1/31/2023

SCRIPTING LANGUAGES

CSC 331 Project – Group 3

Group 3 Members

S/N	NAME	MATRIC NUMBER	ROLE
1	Okafor Lisa	214901	Chapter One – Introduction of scripting languages
2	Soyoye Kehinde	214916	
3	Olalere Khadijat	222502	
4	Kayode Peter	208077	Chapter Two – Comparison of Scripting Languages
5	Onasoga Oluwapelumi	214909	
6	Adeoti Warith	214851	Chapter Three – A Case Study on Python Scripting Language
7	Odulate Oluwatobi	214893	
8	Adim Solomo	222455	Chapter Four – The Applications of Scripting Languages
9	Denedo Oghenetega	214873	Chapter Four – The Applications of Scripting Languages and Organized all meetings held by the group
10	Kareem Mustapha	214883	Prepared the Presentation Slide
11	Animasaun Ahmad	214863	Reviewed Chapter One and formatted the report document.

Dr. Joseph Akinyemi

LECTURER IN CHARGE

TABLE OF CONTENT

CHAPTER ONE

1.1 What is a Script?

Scripts are programs often used to automate routine tasks, such as data processing and testing, and can be executed in a variety of environments, such as on the command line, in a web browser, or within a larger application. They are typically small and focused, designed to perform a specific task quickly and efficiently. They are written in a scripting language, which allows for fast development and execution times.

1.2 Scripting Languages

Scripting languages are a type of computer programming language that is designed for automating tasks and enabling rapid development of software applications. These languages are known for their ease of use, quick learning curve, and versatility, making them a popular choice for developers of all skill levels.

They are typically interpreted, which means that the code is executed line by line, rather than being translated into machine code and executed. This allows for faster development times, as changes can be made to the code and executed immediately without needing a full recompilation. In contrast, compiled programming languages like C++ or Java must be translated into machine code and executed, which results in slower development times and a more rigid workflow.

Scripting languages are also known for their ease of use and quick learning curve. They are often designed with a focus on readability and simplicity, making it easier for developers to quickly write and execute code. This is why scripting languages e.g. JavaScript, are often the first programming languages that people learn, as they provide a gentle introduction to the world of programming. Due to their high flexibility and versatility, they can be used for writing small, focused scripts that perform specific tasks, or for adding functionality to larger applications, such as automating routine tasks or adding new features. This versatility is one of the key reasons why scripting languages have become so popular.

1.3 Key Features of Scripting Languages?

- Dynamic Typing: Scripting languages are typically dynamically typed, which means that variables can change type during the execution of a script. This allows for more flexible and dynamic programming, as variables can adapt to changing conditions in the script.
- Large Standard Library: They often come with a large standard library of functions and tools, which makes it easier to perform common tasks. This can greatly speed up development times and reduce the amount of code that needs to be written.
- Built-in Data Structures: Scripting languages also come with built-in data structures, such as arrays, lists, and dictionaries, which makes it easier to manipulate and organize data.
- Interactive Interpreter: Many of them come with an interactive interpreter, which allows developers to execute code line by line and see the results in real-time. This is a great way to experiment with code and test ideas, without the need to write and run an entire script.
- Cross-Platform Compatibility: Scripting languages are almost always cross-platform compatible, which means that scripts can be written and executed on multiple operating systems and platforms. This makes scripting languages a great choice for developers who need to write code that will run on a variety of systems.

1.4 History of Scripting Languages

The history of scripting languages can be traced back to the early days of computing when simple scripts were used to automate repetitive tasks on mainframe computers. Over time, scripting languages have evolved and become increasingly sophisticated, with the development of new languages such as Perl, Python, and JavaScript.

Perl, which was first released in 1987, was one of the first widely used scripting languages and remains popular to this day. It is often used for web development, system administration, and data processing. Python, another scripting language that came out in 1991, is a high-level, dynamically typed language widely used in scientific computing, data analysis, and machine learning.

JavaScript, which was first released in 1995, is a scripting language that is predominantly used for web development. It is the primary language for creating interactive and dynamic websites and web applications. JavaScript is supported by all major web browsers and is one of the most popular programming languages in use today.

Scripting languages have come a long way since their early days and are now widely used for a variety of applications, from web development and data processing to scientific computing and game development. With their ease of use, quick learning curve, and versatility, they remain a popular choice for developers and programmers of all skill levels.

CHAPTER TWO

2.1 Interpreted Language vs Compiled Languages

An interpreter is present during the course of the application's execution in interpreted languages. The interpreter is in control at this point in the execution. The interpreter, in essence, creates a virtual machine whose "machine language" is the high-level programming language. More or less, one at a time, the interpreter reads and executes statements from the language. A good source-level debugger could also be a function of the interpreter because the source code is being executed directly. It can also work with languages whose basic program properties, such as variable sizes and types, or even names that designate which variables are meant to be referenced, can vary depending on the input data. All scripting languages are interpreted Languages.

Compiler-based languages, as opposed to interpreter-based ones, first translate a high-level source program into a target program that is equivalent (usually in machine language), and then they go away. The user specifies a random moment in the future for operating system required to run the intended program. The source of control during compilation is the compiler, and the target. When a program is being executed, it is where control is located. So, the compiler is a machine language program, produced through the compilation of another high-level application.

2.2 Scripting Languages vs Other Kind of Programming Languages (JavaScript vs C++)

A good example of a scripting language is JavaScript, which is designed for a runtime system to automate task execution. JavaScript does not require an explicit compilation step. Due to its great level of abstraction, it is also known as very high-level programming languages. A "script," or a brief program created for a particular runtime environment, is supported by scripting languages. In place of being compiled, these are interpreted live. Scripting languages therefore utilize an interpreter rather than a compiler to translate source code into machine code. Thus, a scripting language can automate various settings like application software, websites, text editors, operating system shells, video games, etc.

C++, on the other hand, is a good example of a non-interpreted programming language (compiled language). The generated program is extremely efficient because it is typically translated into machine language, which the system can understand directly. In compilation,

the source code is first preprocessed using a preprocessor, followed by compilation using a compiler, assembly of the compiled source code, linking an object code file, and lastly creation of an executable file.

2.3 Types of Scripting Languages

Scripting languages are categorized into two types, which are:

1. Client-side Scripting Languages and
2. Server-side Scripting Languages

Client-side Scripting Languages

Languages for client-side scripting are executed by the user's browser. It is typically carried out in the front-end, where it is visible to visitors and less prone to vulnerabilities and leaks. As a result, it is frequently employed to create user interfaces and other types of lightweight functionality. Since they operate locally, they typically offer superior performance and do not tax your server. More interactivity is possible, some tasks can be completed without the user's involvement, and database connections to web server-based databases are not supported. The file system in the web browser is inaccessible to these programs. Based on the user's preferences, pages are changed. It can also be used to make "cookies" that the user's computer uses to store information. Examples include JavaScript, jQuery, CSS, and HTML.

Server-side Scripting Languages

Languages that run on a web server are referred to as server-side scripting languages. The visitor cannot see the script because it operates on the back-end. It is a more secure method as a result. They are frequently employed to build interactive websites and platforms, respond to user inquiries, develop and give data, among other things. The use of PHP in WordPress is a well-known instance of server-side scripting. There are also Python, Node.js, Perl, and Ruby as examples.

CHAPTER THREE - A CASE STUDY ON PYTHON

3.1 Brief introduction of Python

Python programming language is a high-level, general-purpose programming language that is easy to learn and use, as well as being interactive and interpreted. It is also a cross-platform and open-source language that can be used as a scripting language.

Python is fundamentally both an interpreted and a compiled language, but since the compilation process happens internally and is not visible to the programmer, we typically refer to Python as an interpreted language.

Python has a number of features, such as:

1. In contrast to the majority of programming languages, python's syntax is simple to understand and use.
2. Python's interactive mode enables users to run interactive tests and debug code fragments.
3. Scripting ability: Python is frequently used as a scripting language (a type of programming that can automate tedious tasks and improve the effectiveness of system administration).
4. Python is a cross-platform programming language, making it compatible with many different operating systems, including Linux, Mac and Windows.
5. Different interpreter flavors, providing different implementations upon which python codes can be executed and as well as other language integration. Examples include Cython, Jython, PyPy, IronPython, RubyPython etc. Cython, which is the C language implementation of the interpreter, is the standard.

3.2 How Python codes are executed

It is common to come across Python files with the '***.py***' extension which contains the source codes in python syntax. Although python is listed as an Interpreted language, the process of execution of python source codes involves compilation and interpretation steps.

When the Python interpreter program encounters a source file, it runs the file through a Lexical Analyzer which provides the input (streams of token) for the second stage called Parsing where syntax analysis is performed building an Abstract Syntax Tree (AST) of the source code.

The AST is converted into an intermediate form of code known as Byte Code and this code is saved in a '***.pyc***' file. This is the compilation step of the execution process.

The corresponding byte code is then sent into Python's virtual machine (PVM), where it is interpreted into machine code and executed. If an error arises in any of the processes, the process is stopped, and the resulting error is reported to the programmer.

To show the compilation step, a simple Python script that converts text to ASCII characters is given below saved as 'text_2_ascii.py'.

```
# Please run this script with python3
# American Standard Code for Information Interchange (ASCII)
# This script convert word/character  to ascii (0-127)

print("This script convert text to ASCII\n")
word = input('Enter the word to convert to ascii: ')
print('\nThe ascii  representation of: {}'.format(word))

for i in word:
        ascii_rep = ord(i)
        print(str(ascii_rep), end=" ")
print("\n")
```

When the file is executed normally, the compilation phase is hidden and done internally. But the compilation phase can be done manually by running the file through the python compiler module. To do this:

We run the command:

```
python3 -m py_compile  text_2_ascii.py
```

Meaning:

python3: version of the python interpreted to use, in this case v3.*.*

-m py_compile: -m is used to import the module while py_compile is the module name)

text_2_ascii.py: name of the python file

Running that command will create a __pycache__ directory which will contain our byte code file (in this case, **__pycache__/text_2_ascii.cpython-38.pyc**).

To run the generated bytecode (compiled code) by the PVM:

Command:

```
python3 __pycache__/text_2_ascii.cpython-38.pyc
```

and using an input of "**HELLO**"

Result:

```
This script convert text to ASCII

Enter the word to convert to ascii: HELLO

The ascii  representation of: HELLO
72 69 76 76 79
```

The bytecode can be viewed directly, by running the command:

"python3 -m dis text_2_ascii.py"

```
5           0 LOAD_NAME                0 (print)
            2 LOAD_CONST               0 ('This script convert text to ASCII\n')
            4 CALL_FUNCTION            1
            6 POP_TOP

  7         8 LOAD_NAME                1 (input)
           10 LOAD_CONST               1 ('Enter the word to convert to ascii: ')
           12 CALL_FUNCTION            1
           14 STORE_NAME               2 (word)
***
```

3.3 Python Lexical Analysis

Lexical analysis involves breaking down source code into tokens. This stream of tokens represents different elements of the source code that are later passed to the parser. The list of tokens recognized by a python lexical analyzer program includes: KEYWORD, IDENTIFIER, OPERATOR, STRING/BYTE LITERAL, NUMERIC LITERAL, INDENT (which represents indentation), DEDENT (dedentation), DELIMITER (also called punctuator or separator), NEWLINE (token for representing logical newline - used in determining line continuation is source codes) etc.

Tokens produced by the lexical analyzer can be used for static code analysis, error detection, and code generation.

3.3 The process of Parsing in Python

The parser is given the streams of tokens produced by the afore-mentioned lexical analysis. By applying a set of grammar rules to the tokens it determines their structure and relationships. Token streams are analyzed in parsing to determine if their syntax is correct.

This can be achieved by the parser module in Python, which is used by simply importing it.

3.4 A simple Python Lexical Analyzer Program

As a result of our research and study on Lexical Analysis, we provide a simple analyzer program written in the python language itself to recognize tokens from python source files. The full source codes of the program can be accessed on this *repository*.

As an example, the program is run on the example source codes given above. The result of execution:

```
./lexer.py text_2_ascii.py
```

```
COMMENT --> # Please run this script with python3
COMMENT --> # American Standard Code for Information Interchange (ASCII)
COMMENT --> # This script convert word/character  to ascii (0-127)
IDENTIFIER --> print
PUNCTUATOR --> (
LITERAL --> "This script convert text to ASCII\n"
PUNCTUATOR --> )
IDENTIFIER --> word
Operator --> =
IDENTIFIER --> input
PUNCTUATOR --> (
LITERAL --> 'Enter the word to convert to ascii: '
PUNCTUATOR --> )
IDENTIFIER --> print
PUNCTUATOR --> (
LITERAL --> '\nThe ascii  representation of: {}'
IDENTIFIER --> format
```

CHAPTER 4

4.1 Pros and Cons of Scripting Languages

There are many pros and cons when deciding whether to use a scripting language for a particular project. Here are some valuable points to consider:

4.1.1 Pros of using Scripting Languages

- They are usually easier to learn and use than compiled languages, so they are a good choice for beginners or people who just need to write a quick script to automate a task on an existing system.
- Scripting languages are often interpreted rather than compiled, so it is easy to write and test code without needing a separate compilation step.
- Scripting languages are often more flexible and powerful than graphical user interface (GUI)-based tools, which can be limited in functionality. i.e. BASH, a scripting language used in UNIX operating systems, can be used to write scripts for advanced operations with the File System that cannot be easily achievable with a GUI.
- Many scripting languages have large libraries of pre-written code, called modules, that can be easily imported and used in a script, saving time and effort.

4.1.2 Cons of using Scripting Languages

- Scripting languages are generally slower than compiled languages, so they may not be suitable for applications requiring a lot of processing power or running in real-time.
- Scripting languages are not as portable as compiled languages, so it may be more challenging to run a script on a different type of machine or operating system without modifying the code.

4.2 Applications of Scripting Language

Web development: Interactive and dynamic websites are made using scripting languages like JavaScript and PHP. For instance, JavaScript is frequently used to make animations, check forms, and respond to user actions like clicks and hovers. PHP is a language for creating server-side scripts frequently used to build backend logic and link websites to databases.

System administration: Commonly used scripting languages for computer and network activities include Python and Bash. For instance, a system administrator might use a Bash script

to automate software installation on numerous devices or a Python script to create backups of every file on a network.

Game development: Video game AI and mechanics are frequently created using scripting languages like Lua. For instance, a game developer might utilize a Lua script to make an enemy AI that can move around a game world and attack the player.

Scientific Computing: Scientific computing and data analysis frequently use scripting languages like Python and MATLAB. For instance, a researcher might use a MATLAB script to carry out intricate calculations and simulations or a Python script to analyze and show massive scientific data.

Automation: Scripting languages can automate repetitive processes like renaming files or completing online forms. For instance, a script could automatically fill out an online form with many entries or rename many files in a folder.

Prototyping: Before putting new concepts into practice in a more feature-rich and high-performance language, fast prototyping and testing are frequently conducted using scripting languages. Before spending a lot of time and money on the development, this can be useful for swiftly evaluating an idea's viability or seeing any potential issues.

In conclusion, scripting languages are valuable tools for various programming tasks. They are generally easier to learn and use than general-purpose programming languages and are well-suited for automating tasks, creating user interfaces, and building programs for the web. Scripting languages are widely used in the development of web applications, making them an essential skill for web developers to learn. The simplicity and flexibility of scripting languages make them a good choice for rapid prototyping and development, and they can significantly improve the efficiency and effectiveness of many programming tasks.

REFERENCES

Bhagat, V. (2022, March 18). *Pros and Cons of Python Programming Language*. Retrieved from pixelcrayons.com: https://www.pixelcrayons.com/blog/python-pros-and-cons/

GeeksforGeeks. (2020, July 10). *Understanding the Execution of Python Program*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/understanding-the-execution-of-python-program/

Gkindex. (n.d.). *Flavours of Python*. Retrieved from gkindex.com: https://www.gkindex.com/python-tutorial/python-flavours.jsp

Kaptur, A. (n.d.). *A Python Interpreter Written in Python*. Retrieved from aosabook.org: https://www.aosabook.org/en/500L/a-python-interpreter-written-in-python.html

Python.org. (2023, February 2). *Lexical Analysis*. Retrieved from python.org: https://docs.python.org/3/reference/lexical_analysis.html

Scott, M. L. (2015). *Programming Language Pragmatics.* Morgan Kaufmann.

TargetTech. (2016, May). *What is scripting language*. Retrieved from techtarget.com: https://www.techtarget.com/whatis/definition/scripting-language

TechTarget. (2021, December). *What is a script*. Retrieved from techtarget.com: https://www.techtarget.com/whatis/definition/script

Wikipedia. (2023, February 1). *Python (programming language)*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Python_(programming_language)