Programming Paradigms

↳ Programming languages

- A Programming language is a notational system for describing tasks/computations in a machine and human readable form.
- Most computer languages are designed to facilitate certain operations and not others: numerical computation, or text manipulation, or I/O.
- More broadly, a computer language typically embodies a particular *programming paradigm*.

↳ Characteristics of a programming language:
Every language has syntax and semantics:

- **Syntax**: The syntax of a program is the form of its declarations, expressions, statements and program units.
- **Semantic**: The semantic of a program is concerned with the meaning of its program.

↳ Which programming language?

- Since a task can be solved in different ways (paradigms), the language used to describe the solution differs in abstractions, structures due to the way in which the problem is solved.

- There is no theory that dictates the best paradigm to solve a particular problem.
- Efforts by Sebesta in his Concepts of Programming Languages book:
    - He based his evaluation criteria on three factors and 9 characteristics.
    - The three criteria (R,W,R) are:
        - Readability
        - Writability
        - Reliability
    - The nine characteristics are:
        - **Simpilicity/orthogonality(R,W,R):**
            - "Orthogonality in a programming language means that a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language" [Sebesta]
            - Relatively small set of primitive constructions combined in a number (of logically consistent) ways to provide the required control and data structures.
            - The concepts of a programming language do not interfere with each other: different methods of passing parameters.
            - **Non-orthogonality**: means exceptions to the general language rules, which make it harder to learn. It means that you cannot combine language features in all possible ways.
            - Examples of **non-orthogonal** languages:
                - Arrays in Perl4 can't contain other arrays.
                - In C, parameters can be passed by value, unless they are arrays, in which case they are passed by reference.

        - In C++, we start a Switch statement by using the Switch keyword, followed by the expression that we would like to evaluate. This expression is just a single variable or a complex expression that must evaluatge to an integral type (that is, char, short, int, long, or enum). Floating point variables and other non-integral types may not be used here.
    - Good Orthogonality:
        - Functional languages (Lisp, ML,etc.): everything is a function.
    - Given a programming language:
        - ***m*** options on x axis
        - ***n*** options on y axis
        - Total number of possible interactions:
            - ***mn*** interactions
        - Total number of basic facts to learn:
            - ***m+n*** facts
        - Total number of exceptions:
            - ***e*** exceptions
        - Basic elements of the language:
            - ***m+n+e*** facts and exceptions to be learned
        - Total number of features:
            - ***mn-e***
        - Objective:
            - ***m+n+e*** << ***mn-e***
- **<u>Control structures (R,W,R)</u>**
- **<u>Data types and structures (R,W,R):</u>**
    - Adequate data types help increase the readability of a programming language.
    - Example:
        - Boolean data type:

- How do you interpret the followingstatement: visited = 1;

    - Is this an assignment or a conditional statement?

- **Syntax design (R,W,R):**
    - Special words which is easier to read:
        - braces or "if…end if" provided by Ada?
        - Braces or "Begin..end" provided in Pascal.
- **Support for abstraction (,W,R):**
    - Ability to hide detail
    - Abstraction is an important factor in the writability of a language.
    - Two types of abstraction: process and data
        - Process: subprograms and modules
        - Data: structures, records, objects

- **Expressivity (,W,R):**
    - Programming languages with poor support for abstraction and weak primitives will have poor writability.
    - Short circuit evaluation for Boolean expressions:
        - Ada uses "and then" and "or else"
        - Java uses:
            - "&&" and "||" (regular operator for "and" and "or" operators
            - "&" and "|" for short circuit evaluation

- **Type checking (,,R)**
- **Exception handling (,,R)**
- **Restricted aliasing (,,R):**
    - example: (C)

```
int salary, *p_salary;
salary = 98000;
p_salary = &salary;
```
salary and *p_salary are aliases.

	Readability	Writability	Reliability
Syntax design	★	★	★
Control structures	★	★	★
Data types	★	★	★
Simplicity/orthogonality	★	★	★
Abstraction		★	★
Expressivity		★	★
Type checking			★
Exception handling			★
Restricted aliasing			★

- Maintainability
  - **Factoring**: The ability to group related features into a single unit. Use subroutines to group related computations units so they can be re-used in different parts of the application.
  - **Locality**: The ability to implement information hiding so that changes to a grouping (either control or data) are transparent.
- Cost
  - Programmer training
  - Software creation
  - Compilation
  - Execution
  - Compiler cost
  - Poor reliability
  - Maintenance
- Others: portability and generality

Programming paradigms

- The paradigms are not exclusive, but reflect the different emphasis of language designers. Most practical languages embody features of more than one paradigm.

- **Classification:**

Imperative/ Algorithmic	Declarative		Object-Oriented
	Functional Programming	Logic Programming	
Algol Cobol PL/1 Ada C Modula-3	Lisp Haskell ML Miranda APL	Prolog	Smalltalk Simula C++ Java

Imperative paradigms

- It is based on commands that update variables in storage. The Latin word *imperare* means "to command".
- The language provides statements, such as *assignment statements*, which explicitly change the *state* of the memory of the computer.
- This model closely matches the actual executions of computer and usually has high execution efficiency.
- Many people also find the imperative paradigm to be a more natural way of expressing themselves.

↳ Functional programming paradigms

- In this paradigm we express computations as the evaluation of mathematical functions.

- Functional programming paradigms treat values as single entities. Unlike variables, values are never modified. Instead, values are transformed into new values.

- Computations of functional languages are performed largely through applying functions to values, i.e., (+ 4 5).

Logic programming paradigms

- In this paradigm we express computation in exclusively in terms of ***mathematical logic***.

- While the functional paradigm emphasizes the idea of a mathematical function, the logic paradigm focuses on predicate logic, in which the basic concept is a *relation*.

- Logic languages are useful for expressing problems where it is not obvious what the functions should be.

- For example consider the *uncle* relationship: a given person can have many *uncles*, and another person can be *uncle* to many *nieces* and *nephews*.

- Let us consider now how we can define the *brother* relation in terms of simpler relations and properties *father, mother, and male*. Using the Prolog logic language one can say:

```
brother(X,Y)      /* X is the brother of Y                    */
                  /* if there are two people F and M for which*/
     father(F,X),         /*       F is the father of X      */
     father(F,Y),         /*  and  F is the father of Y       */
     mother(M,X),         /*  and  M is the mother of X       */
     mother(M,Y),         /*  and  M is the mother of Y       */
     male(X).             /*  and  X is male                  */
```

The Object-Oriented Paradigm

- OO programming paradigm is not just a few new features added to a programming language, but it a new way of

thinking about the process of decomposing problems and developing programming solutions

- Alan Kay characterized the fundamental of OOP as follows:
    - Everything is modeled as object
    - Computation is performed by message passing: objects communicate with one another via message passing.
    - Every object is an instance of a class where a class represents a grouping of similar objects.
    - Inheritance: defines the relationships between classes.

- The Object Oriented paradigm focuses on the *objects* that a program is representing, and on allowing them to exhibit "behavior".

- Unlike imperative paradigm, where data are passive and procedures are active, in the O-O paradigm data is combined with procedures to give *objects*, which are thereby rendered active.

↳ Concurrent programming

- Improve performance
- Multiprogramming systems attempt to utilize resources that would otherwise be wasted, by running two or more jobs concurrently.
- Multiaccess systems extend this principle, allowing many jobs to be run, each on behalf of a user at an interactive terminal.

- Concurrency can be classified into:

- Apparent concurrency: single processor (interleaved execution of concurrent tasks)
- Real concurrency: multiprocessor environment

- Issues:
  - How to synchronize the interactions among concurrently executing processes to maintain the internal data integrity.
  - Another problem is to schedule the racing processes for a limited set of shared resources.