**Option 1. OO vs. Functional Programming**

So you've had some experience with Object-Oriented Programming (OOP) this week. Some consider Functional programming the opposite of OO - compare them.

**Option 2: Variable Scope**

What kinds of variables does Ruby support? Where are they accessible? When would you use them and what makes them different?

**Option 3: Class Methods**

Research class methods and desribe what they are, how to use them, and when you would use them

**Option 4: Blocs, Procs, and Lambdas**

What are these? How do they work?

**Option 5: Classes vs Modules**

Describe and explain the purposes of classes and modules. How are they similar? How do they compare?

**Option 6: Inheritance vs Composition**

What is ruby inheritance? What is composition? Which have you used this week? Create your own examples to explain how to use each.

### Higher Order Functions & Their Use

A function is something that can be invoked or, to use the terminology of the lambda calculus, can be applied to arguments.

A higher order function is one that may take functions as arguments or return functions and does some operation with them, either manipulating them or applying them. Examples include a sort function which takes a comparison function or a function which takes two functions and returns a function which invokes both.

Functional programming

**What is it?**

Functional programming evaluates mathematical functions via expressions. Pure functional programming eliminates ‘side effects’ (i.e. changes on the state that do not depend on function inputs), so it can be much easier to understand and predict, as the same functions when re-applied result in the same value. As such, it describes a desired result, commonly referred to as ‘declarative programming”.

Functional programming is believed to have evolved out of lamda calculus, *“a formal system of mathematical logic and computer science for expressing computation based on functions…and variables…”* (Wikipedia).

It was created to support the functional or procedural approach to problem-solving, which details every step a computer must take to achieve a goal (also called “algorithmic programming”). It’s akin to a long procedure sheet of code which mixes up data and behaviours throughout.

**How does it work?**

You define carefully the input to each function and what each function returns.. It’s a form of declarative programming.

**What languages use it?**

Haskell, Miranda, Clean.

**What are its advantages?**

Because each function is specifically designed to perform a specific task given its arguments, it is easy to read and maintain.

And because of this readability and ease of maintenance, it is also easier to iterate with new designs, whereby functions can be tested in isolation.

Object-Orientated

**What is it?**

OO is believed to be an off-shoot from procedural programming. It is built on the concept of self-contained units, called “objects”, whereby data (variables) and its behaviour (functions) are encapsulated – this means the actions of one object cannot interfere with those of another (unless specifically manipulated to do so). By hiding this data not directly accessible by other parts of the program, methods are required to enable encapsulated data to be accessed (and manipulated). These objects point back to classes, which are designed in “class hierarchies’. Classes are ideas, blueprints or concepts that facilitate the creation of these objects (they are akin to sets and subsets arising from Set Theory, in mathematics). Other key principles include:

* + Dynamic dispatch – where the object determines the code to be run by looking up the method.
  + Polymorphism – all data structures are connected in some way, and via this modular approach to data structure, facilitates class and object hierarchy and inheritance.
  + Open recursion – whereby larger problems can be solved by splitting them into sub-problems and then combine all the results to solve the original problem (i.e. the “divide and conquer” approach).

**Why use it?**

OO programming is particularly useful for when there is a fixed set of operations for a program. It facilitates the addition of new mini-programs to the original program. For instance, new classes (in-built or developer-created) can be added to implement existing or new methods.

**What languages use it?**

Ruby, Python, Javascript, Perl, PHP and Objective-C.

**Which one?**

James Hague argues that 100% functional purity is impractical and that one should aim for [85% purity](http://www.johndcook.com/blog/2010/04/15/85-functional-language-purity/). But the 15% impurity needs to be partitioned, not randomly scattered across your code base. A simple strategy for doing this is to use [functional in the small and OO in the large](http://www.johndcook.com/blog/2009/03/23/functional-in-the-small-oo-in-the-large/).