Unit 3: Programming Languages

History

Programming language can be traced back to 1948 which was when the SSEM (Small Scale Experimental Machine) ran its first programming language. In the 50s, Fortran, Cobol and Lisp were released.

Programming Paradigms

There are four general paradigms of programming namely: Imperative, Functional, Logical, Object-oriented although most modern programming languages are based on multiple paradigms.

Language Concepts:

Abstraction: Hiding implementation details of a class from Users or Programmers

Casting (or Typecasting): conversion from one datatype to another

Encapsulation: A way of collecting related data and procedures into a class.

Garbage Collection: is a method of memory management

Grammar: The grammar of a programming language is used to convert a human-readable program in a given language into a syntax tree by a parser.

Inheritance: Inheritance is one of the principles that supports DRY (don't repeat yourself) by allowing child classes to inherit (I.e. use) methods defined in parent classes.

Mutability: whether an object, once created can be changed or not.

Pointers: used for fast copying data and direct access to RAM

Polymorphism: it allows child classes to have methods with the same name as their parent classes.

Queue: it is a data structure in computer science. Queues work in FIFO order.

Recursion: when a function calls itself

Regex (Regular Expressions): are another often used programming paradigm. They are a superset of the commonly used wild cards (*,?, and so on) utilised in search strings in database queries and suchlike.

Stacks: it is a data structure in computer science. Stacks work in LIFO order.

Syntax: The syntax provides a set of rules that governs whether is program is correctly formed.

Programming Challenges

They are a lot of design related errors caused by poor programming style which is why one of the first best practices in software design is to follow the style guidelines –making it easier to detect insecure or error-prone programming. Python is a good example. It has a style guide (PEP 8) and a set of guidelines (PEP 20).

Use of certain libraries also pose a risk.

Mechanism that can increase user's trust include: cryptographic signature and hashes. Other considerations are principle of least privilege, open design, fail safe defaults, separation of privileges and the least common mechanism.

Design patterns

Design patterns describe the ways in which code can be organised to provide solutions to well-known and frequently-encountered software engineering problems.

Anti-patterns tend to evolve when a pattern which is appropriate to the solution does not exist. Examples of anti-pattern include: Spaghetti, big ball of mud and copy & paste programming