What is an imperative language?

The programming model in imperative languages is based on a statement-at-a-time paradigm where each statement has some effect on a memory store. Imperative programming is centered around the assignment statement, which allows one to change the content of cells in the memory store. Programs written in imperative languages are generally harder to write, debug, and maintain compared to those written in declarative languages. Imperative programming lays more stress on "how" a solution procedure is specified. Programs written in imperative languages are generally larger in terms of code size and run faster compared to programs written in declarative languages. Imperative languages do not have a solid mathematical basis (although in the semantics part of the course, we will see how we can solve this problem).

**What is a declarative language?**

The programming model in declarative languages is based on stating the relationship between inputs and outputs. The actual computation procedure adopted is left to the runtime system. A declarative program can be viewed as a high level specification. Declarative programs are shorter, supposedly easier to write, debug, and maintain. Declarative programs are generally slower than imperative programs in execution speed.

**What is a functional language?**

functional programming defines the outputs of a

program as a mathematical function of the inputs, with no notion of internal

state, and thus no side effects.

**Why are scripting languages described by the term scripting?**

By contrast scripting

languages tend to stress flexibility, rapid development, local customization, and

dynamic (run-time) checking.

A **scripting language** or **script language** is a [programming language](http://en.wikipedia.org/wiki/Programming_language) that supports **scripts**, programs written for a special [run-time environment](http://en.wikipedia.org/wiki/Run-time_environment) that can [interpret](http://en.wikipedia.org/wiki/Interpreted_language) (rather than[compile](http://en.wikipedia.org/wiki/Compiler)) and [automate](http://en.wikipedia.org/wiki/Automate) the [execution](http://en.wikipedia.org/wiki/Execution_(computing)) of tasks that could alternatively be executed one-by-one by a human operator.

**Why can we say neither an imperative language nor a declarative language more powerful that the other?**

 This result led Church to conjecture that any intuitively

appealingmodel of computing would be equally powerful as well; this conjecture

is known as Church’s thesis.

**Topics**

You should be able to discuss these topics and issues. You should also be familiar with the assigned Homework problems and study problems.

These are a comprehensive, but not complete, list of what you are expected to know.

***Sections 10.1 and 10.2***

**Explain why neither imperative or functional languages are more powerful.**

Over time, these various formalizations were

shown to be equally powerful: anything that could be computed in one could be

computed in the others. This result led Church to conjecture that *any* intuitively

appealing model of computing would be equally powerful as well; this conjecture

is known as *Church’s thesis*.

**Be able to explain the characteristics of a functional language.**

First-class function values and higher-order functions

\_ Extensive polymorphism

\_ List types and operators

\_ Recursion

\_ Structured function returns

\_ Constructors (aggregates) for structured objects

\_ Garbage collection

**Be able to explain a first class function.**

Functional programming requires that functions are *first-class*, which means that they are treated like any other values and can be passed as arguments to other functions or be returned as a result of a function. Being first-class also means that it is possible to define and manipulate functions from within other functions. Special attention needs to be given to functions that reference local variables from their scope. If such a function escapes their block after being returned from it, the local variables must be retained in memory, as they might be needed later when the function is called. Often it is difficult to determine statically when those resources can be released, so it is necessary to use automatic [memory management](https://wiki.haskell.org/Memory_management).

**What is a higher order function?**

A *higherorder*

*function* takes a function as an argument or returns a function as a result

**What is meant by side effects?  What are the benefits of not having side effects?**

in addition to returning a value, it also modifies some [state](http://en.wikipedia.org/wiki/State_(computer_science)) or has an *observable* interaction with calling functions or the outside world. For example, a function might modify a [global variable](http://en.wikipedia.org/wiki/Global_variable) or [static variable](http://en.wikipedia.org/wiki/Static_variable), modify one of its arguments, raise an exception, write data to a display or file, read data, or call other side-effecting functions. In the presence of side effects, a program's behavior may depend on history; that is, the order of evaluation matters. Understanding and debugging a function with side effects requires knowledge about the context and its possible histories.[[1]](http://en.wikipedia.org/wiki/Side_effect_%28computer_science%29#cite_note-1)[[2]](http://en.wikipedia.org/wiki/Side_effect_%28computer_science%29#cite_note-2)

Easy to maintain debug

***Section 11.1***

**What is meant by resolution and unification?**

Unification, in [computer science](http://en.wikipedia.org/wiki/Computer_science) and [logic](http://en.wikipedia.org/wiki/Logic), is an algorithmic process of [solving](http://en.wikipedia.org/wiki/Equation_solving) [equations](http://en.wikipedia.org/wiki/Equations) between

symbolic [expressions](http://en.wikipedia.org/wiki/Expression_(mathematics)).

Resolution

 iteratively applying the resolution rule in a suitable way allows for telling whether a [propositional formula](http://en.wikipedia.org/wiki/Propositional_formula) is satisfiable and for proving that a first-order formula is unsatisfiable.

***Section 13.1***

**What role did the Web play in popularizing scripting languages?**

With the growth of theWorldWideWeb, scripting

languages have gained new prominence in the generation of dynamic content.

They are also widely used as *extension languages*, which allow the user to

customize or extend the functionality of “scriptable” tools.

**What is meant by client-side scripting and server-side scripting?**

server side” web scripting, in which a web server executes a program

(on the server’s machine) to generate the content of a page. Instead client side they are executed on the users end