Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages

Assignment-01: Exploring Programming Paradigms

Gokulachselvan C D

21st January, 2024

Programming Paradigm

- A programming paradigm is a fundamental style or approach to programming that provides a set of principles, concepts, and guidelines for designing and structuring code.
- Programming paradigms decide how tasks are expressed, organized, and executed in a programming language and it defines the overall flow and structure of a program.
- In simple words, programming paradigm is the classification of programming languages based on their features.
- There are various programming paradigms. They are broadly divided in two sub classes. They are:

Imperative programming paradigm

- **Features:**
  - It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step task by changing state. The main focus is based on how to achieve the goal.
  - In simple words, Imperative programming is an approach that focuses on describing a sequence of steps for the computer to perform to achieve a specific result. Here's a simple example in python, which is an imperative programming language:

```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

    number = 5
    result = factorial(number)
    print(f"The factorial of {number} is {result}")
```

  - This example shows the imperative nature of the code, it explicitly outlines the steps to achieve the desired outcome, by keep on changing the program's state (the value of the result variable) through a series of instructions till we obtain a desired result.

- **Advantages:**
  - Very simple to implement.
  - It contains loops, variables etc.
- **Disadvantages:**
  - Complex problem cannot be solved.
  - Less efficient and less productive.
  - Parallel programming is not possible.

- **Imperative programming is divided into three broad categories:**
  - **Procedural:**
    * Procedural means a list of instructions that were given to the computer to do something. Procedural programming aims more at procedures.
    * Programs are organized in the form of subroutines or sub programs.
    * All data items are global.
    * Suitable for small sized software application.
    * Difficult to maintain and enhance the program code as any change in data type needs to be propagated to all subroutines that use the same data type. This is time consuming.
    * Example: FORTRAN and COBOL.
  - **Object Oriented:**
    * Object Oriented Programming paradigm emphasizes on the data rather than the algorithm.
    * It implements programs using classes and objects.
    * Data abstraction is introduced in addition to procedural abstraction.
    * Data and its associated operations are grouped in to single unit.
    * Programs are designed around the data being operated.
    * Relationships can be created between similar and distinct data types.

    - Example: C++, Java, VB.Net, Python etc.
  - **Parallel Processing:**
    - Parallel processing is the processing of program instructions by dividing them among multiple processors.
    - A parallel processing system possess many numbers of processor with the objective of running a program in less time by dividing them.
    - This approach seems to be like divide and conquer.
    - Examples: NESL (Nested Data-Parallel Language) and C/C++ also supports parallel processing because of some library function uses parallel processing method.

Declarative programming paradigm

- **Features:**
  - Declarative programming is a programming paradigm that emphasizes expressing what a program should accomplish rather than specifying how to achieve it.
  - In this paradigm, the programmer declares the desired outcome, and the underlying system takes care of the implementation details.
  - It often considers programs as theories of some logic. It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing.
  - **Example:** Consider a simple database with a table named employees that contains information about employees, such as employee_id, name, and salary.

```sql
-- Declarative Approach
SELECT employee_id, name, salary FROM employees;
```

  - This query does not specify how the database system should perform the retrieval and it abstracts away the implementation details. The database system takes care of executing the query.
- **Advantages:**
  - Declarative programming can be simpler and more straightforward than imperative programming.
  - Declarative code is often easier to read and understand.
  - Declarative code maintenance is simple.
- **Disadvantages:**
  - Declarative programming can be less precise than imperative programming.
  - It can be less performant than imperative programming, as it can involve more steps to accomplish a particular task.
- Declarative programming is divided into three broad categories:

- **Functional:**
  * It is a declarative programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
  * It has its roots in mathematics and it is language independent.
  * The key principal of this paradigms is the execution of series of mathematical functions.
  * Data are loosely coupled to functions.
  * The function hide their implementation and can be replaced with their values without changing the meaning of the program.
  * Example: Languages like Perl, Java script, Haskell mostly uses this paradigm.
- **Logic:**
  * It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc.
  * In logic programming we have a knowledge base which we know before and along with the question the knowledge base is also given to machine and it produces the result.
  * In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence and machine learning knowledge base is used.
  * In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement.
  * Example: Prolog.
- **Data Driven:**
  * This programming technique is based on data and its movement.
  * Program statements are defined by data rather than hard coding a series of steps.
  * A database program is the heart of a business information system and provides file creation, data entry, update, query and reporting functions. There are several programming languages that are developed mostly for database application.
  * Example: SQL.

Paradigm 1: Declarative Programming Paradigm

- **Features:**
  - Declarative programming is a programming paradigm in which the programmer defines what needs to be accomplished by the program without defining how it needs to be implemented.
  - In simple words, the approach focuses on what needs to be achieved instead of instructing how to achieve it.
  - It is a programming paradigm that focuses on expressing what the desired result should be, rather than providing a step-by-step solution.
  - It is different from an imperative program which has the command set to resolve a certain set of problems by describing the steps required to find the solution.
  - Order of expression or statements or the replication of a statement would not have any impact in declarative programming.

- The declarative programming approach helps in simplifying the programming behind some parallel processing applications.

- **History and Evolution of Declarative paradigm:**
  - The declarative paradigm is influenced by the theory and practice of logic programming, functional programming, and constraint programming.
  - The history of the declarative programming paradigm can be traced back to the early developments in artificial intelligence, formal logic, and database management.
  - Here are key milestones in the history of declarative programming:
  - **LISP(1958):**
    * One of the earliest influences on declarative programming was the development of LISP (List Processing) by John McCarthy in 1958.
    * LISP introduced a symbolic expression language with a focus on expressing relationships rather than execution steps.
  - **Prolog(1972):**
    * Prolog, developed in the early 1970s by Alain Colmerauer and Philippe Roussel, is a logic programming language that plays a significant role in the history of declarative programming.
    * Prolog's core concept is based on a declarative approach to problem-solving using logical inference.
  - **SQL(1970s):**
    * Structured Query Language (SQL) emerged in the 1970s as a declarative language specifically designed for managing and querying relational databases.
    * SQL allows users to express what data they want to retrieve or manipulate without specifying how the database system should perform these operations.
  - **Functional Programming(1970-1980s):**
    * The development of functional programming languages, such as Haskell and ML, contributed to the declarative paradigm.
    * These languages emphasize the use of mathematical functions and immutability, providing a clear separation between function definition and execution.
  - **Constraint Logic Programming(1980s):**
    * The 1980s saw the rise of constraint logic programming languages like CLP(R) and CHIP.
    * These languages extended the declarative paradigm by allowing the specification of relationships among variables using constraints.
  - **Logic Programming and AI(1980-1990s):**
    * Declarative programming became prominent in the field of artificial intelligence, with logic programming languages like Prolog finding applications in knowledge representation and expert systems.
  - **Declarative Web Development(2000s):**
    * The declarative paradigm has gained traction in web development, particularly with the advent of frameworks like React.js and Vue.js.

    - These JavaScript libraries enable developers to describe the user interface in a declarative manner, making it more intuitive and easier to maintain.
  - **Domain Specific Languages (DSLs):**
    - The use of domain-specific languages, which often embrace declarative principles, has become more prevalent in various domains.
    - DSLs enable developers to express solutions in a way that is closely aligned with the problem domain.
  - The history of declarative programming reflects its evolution from early symbolic processing in LISP to the development of specialized declarative languages for various domains.
  - Today, declarative principles continue to influence language design and software development methodologies.
- **Advantages of Declarative paradigm:**
  - Declarative programming can be simpler and more straightforward.
  - It is often easier to read and understand.
  - code maintenance is simple.
- **Disadvantages of Declarative paradigm:**
  - It can be less precise as only it contain what to do than the full step by step methods.
  - It can be less performant than imperative programming, as it can involve more steps to accomplish a particular task.
- **Difference between Declarative and imperative paradigms:**

Imperative Programming	Declarative Programming
We specify **how** to get the desired result by providing detailed instructions.	We specify **what** result we expect from the program.
Imperative programming specifies and directs the control flow of the program.	Declarative programming specifies the expected result and core logic without directing the program's control flow.
The programmer makes the major decisions about how the program works.	The compiler makes the major decisions about how the program works.
It is easy to learn and understand.	The code is clean, readable, and effective.
It uses mutable variables, i.e., the values of variables can change during program execution.	It uses immutable variables, i.e., the values of variables cannot change.
Due to the use of mutable variables, imperative programming often changes the program's state by changing the internal data.	Declarative programming doesn't change the program's state.
Procedural Programming and Object-Oriented Programming are examples of imperative programming.	Functional Programming and Logic Programming are examples of declarative programming.
C, C++, Java, PHP, JavaScript, Python, etc., are programming languages that focus on the imperative paradigm.	SQL, LISP, Scala, Haskell, Prolog, Absys, Alice, etc., are programming languages that focus on the declarative paradigm.

- **Example:**
  - **SELECT product_name, price FROM products WHERE category = 'Electronics';**
  - Here we just declare the desired result (selecting product names and prices from the 'Electronics' category) without specifying the procedural steps.

- **Key Principles of Declarative paradigm:**
  Some key principles of Declartive paradigm are:
  - Object-oriented design
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Higher-order abstractions
  - Immutable data
  - Pure functions
  - Declarative DSLs
  - Loop constructs
  - Mutable variables
  - Error handling

- **Implementation of any basic Declarative paradigm:**
  - **Define the Problem:**
    * Problem: Retrieve a list of customers who made a purchase in the last month.
  - **Identify Data and Relationships:**
    * Relevant Tables: customers, purchases
    * Relationships: Each customer has a unique identifier (customer_id). Purchases are linked to customers through the customer_id foreign key.
  - **Choose Declarative Language:**
    * Here we choose SQL as it is a declarative language commonly used for database management.
  - **Write Declarative Statements:**

```sql
SELECT customer_name, email
FROM customers
WHERE customer_id IN (
    SELECT DISTINCT customer_id
    FROM purchases
    WHERE purchase_date >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
);
```

  - **Utilize Built-in Functions:**
    * SQL provides built-in functions like DATE_SUB for date manipulation.

    - CURDATE for the current date, and IN for filtering based on a list of values use it for our ease.

- **Test the result:**
    - Execute the query against the database and verify the results.
    - Refine the query based on performance considerations or additional requirements.

```
+--------------+-------------------------+
| customer_name| email                   |
+--------------+-------------------------+
| John Doe     | john.doe@example.com    |
| Jane Smith   | jane.smith@example.com  |
+--------------+-------------------------+
```

- Declarative programming is divided into three broad categories:
    - **Functional programming approach**
    - **Logic programming approach**
    - **Data-Driven programming approach** (These are explained in Programming Paradigm section)

Language for Paradigm 1: HiveQL

- **HiveQL:**
    - Apache Hive is a data warehouse implementation on top of Hadoop/HDFS.
    - HDFS is an open source framework that quickly moves data between nodes. It is often used by companies that need to store and manage big data. HDFS can manage large amounts of data using low-cost hardware.
    - SQL query written in Hive is called as HiveQL or HQL.
    - Hive Query Language (HiveQL) is a query language in Apache Hive for processing and analyzing structured data.
    - HQL enables users to write SQL-like queries to interact with and analyze large datasets stored in Hadoop's distributed file system(HDFS).
    - It reuses common concepts from relational databases, such as tables, rows, columns, and schema, to ease learning.
    - Hive supports four file formats which are: TEXTFILE, SEQUENCEFILE, ORC and RCFILE (Record Columnar File).

- **History and Evolution of HiveQL:**
    - **Inspiration:** HiveQL draws inspiration from SQL (Structured Query Language), specifically the SQL syntax used in RDBMS. Most of this was written in Java.
    - **Paradigm used:** Declarative.

* **2007:** Hive was developed at Facebook in 2007 by Jeff Hammerbacher and his team to handle large-scale data processing tasks.It was initially designed to provide a SQL-like interface for querying data stored in Hadoop.
* **2008:** Facebook open-sources Hive, expanding its availability and paving the way for broader community contributions.
* **2009-2010:** Hive becomes integral to the Apache Hadoop ecosystem, integrating with components like HBase and Pig for comprehensive data processing.
* **2009 Onwards:** Continuous evolution of HiveQL with additional SQL-like features and optimizations to enhance its querying capabilities.
* **2010:** Hive becomes an Apache Software Foundation project, solidifying its status as an open-source, community-driven data warehousing solution for Hadoop.
* **2014:** Hive introduces support for ACID transactions, providing enhanced data integrity and consistency in transactional workflows.
* **2014:** Integration of Apache Tez as an execution engine for improved performance, offering an alternative to traditional MapReduce.
* **2015:** Introduction of Hive-on-Spark, allowing users to run Hive queries on the Apache Spark processing engine.
* **2016 Onwards:** Continued enhancements focus on performance, scalability, and compatibility with evolving Hadoop ecosystem components.
* **2016:** Introduction of Hive LLAP (Live Long and Process) in Hive 2.1, addressing latency challenges by providing low-latency query capabilities for interactive analytics.

* HiveQL has played a crucial role in making Hadoop accessible to a broader audience by providing a familiar SQL-like interface.
* Its continuous evolution and integration with emerging technologies in the big data ecosystem have contributed to its significance in the field of data processing and analytics.

– **Features and advantages of HiveQL:**

* Hive is designed for querying and managing only structured data stored in tables.
* Hive is scalable, fast, and uses familiar concepts.
* Schema gets stored in a database, while processed data goes into a Hadoop Distributed File System (HDFS).
* Tables and databases get created first; then data gets loaded into the proper tables.
* Hive supports four file formats: ORC, SEQUENCEFILE, RCFILE (Record Columnar File), and TEXTFILE.
* Hive uses an SQL-inspired language, sparing the user from dealing with the complexity of MapReduce programming. It makes learning more accessible by utilizing familiar concepts found in relational databases, such as columns, tables, rows, and schema, etc.

– **How HQL differs from RDBMS:**

* Hive does not support row-level updates and transactions. All these are supported in RDBMS.

    * Hive follows schema on Read. Hence when we load data in any Hive table, it is just copying the files without any checking or validation. In RDBMS all checks and validations are enforced while inserting data into any table itself.
    * Hive works on the concept of Write Once, Read Many times (WORM) while RDBMS works on Read and Write Many times.

- **Characteristics and features found in HiveQL associated with the declarative programming paradigm:**

    * **HQL Syntax are like SQL:**
        · HiveQL employs a SQL-like syntax, making it accessible and familiar to users with relational database query experience.
        · **Declarative aspect:** Users express what data they want to retrieve or manipulate without specifying the detailed procedural steps.

    * **DDL AND DML:**
        · HiveQL supports both DDL and DML operations, allowing users to define and manipulate schema structures as well as query and analyze data.
        · **Declarative aspect:** Users declare the desired structure or outcome, and Hive determines the execution details.

    * **Table Creation and Schema Inference:**
        · Users can create tables and infer schema from data, providing a level of flexibility and abstraction.
        · **Declarative aspect:** The user declares the desired structure through the CREATE TABLE statement, and Hive infers schema details from the data.

    * **Set-Based Operations:**
        · HiveQL supports set-based operations similar to traditional relational databases, allowing users to perform operations on entire sets of data.
        · **Declarative aspect:** Users express the desired operations on sets of data without specifying individual data manipulation steps.

    * **Join and Aggregation Capabilities:**
        · HiveQL supports various types of joins and aggregation functions for data analysis and reporting.
        · **Declarative aspect:** Users declare the desired outcome of joins and aggregations without detailing how these operations should be executed.

    * **Integration with Hadoop Ecosystem:**
        · HiveQL integrates seamlessly with the broader Hadoop ecosystem, allowing users to leverage features like HBase, Pig, and Spark.
        · **Declarative aspect:** Users interact with different components of the Hadoop ecosystem declaratively, expressing their intentions without dealing with low-level details.

    * HiveQL, while adopting a SQL-like syntax, aligns with the declarative programming paradigm by allowing users to express their intentions without specifying the intricate details of how those intentions are fulfilled.

- **Example Query using HiveQL:**
  - Almost all the queries of HiveQL are similar to SQL queries only some are different.
  - Here is the basic HiveQL query to create a simple table named employees with columns id, name, and salary.
  - **Example:** CREATE TABLE employees (id INT, name STRING, salary DOUBLE);
  - This query will create a table named employees with three columns.
- **Limitations of HQL:**
  - Hive is not designed for the OLTP (Online transaction processing). We can only use it for OLAP(online analytical processing).
  - It does not offer real-time queries.
  - It provides limited subquery support.
  - Latency of HQL is generally very high.

Paradigm 2: Scripting

- **Features:**
  - Scripting is a programming paradigm that emphasizes the automation of tasks by using scripts, which are sequences of commands or instructions.
  - The focus is on specifying what actions or operations need to be performed rather than providing detailed step-by-step instructions.
  - Scripting languages are often interpreted rather than compiled, allowing for quick development and execution.
  - Scripting is commonly used for tasks such as automation, system administration, and rapid prototyping.
- Scripting language is a type of language which is used to implement scripting approach.
- **Scripting Language:**
  - A Scripting language (aka scripting, or script) is a series of commands that can be executed without the need for compiling.
  - While all scripting languages are programming languages, but not all programming languages are scripting languages.
  - PHP, Perl, and Python, Lua are common examples of scripting languages.
  - Scripting languages uses an interpreter to translate commands and are directly interpreted from source code, not requiring a compilation step.
  - Other programming languages, on the other hand, may require a compiler to translate commands into machine code before it can execute those commands.
- **Is Scripting equals to Coding?**
  - Scripting is not the same as coding, but these two are very similar.

* Though both are used in the backend of websites and applications, there are key differences when comparing a scripting language vs a programming language. The main difference lies in how they are each used.
* A programming language allows you to create a new program.
* But a scripting language allows you to provide instructions for a program that already exists.
* Scripting provides functionality, while coding provides structure.

– **History and Evolution of Scripting:**

* **The Early Days of Scripting(1950s-1960s):**
  · Scripting began with punch cards submitted to mainframe operators during the non-interactive batch processing era.
  · IBM's Job Control Language (JCL) is recognized as one of the first scripting languages.
  · While functional, early scripting languages had slow response times, often taking at least a day to produce results.
* **Emergence of Interactive Time-Sharing Systems(1960s):**
  · Interactive time-sharing systems, such as MULTICS, introduced the concept of scriptable shells.
  · UNIX, developed by Bell Labs programmers, innovated Unix shells with the ability to chain program outputs as inputs, enabling complex tasks in one line of shell code.
  · Other Unix scripting languages like AWK and Sed emerged for text manipulation.
* **Evolution of Interactive Shells(1960s-1970s):**
  · Interactive shells were developed in the 1960s for remote operation of time-sharing systems.
  · TRAC language by Calvin Mooers introduced command substitution, allowing embedding of commands in scripts.
  · Stuart Madnick at MIT created EXEC, a scripting language for IBM's CP/CMS in 1966.
  · Multics incorporated an offshoot of CTSS RUNCOM, known as RUNCOM, and eventually transitioned to EXEC 2 and REXX.
  · These all boosted the popularity and the need for the development of advanced scripting languages.
* **Perl and the Web Application Boom(1987):**
  · Perl, invented by Larry Wall, was introduced as a major scripting language in 1987.
  · It became popular during the World Wide Web boom of the '90s for creating web applications.
  · It paved the way for subsequent scripting languages like Python and Ruby to gain their popularity.
* **General-Purpose Scripting Languages Emerge(1990s):**
  · Tcl and Lua were designed in the 1990s as general-purpose scripting languages with the ability to be embedded in any application.
  · Visual Basic for Applications (VBA) emerged as a scripting language providing robust integration with system automation facilities.

· The embedding of general-purpose scripting languages became a trend, relieving developers from the need to create new languages for each application.

* **Integration in Software and Web Browsers(2000s-2010s):**
  · In the modern era, software often incorporates multiple scripting languages to enhance functionality.
  · Web browsers, such as those in the modern era, offer scripting languages like JavaScript (ECMAScript) or XUL for browser control and extension development.
* The development of general-purpose scripting languages in the 1990s marked a significant phase in scripting history, offering versatility and integration capabilities. This integration trend continued into the modern era, influencing the use of multiple scripting languages in software and web development.

– **Some of the Key features of scripting:**
* Scripting languages can be directly embedded into an application or in a variety of environments.
* They are quite easy to learn and use. For example, JavaScript and PHP are the easiest scripting languages.
* They provide all of the basic features common to modern programming languages, such as powerful variable types, basic operations (addition, subtraction, etc.), standard control statement, functions, etc.
* The scripted code is not usually compiled. It is simply interpreted by the interpreter at runtime.
* Scripting languages are free and open source (for example, JavaScript, Python, etc.) that means a programmer can have full control to view and edit it. As scripting languages are open source, anyone from anywhere around the world can contribute to their development.
* Most scripting languages are dynamic and weakly typed, but not all of them use these policies. For example, Python is a popular scripting language that is considered as dynamic and strongly typed language.
* The syntax of scripting language is familiar and similar to the syntax of other programming languages such as C, C++, or Java.
* Scripting languages also provides object-oriented functionality like Encapsulation, Inheritance and Polymorphism.
* They allow the dynamic loading and unloading of pieces of code that make it easy to override the functionality of a program.
* They also provide exception feature like other programming languages.
* Scripting languages require less memory from the system because they are not compiled, are interpreted.
* Scripting languages portable and cross-platform, as they can run on a remote server or in the visitor's web browser on any operating system. Therefore, they do not require any additional application to execute the code.
* Scripting languages have been designed and developed with security in mind. They do not allow any illicit operations to be performed from another script. Therefore, they are safe and secure.

- **Implementation of any basic scripting**
  - **Define the Task:**
    - Problem: Rename all text files in a directory by replacing "old_" with "new_" in their filenames.
  - **Write Script:**

```
import os

for filename in os.listdir('.'):
    if filename.endswith('.txt'):
        new_name = filename.replace('old_', 'new_')
        os.rename(filename, new_name)
```

    - Utilize any scripting language to iterate through files and perform the renaming.
  - **Execute and Test:**
    - Run the script and verify that the files are renamed as expected.
- **Advantages of Scripting:**
  - Scripting language enhances the visual appeal of online sites.
  - It is easy to learn and write in comparison with other languages.
  - It is an open-source platform that makes capable of users to view and edit the script when required.
  - As compared to an actual program, it is much faster to develop.
  - It is particularly efficient since it uses a small amount of data structures and variables.
  - Scripting languages contain different libraries, which are used to create new applications in web browsers.
- **Disadvantages of Scripting:**
  - Scripting languages did not compile the file and interpret it directly, which need to install an interpreter or separate program by the users before running the script.
  - The main disadvantage of scripting languages is that they are slower than compiled languages or programming languages because the interpreter in scripting languages read and analyze each statement line by line during the execution.
  - Every time the interpreter detects an error during the execution of program, it stops further execution until the error gets sorted out.
- **Types of scripting:**
  - There are two main types of scripting languages: **server-side and client-side.**
  - **Server-side scripting languages:**
    - The term server-side scripting language refers to those that run off a web server.
    - Since it performs from the back-end side, the script is not visible to the visitor. so, it is a more secure approach.
    - They are often used to create dynamic websites and platforms, handle user queries, and generate and provide data and others.

    - A famous example of server-side scripting is the use of PHP in WordPress.
    - Examples of server-side scripting languages: PHP, Python, Node.js, Perl, and Ruby.
  * **Client-side scripting languages:**
    - Unlike the above, client-side scripting languages run off the user's browser.
    - It is usually performed at the front end, which makes it visible to visitors and makes it less vulnerable to exploits and leaks.
    - Since it runs locally, they usually provide better performance and, therefore, do not strain your server.
    - Examples of client-side scripting languages: HTML, CSS, jQuery, and JavaScript.

Language for Paradigm 2: Lua

- **Lua:**
  * Lua is a scripting language that supports multiple programming methods, including procedural, object-oriented, functional, and data-driven programming.
  * It is used for many applications, including games, web applications, and image processing.
  * It is considered fast and easy to learn and use.
  * It has a C-like syntax, but is dynamically typed. It also features automatic memory management and garbage collection.
  * It is named after the moon and was designed by a team of computer scientists in Brazil in 1993.
  * It is most useful for people who want to add new functionality to an existing game, website, or application that allows Lua code to extend it.
  * For example, the mobile payment app Venmo and the game Angry Birds were both made using Lua.
- **History and Evolution of Lua:**
  * **Inspiration:** Lua draws inspiration from Simple Object Language (SOL) and Data-Entry Language (DEL). It is mostly written in C.
  * **Paradigm used:** Multi-paradigm and majorly used for scripting.
  * **Trade Barriers in Brazil(1977-1992):**
    - Brazil implements strong trade barriers for computer hardware and software.
    - Tecgraf clients face restrictions on buying customized software from abroad.
  * **Tecgraf's Need for Custom Tools(1992-1993):**
    - Tecgraf, part of the Pontifical Catholic University of Rio de Janeiro, develops basic in-house tools.
    - Trade barriers push Tecgraf to create solutions domestically.
  * **Predecessor Languages - SOL and DEL(1992-1993):**
    - SOL (Simple Object Language) and DEL (data-entry language) developed at Tecgraf.

  - Lack of flow-control structures in SOL and DEL sparks the need for a more powerful language.

* **Contenders in Choosing a Scripting Language(1993):**
  - Tecgraf explores options for a scripting language.
  - Tcl is considered but has unfamiliar syntax.
  - LISP and Scheme are excluded due to unfriendly syntax.
  - Python is in its infancy.
  - Decision to develop an in-house scripting language.
* **Design Goals of Lua 1.0(1993):**
  - Lua 1.0 designed to avoid cryptic syntax and semantics.
  - Emphasis on accessibility for non-professional programmers.
  - Highly portable implementation for diverse computer platforms.
  - Lua named after SOL, meaning "Moon" in Portuguese.
* **Influences on Lua Syntax:**
  - Control structures borrowed from Modula.
  - Influence from CLU, C++, SNOBOL, AWK, LISP, and Scheme.
  - Lua's syntax shaped by various languages and paradigms.
* **Semantics Evolution of Lua:**
  - Lua's semantics evolve over time.
  - Increasing influence from Scheme.
  - Paved the way for the foundation of anonymous functions and full lexical scoping.
* **Licensing Changes of Lua:**
  - Versions of Lua before 5.0 released under a license similar to the BSD license.
  - Lua version 5.0 onwards adopts the MIT License.
  - Therfore, it maintains an open-source nature.
* Lua's history is marked by its emergence as an in-house solution at Tecgraf, evolving into a powerful scripting language influenced by diverse paradigms. Its simplicity, portability, and open-source principles made it popular across various application domains.

– **Key features and advanatages of Lua:**

* The compact C implementation makes Lua fast and memory-efficient. The interpreters and JIT compilers produce highly optimized bytecode.
* It's code can be built and run across various platforms like Windows, Linux, macOS, iOS, Android etc. This flexibility is powered by its ANSI C codebase.
* It is implemented as a library rather than a standalone interpreter. This allows it to be deeply integrated and heavily customized within host applications.
* It's semantics are extensible with mechanisms like metatables. Numerous modules and libraries extend the core language capabilities.
* It uses an easy to learn procedural syntax with advanced construct like first-class functions, closures and prototypal inheritance.
* It features incremental garbage collection which makes memory management seamless.

– **Comparision of Lua with other scripting language python:**

Basis of Comparison	Python	Lua
Language	Python is a widely used, powerful, high-level scripting language that is interpreted. It is also one of the most popular scripting languages.	Lua is a high-level scripting language that may be used for a variety of purposes, is flexible, and is very lightweight.
Inheritance	It allows classes to be created using inheritance, in addition to supporting the inheritance concept itself.	It does not support things like classes and inheritances like other programming languages do.
Features	It features an exception handling system that can be used to build applications that are more reliable.	The feature of handling exceptions is missing in Lua.
Speed	Python is slow in speed, when compared to Lua.	When compared to Python, it is faster in speed.
Community	It has a sizable community and excellent community support.	Because it is newer than Python, it lacks a huge community and strong community support.

– **Characteristics and features found in Lua associated with the scripting programming paradigm**

* **Dynamic Typing:**
  · Lua is dynamically typed, meaning variable types are determined at runtime rather than compile-time.
  · This flexibility is a common trait in scripting paradigm, allowing for rapid development and ease of use.
* **Interpreted:**
  · Lua is an interpreted language, which means it doesn't require a separate compilation step.
  · Scripts can be executed directly by an interpreter, enabling quick testing and modification of code.
* **Garbage collection:**
  · Lua has automatic garbage collection, handling memory management without explicit developer intervention.
  · This helps simplify the scripting process and reduces the risk of memory-related errors.
* **Embeddable:**
  · Lua is designed to be easily embedded into other applications, making it an excellent

choice for scripting within larger systems. It provides a simple and lightweight API for integration.

* **Extensibility:**
  - Lua is highly extensible, allowing developers to add custom functionality and modules.
  - This makes it suitable for creating domain-specific languages or adapting the language to specific project requirements.
* **Portability:**
  - Lua is designed to be portable across different platforms, making it suitable for scripting in various environments.
  - This portability is essential for scripting languages to be adaptable to diverse systems.
* These are the characteristics and features found in Lua associated with the scripting paradigm.

– **Example program using Lua:**

```lua
-- Simple Lua Script
local name = "Lua User"
local age = 25

print("Output:")
print("Hello, " .. name .. "!")

local result = age * 2
print("Twice your age is: " .. result)
```

```
Output:
Hello, Lua User!
Twice your age is: 50
```

– **Applications of Lua:**

* Game development
* 3D Animation
* Adobe photoshop lightroom
* Apache HTTP server for request process
* Hard disk recorder

– **Limitations of Lua:**

* The all codes of Lua are not easily available, some of them need to coded manually.
* It has a very small community of users, still relatively unknown next to Perl and Python languages.
* The Lua does not support inheritance in coding while we have to use the Meta table to use inheritance.
* It is rarely used for standalone programming language, usually it is used as embedded scripting language for individual programs.

Analysis

- **Declarative paradigm:**
  - **Strengths:**
    - Declarative programming can be simpler and more straightforward than any paradigms, as it focuses on describing what should happen rather than how it should happen. Example: SQL commands which was discusses earlier.
    - Declarative code is often easier to read and understand, as it contains codes with more implementation details.
    - In declarative programming optimization is easy as its implementation is controlled by an predefined algorithm.
    - Using the declarative programming style, programmers can make code that they can use for different purposes in the future.
  - **Weakness:**
    - In declarative programming, it is tough to personalize codes as per requirement as it depends on the implementation of an algorithm. Thus, it is very difficult to modify the programs.
    - Sometimes, declarative programming becomes very difficult to understand for beginners.
    - It is less efficient than imperative programming, as it can involve more steps to accomplish a particular task.
    - Debugging declarative programs can be more challenging than debugging other programs as tracing the flow of execution can be less straightforward, making it harder to identify and fix bugs.
  - **Notable features:**
    - **Specifies style** - It focuses on expressing the desired result, allowing us to declare what needs to be done rather than specifying the step-by-step instructions on how to achieve it.
    - **Improves code readability** - It's code tends to be more readable and expressive. By removing the low-level details of implementation, the code becomes more closer to natural language making it easier to understand.
    - **Provides more abstraction** - Declarative languages provides higher levels of abstraction, allowing developers to work with concepts and entities at a more abstract level.
    - **Uses immutable data** - It increases the use of immutable data structures, rather than modifying existing data, this results in creating new data structures, which can lead to better program stability and easier reasoning about code.
- **HiveQL:**
  - **Strengths:**
    - HiveQL is easy to use application for both beginners and experts in programming. Anyone who is familiar with SQL can work with Hive easily.
    - It supports various types of file formats such as textfile, ORC, Parquet, LZO Compression, etc.
    - It is used to manage the very large datasets that are stored in the Hadoop Distributed File System.

* **Weakness:**
  - Hive Query Language does not support the transaction processing feature.
  - Latency of HiveQL is generally very high.
  - HiveQL uses table form to store data which always process structured data. Even if the unstructured data is written using SQL queries, HiveQL cannot support them.
* **Notable features:**
  - **Syntax like SQL** - HiveQL has a syntax similar to SQL, making it accessible and familiar to users who are already good in SQL. This helps in quickly transitioning users from traditional relational databases to Hive.
  - **DML and DDL support** - It supports standard SQL operations for defining and manipulating data. This includes statements for creating and altering tables (DDL) as well as querying and updating data (DML).
  - **Large Built-in functions** - It provides a variety of built-in functions for performing operations on data like mathematical operations, string manipulation, date and time functions, etc.
  - **Supports subqueries** - It supports subqueries, allowing us to nest queries within other queries to express more complex logic and perform advanced data manipulations.

– **Scripting paradigm:**
* **Strengths:**
  - Scripting languages are embeddable, that mean they can be easily embedded into existing components or applications such as Microsoft Word, etc.
  - As most scripting languages use an interpreter so no executable files are stored. Therefore, they require less memory.
  - They are portable, that means they can run on any operating system.
  - The coding of scripting language is easy to use and learn so it is very beginner friendly language.
* **Weakness:**
  - The main disadvantage of scripting languages is that they are slower than compiled languages because the interpreter in scripting languages need to read and analyze each statement line by line during the execution.
  - Every time the interpreter detects an error during the execution of program, it stops further execution until the error is resolved.
  - It can lack the coding standards that are followed by the developers.
* **Notable features:**
  - **Provides automation** - Scripting languages are commonly used for automating repetitive tasks, system administration, and workflow automation. They allow users to create scripts that perform a series of commands or tasks with less effort.
  - **Platform independent** - Many scripting languages are platform-independent(i.e, The scripts that are written once can be executed on different operating systems without modification).
  - **Used as Glue code** - They are often used as "glue code" to integrate different components in application. They can facilitate communication between various software components written in different languages.

- **Easy to understand** - The syntax of scripting language is similar to the syntax of other programming languages such as C, C++, or Java. So it is easy to understand for all the developers.

– **Lua:**
  * **Strengths:**
    - Lua has a fast and efficient interpreter, which contributes to its high performance. It is often used in performance-critical applications such as game development.
    - It has automatic memory management through garbage collection, it simplifies the memory management for developers and helps to prevent memory leaks.
    - It's small size and high performance make it easy to integrate into applications.
    - It supports all versions of Unix and Windows, mobile devices, embedded microprocessors, and IBM mainframes.
  * **Weakness:**
    - Lua's standard libraries are relatively minimal compared to some other languages.
    - Limited error handling support can lead to longer debuging times to identify the exact errors and correct it.
    - All variables in lua are created as global variables (global scope), which can lead to errors in variable assignments.
    - It is rarely used for standalone programming language, it is used as embedded scripting language for individual program.
  * **Notable features:**
    - **Embeddability** - Lua is designed to be easily embedded into other applications, providing a simple and clean API for integration. This feature makes it a popular choice for extending the functionality of existing software.
    - **Portability** - It is highly portable and can run on various platforms, including Windows, Linux, macOS, and embedded systems. This makes it a best choice for cross-platform development.
    - **Lightweight** - It is designed to be lightweight and fast, with a small memory usage. This makes it suitable for embedded systems and environments with limited resource constraints.
    - **Garbage collection** - It includes automatic memory management through a garbage collector, reducing explicit memory management. It helps to prevent memory leaks.

Comparison

– **Declarative paradigm vs Scripting paradigm**
  * **Similarities:**
    - **Readability:**
    - Both paradigms emphasizes readable code.
    - Declarative focuses on expressing what should happen.
    - scripting languages focuses for human-readable syntax like natural language.

· **Abstraction:**

· Both Declarative and Scripting paradigm provides a level of abstraction by simplifying complex operations for users.

· **User friendly:**

· Both paradigms strive to make programming accessible, either by simplifying code (declarative) or using an easily understandable syntax (scripting).

∗ **Differences:**

· **Focus of the language:**

· Declarative focuses on "what" needs to be achieved without specifying the step-by-step process.

· Scripting focuses on automation and scripting tasks, detailing "how" a task is accomplished through a series of commands.

· **Efficient execution:**

· In declarative codes are generally optimized through predefined algorithms, allowing for efficient execution.

· In scripting the execution speed is slower due to interpretation of commands line by line.

· **Portability:**

· In declarative, code may be less portable compared to scripting

· In scripting, is more platform independence, their codes are easily portable on multiple platforms.

· **Use cases:**

· Declarative programs are commonly used in database queries, configuration files, etc.

· while scripting files are used for automation, integration, and glue code to connect different components.

– **HiveQL vs Lua:**

∗ **Similarities:**

· **Abstraction:**

· Both HiveQL and Lua provide a level of abstraction, allowing users to work at a higher conceptual level without dealing with low-level details.

· Also, abstraction enhances code readability, maintenance, and reduces complexity in different domains.

· **Scripting capabilities:**

· Scripting features provide flexibility and are valuable for various applications, from data processing to game development.

· So, both HiveQL and Lua have scripting capabilities, enabling users to automate tasks and execute sequences of commands.

* **Differences:**
    - **Domain:**
    - HiveQL is specifically designed for querying and managing large-scale distributed data in the context of Apache Hive, often used in big data processing and analytics.
    - While, Lua is a lightweight scripting language used in a variety of domains, including game development and embedded systems.
    - **Scalaing and optimization:**
    - HiveQL is Optimized for distributed data processing in large-scale environments.
    - While, Lua is efficient for various applications, it is not optimized for large-scale distributed processing like HiveQL.
    - **Ecosystem:**
    - HiveQL is a part of the Apache Hive ecosystem, integrating with Hadoop for distributed computing.
    - While, Lua is Known for its embeddability and extensibility, used as a scripting language in applications and games.
    - **Uses:**
    - HiveQL is used to write queries to analyze and process large datasets stored in distributed environments.
    - While, Lua is a Scripting language used to write game logic, configure applications, and embedding scripts in various softwares.

Challenges Faced

- 1) Too much resources for declarative paradigm and could not find the exact thing to sort and put into this report.
    * Solved this by just checking through 3-4 resources and finding the most similar contents in all these and select that topic as a choice to put into my report.
- 2) Too much of resources but can't find whether their resource content are validated and checked.
    * Took the content given in such like unpopular sites and done some checks with other similar websites and some AI tools to confirm whether their content is true to their definition or not.
- 3) Very low resources for Lua and HiveQL language.
    * I referred to multiple quora and reddit sites and collect info about it and finally used collective informations in this report.
- 4) As i refered more materials and collected just some points it is very difficult sometimes to make it structured with grammatical errors.

    - Solved this by using Grammarly and Grammar AI tools.
- 5) In analysis and comparision part as initially it was handwritten by me the points and its structure was very poor.
    - So i again compared many resources used many AI tools to refine my writings and made my analysis and comaprision part crisp.
- Too much time consuming to make a report in overleaf. These are some of the challenges, I faced.

Conclusion

- Quick summary obtained by exploring the programming paradigms like declarative and scripting as well as the languages like HiveQL and Lua are:
    - **Declarative vs Scripting:**
        - **Similarities:**
        - Both paradigms prioritize readable code, abstraction, and user-friendliness.
        - **Differences:**
        - Declarative focuses on "what," scripting on "how."
        - Declarative is more efficient, but less portable while scripting is less efficient but portable.
        - Declarative suits database queries while scripting for automation and integration.
    - **HiveQL vs Lua:**
        - **Similarities:**
        - Both languages offer abstraction and scripting capabilities.
        - **Differences:**
        - HiveQL is used for big data processing while Lua is used for low level games, embedded systems, etc.
        - HiveQL is optimized for large-scale distributed processing while Lua is efficient but not optimized for large-scale.
        - HiveQL integrates with Hadoop while Lua is embeddable and extensible.
        - HiveQL used to query large datasets while Lua for game logic and software scripting.
- In conclusion, Declarative and scripting paradigms share foundational principles but differ in focus and efficiency. Similarly, HiveQL and Lua offer abstraction and scripting features but differ in domains, scalability, ecosystems, and use cases.

References

- https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/
- https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
- https://www.info.ucl.ac.be/ pvr/VanRoyChapter.pdf
- https://www.geeksforgeeks.org/difference-between-imperative-and-declarative-programming/
- https://www.techtarget.com/searchitoperations/definition/declarative-programming
- https://en.wikipedia.org/wiki/Declarative$_{p}rogramming$
- https://www.geeksforgeeks.org/difference-between-sql-and-hiveql/
- https://www.scaler.com/topics/hadoop/hadoop-hiveql/
- https://medium.com/@sujathamudadla1213/what-is-hiveql-abce2699cd36
- https://en.wikipedia.org/wiki/Scripting$_{l}anguage$
- https://mrcet.com/downloads/digital$_{n}otes/CSE/Paradigms$
- https://rockcontent.com/blog/scripting-languages/
- https://www.javatpoint.com/what-is-a-scripting-language
- https://www.lua.org/
- https://www.tutorialspoint.com/lua/index.htm
- https://www.bmc.com/blogs/lua-programming-language/
- https://www.lua.org/pil/1.html
- https://www.codecademy.com/resources/blog/what-is-lua-programming-language-used-for/
- https://www.linkedin.com/pulse/lua-programming-language-bikash-kumar-sundaray