1. Programming Paradigms

Before we start on the functional programming paradigm we give a broad introduction to programming paradigms in general.

In this section we will discuss the meaning of the word 'paradigm', and we will enumerate the main programming paradigms, as we see them.

In Chapter 2 we will discuss each of the main programming paradigms in some details. Be aware, however, that this material is about the functional programming paradigm. The two first chapters of the material mainly serve as background, and as contrast for an enhanced understanding of the functional school.

1.1. Paradigm

Lecture 1 - slide 2

It is interesting and useful to investigate the meaning of the word 'paradigm'

We will here look at the meaning of the word 'paradigm', as it appears in 'The American Heritage Dictionary of the English Language, Third Edition':

> *"An example that serves as pattern or model."*

Another and slightly more complicated explanation stems from the 'The Merriam-Webster's Collegiate dictionary':

> *"A philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated"*

Below we will first describe the meaning of the word 'paradigm' as we adopt it in this course. After that we describe related concepts, namely 'programming technique', 'programming style', and 'programming culture'.

- Programming paradigm (in this course)
  - A pattern that serves as a *school of thoughts* for programming of computers
- Programming technique
  - Related to an algorithmic idea for solving a particular class of problems
  - Examples: 'Divide and conquer' and 'program development by stepwise refinement'
- Programming style

    - The way we express ourselves in a computer program
    - Related to elegance or lack of elegance
- Programming culture
    - The totality of programming behavior, which often is tightly related to a family of programming languages
    - The sum of a main paradigm, programming styles, and certain programming techniques.

1.2. The main programming paradigms

Lecture 1 - slide 3

In this section we will enumerate the four main programming paradigms which will be treated in additional details in Chapter 2. It may very well be a matter of taste if some of the additional programming paradigms, which we also mention below, should be considered as main programming paradigms as well.

We identify four main programming paradigms and a number of minor programming paradigms

In the concept definition below, we characterize a main programming paradigm in terms of an *idea* and a *basic discipline*.

A *main programming paradigm* stems an *idea* within some *basic discipline* which is relevant for performing computations

- Main programming paradigms
    - The imperative paradigm
    - The functional paradigm
    - The logical paradigm
    - The object-oriented paradigm
- Other possible programming paradigms
    - The visual paradigm
    - One of the parallel paradigms
    - The constraint based paradigm

In Chapter 2 we will characterize the four main programming paradigms mentioned above. We will in particular attempt to trace the idea and basic discipline behind the four main programming paradigms. We do not go into the other programming paradigms, mentioned above, in this material.

1.3. References

[-] Foldoc: visual programming
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=visual+programming

2. Overview of the four main programming paradigms

In this section we will characterize the four main programming paradigms, as identified in Section 1.2.

As the main contribution of this exposition, we attempt to trace the *basic discipline* and the *idea* behind each of the main programming paradigms.

With this introduction to the material, we will also be able to see how the functional programming paradigm corresponds to the other main programming paradigms.

2.1. Overview of the imperative paradigm

Lecture 1 - slide 5

The word 'imperative' can be used both as an adjective and as a noun. As an adjective it means 'expressing a command or plea'. In other words, asking for something to be done. As a noun, an imperative is a command or an order. Some programming languages, such as the object oriented language Beta, uses the word 'imperative' for commands in the language.

First **do this** and next **do that**

The 'first do this, next do that' is a short phrase which really in a nutshell describes the spirit of the imperative paradigm. The basic idea is the command, which has a measurable effect on the program state. The phrase also reflects that the order to the commands is important. 'First do that, then do this' would be different from 'first do this, then do that'.

In the itemized list below we describe the main properties of the imperative paradigm.

- Characteristics:
  - Discipline and idea
    - Digital hardware technology and the ideas of Von Neumann
  - Incremental *change of the program state* as a function of *time.*
  - Execution of computational steps in an order governed by *control structures*
    - We call the steps for *commands*
  - Straightforward abstractions of the way a traditional Von Neumann computer works
  - Similar to descriptions of everyday routines, such as food recipes and car repair
  - Typical commands offered by imperative languages
    - Assignment, IO, procedure calls
  - Language representatives
    - Fortran, Algol, Pascal, Basic, C

- The natural abstraction is the procedure
    - Abstracts one or more actions to a procedure, which can be called as a single command.
    - "Procedural programming"

We use several names for the computational steps in an imperative language. The word *statement* is often used with the special computer science meaning 'a elementary instruction in a source language'. The word *instruction* is another possibility; We prefer to devote this word the computational steps performed at the machine level. We will use the word 'command' for the imperatives in a high level imperative programming language.

A procedure abstracts one or more actions to a procedure, which can be activated as a single action.

2.2. Overview of the functional paradigm

Lecture 1 - slide 6

We here introduce the functional paradigm at the same level as imperative programming was introduced in Section 2.1.

Functional programming is in many respects a simpler and more clean programming paradigm than the imperative one. The reason is that the paradigm originates from a purely mathematical discipline: the theory of functions. As described in Section 2.1, the imperative paradigm is rooted in the key technological ideas of the digital computer, which are more complicated, and less 'clean' than mathematical function theory.

Below we characterize the most important, overall properties of the functional programming paradigm. Needless to say, we will come back to most of them in the remaining chapters of this material.

Evaluate an expression and use the resulting value for something

- Characteristics:
    - Discipline and idea
        - Mathematics and the theory of functions
    - The values produced are *non-mutable*
        - Impossible to change any constituent of a composite value
        - As a remedy, it is possible to make a revised copy of composite value
    - Atemporal

        - Abstracts a single expression to a function which can be evaluated as an expression
    - Functions are first class values
        - Functions are full-fledged data just like numbers, lists, ...
    - Fits well with computations driven by needs
        - Opens a new world of possibilities

2.3. Overview of the logic paradigm

Lecture 1 - slide 7

The logic paradigm is dramatically different from the other three main programming paradigms. The logic paradigm fits extremely well when applied in problem domains that deal with the extraction of knowledge from basic facts and relations. The logical paradigm seems less natural in the more general areas of computation.

Answer a question via search for a solution

Below we briefly characterize the main properties of the logic programming paradigm.

- Characteristics:
    - Discipline and idea
        - Automatic proofs within artificial intelligence
    - Based on axioms, inference rules, and queries.
    - Program execution becomes a systematic search in a set of facts, making use of a set of inference rules

2.4. Overview of the object-oriented paradigm

Lecture 1 - slide 8

The object-oriented paradigm has gained great popularity in the recent decade. The primary and most direct reason is undoubtedly the strong support of encapsulation and the logical grouping of program aspects. These properties are very important when programs become larger and larger.

The underlying, and somewhat deeper reason to the success of the object-oriented paradigm is probably the conceptual anchoring of the paradigm. An object-oriented program is constructed with the outset in concepts, which are important in the problem domain of interest. In that way, all the necessary technicalities of programming come in second row.

Send messages between objects to simulate the temporal evolution of a set of real world phenomena

As for the other main programming paradigms, we will now describe the most important properties of object-oriented programming, seen as a school of thought in the area of computer programming.

- Characteristics:
  - Discipline and idea
    - The theory of concepts, and models of human interaction with real world phenomena
  - Data as well as operations are encapsulated in objects
  - Information hiding is used to protect internal properties of an object
  - Objects interact by means of message passing
    - A metaphor for applying an operation on an object
  - In most object-oriented languages objects are grouped in classes
    - Objects in classes are similar enough to allow programming of the classes, as opposed to programming of the individual objects
    - Classes represent concepts whereas objects represent phenomena
  - Classes are organized in inheritance hierarchies
    - Provides for class extension or specialization

This ends the overview of the four main programming paradigms. From now on the main focus will be functional programming in Scheme, with special emphasis on examples drawn from the domain of web program development.

2.5. References

[-] Foldoc: object-oriented programming
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=object-oriented+programming

[-] Foldoc: logic programming
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=logic+programming

[-] Foldoc: functional programming
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=functional+programming

[-] Foldoc: imperative
http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?query=imperative