variable takes the value of the corresponding formal parameter at termination

Call-by-name: The address of a function that computes the value of the actual parameter - effectively substitutes the parameter

What is type safety, thread safety, memory safety and why are these things important?

Type safety: The absence of erroneous or undesirable behavior caused by a discrepancy between differing data types. Java is supposed to be type safe, Haskell is pretty much type-safe, scheme is not.

Thread safety: only manipulates shared data structures in a manner that guarantees safe execution by multiple threads at the same time.

Memory Safety: Avoid jumps to invalid data addresses and manipulation of code addresses

C/C++ are highly unsafe - dangling pointers, etc.

Java is fairly safe - no deallocator

Haskell is safe - no assignment

Modify a given recursive Scheme function to be tail recursive?

Rewrite again in continuation passing style?

Rewrite a more difficult function in CPS?

Use call/cc and assert to solve a logic problem?

Use foldr and foldr (Haskell) in the proper way for map-reduce?

Use maybe Monad to handle exceptional cases?

Create a new data type, supporting polymorphism, and type constructor?

Use maybe monad to make code more generally applicable.

Solve a problem in Haskell using a list comprehension

What makes a data type a Monad?