PURDUE UNIVERSITY NORTHWEST

Token Analysis : Python vs. Java

Group 1

Vaishak Balachandra - Keywords and Identifiers
Full Name - Literals and Operators
Full Name - Separators/Delimiters, Comments, and Whitespace

Table of Contents

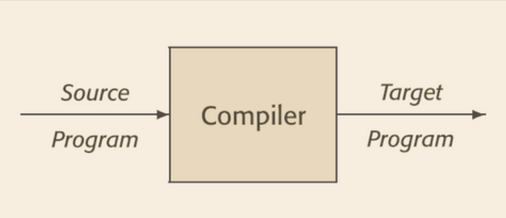
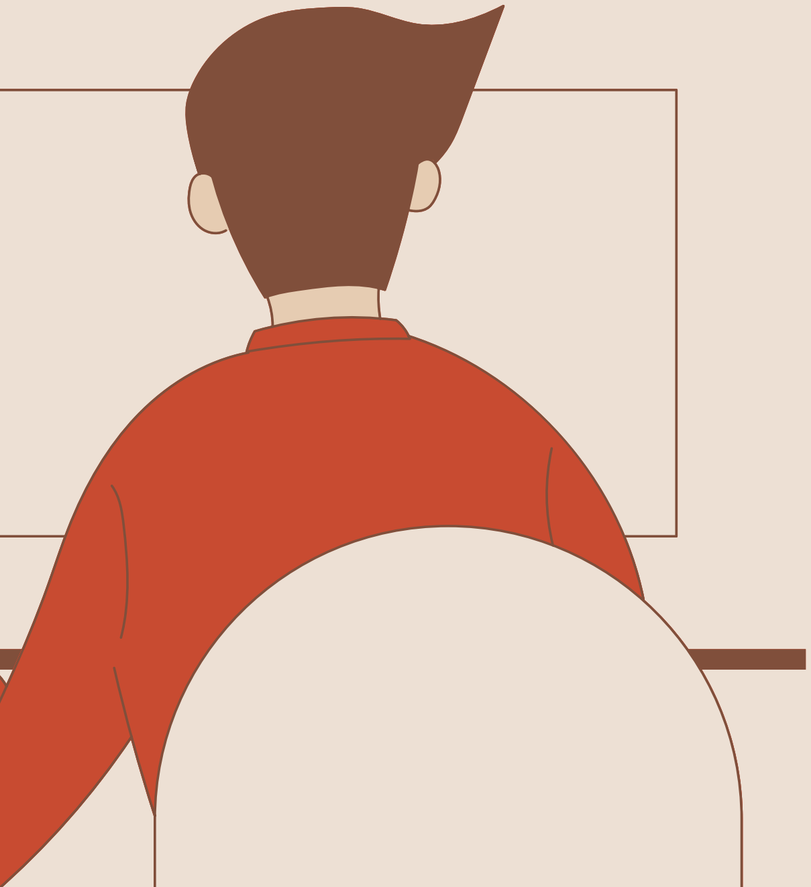


FIG 1. COMPILER ACTION



- **Introduction to Compilers**

  Definition: A compiler is a program that translates high-level source code into machine code.

  Phases of a Compiler:

  - Front-End: Focuses on analyzing and understanding the code.
  - Back-End: Focuses on optimization and generating executable code.

  Importance: Helps convert human-readable code into efficient instructions for computers

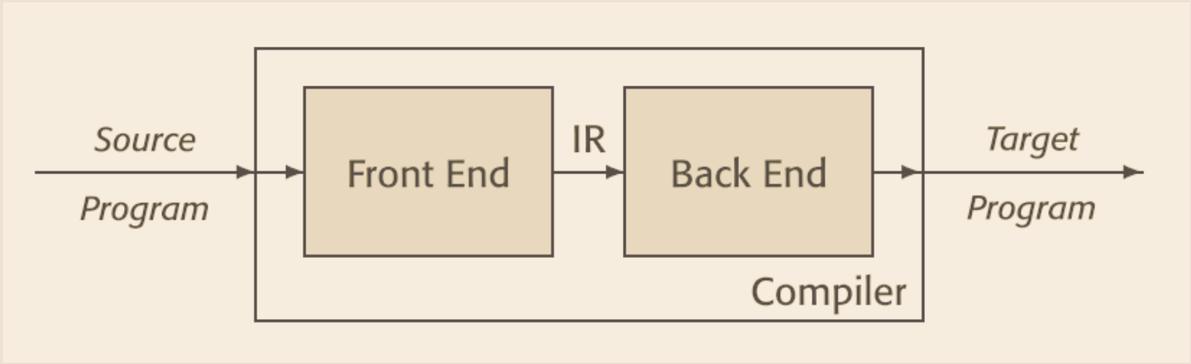
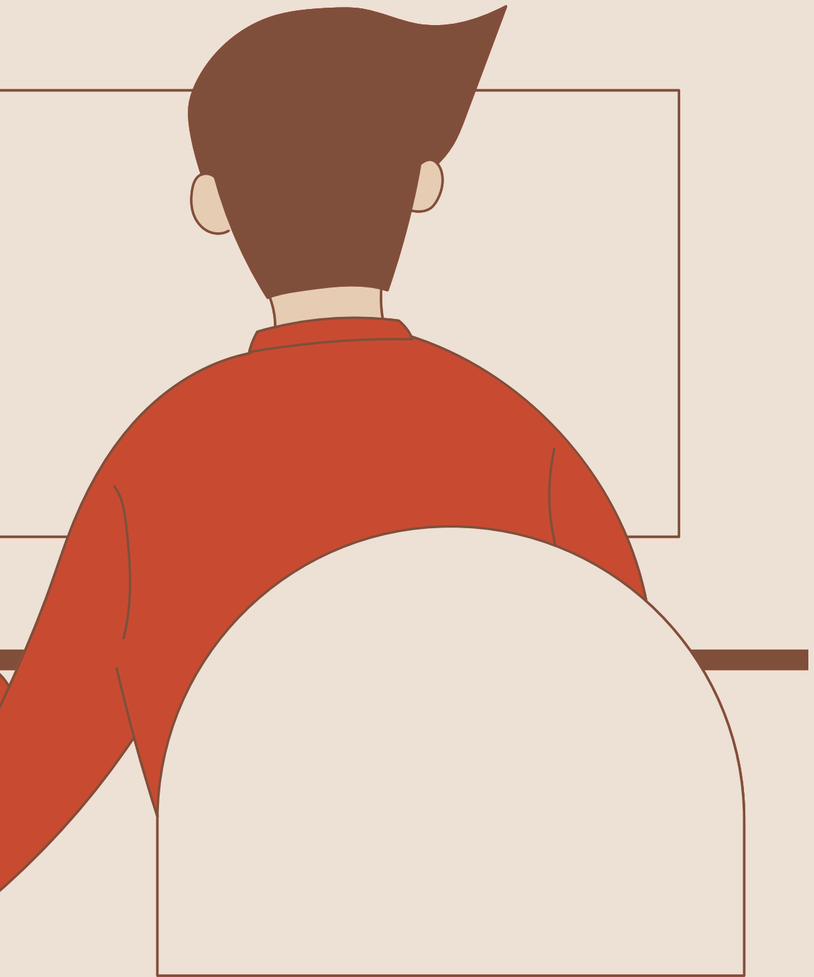


FIG 2. TWO PHASE COMPILER BLOCK DIAGRAM



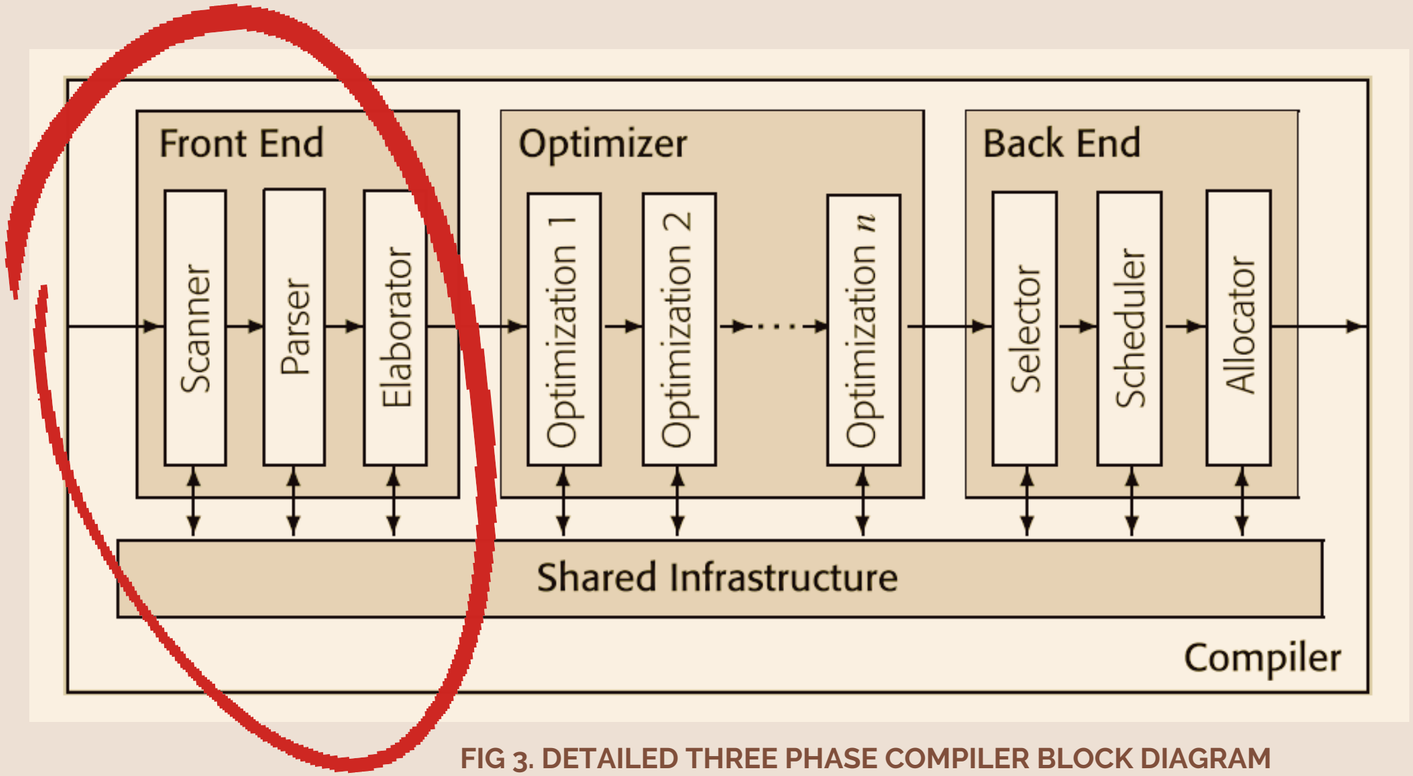
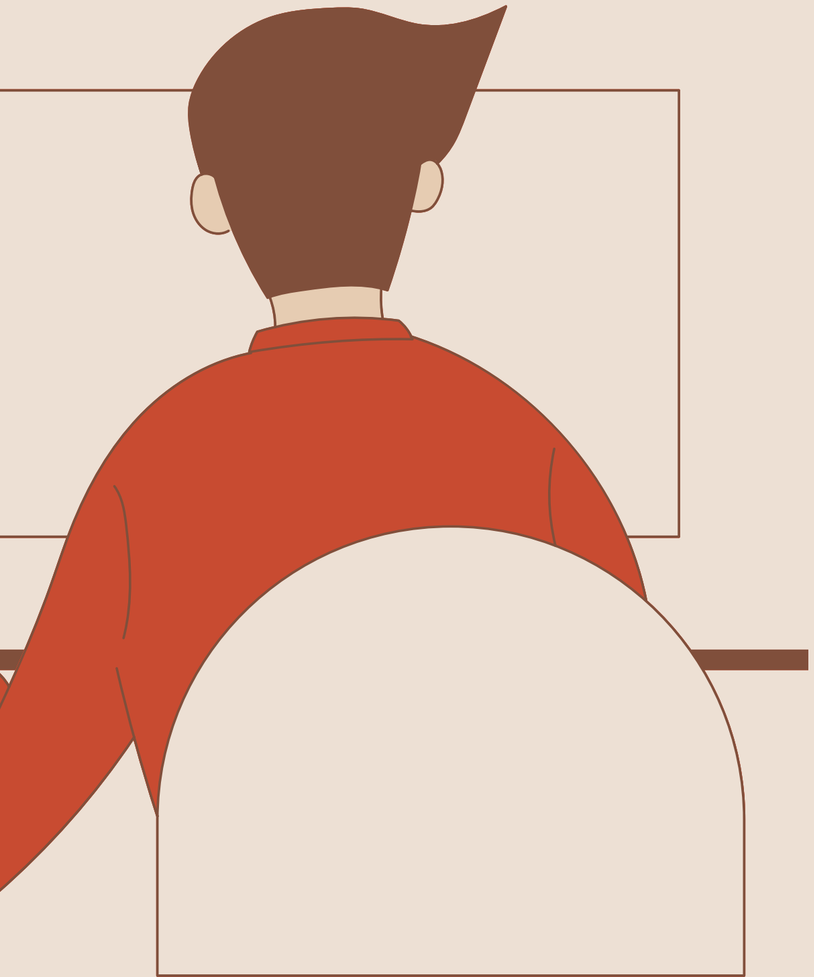


FIG 3. DETAILED THREE PHASE COMPILER BLOCK DIAGRAM

- **The Front-End of a Compiler**

Main Components:

- Scanner (Lexical Analyzer): Converts source code into tokens.
- Parser (Syntax Analyzer): Checks the grammar and structure of tokens.
- Semantic Analyzer: Ensures the meaning of code is valid (e.g., type checking).



- **The Scanner Phase (Lexical Analysis)**

Role of the Scanner:

- Reads the source code character by character.
- Groups characters into meaningful units called tokens.
- Eliminates whitespace and comments.

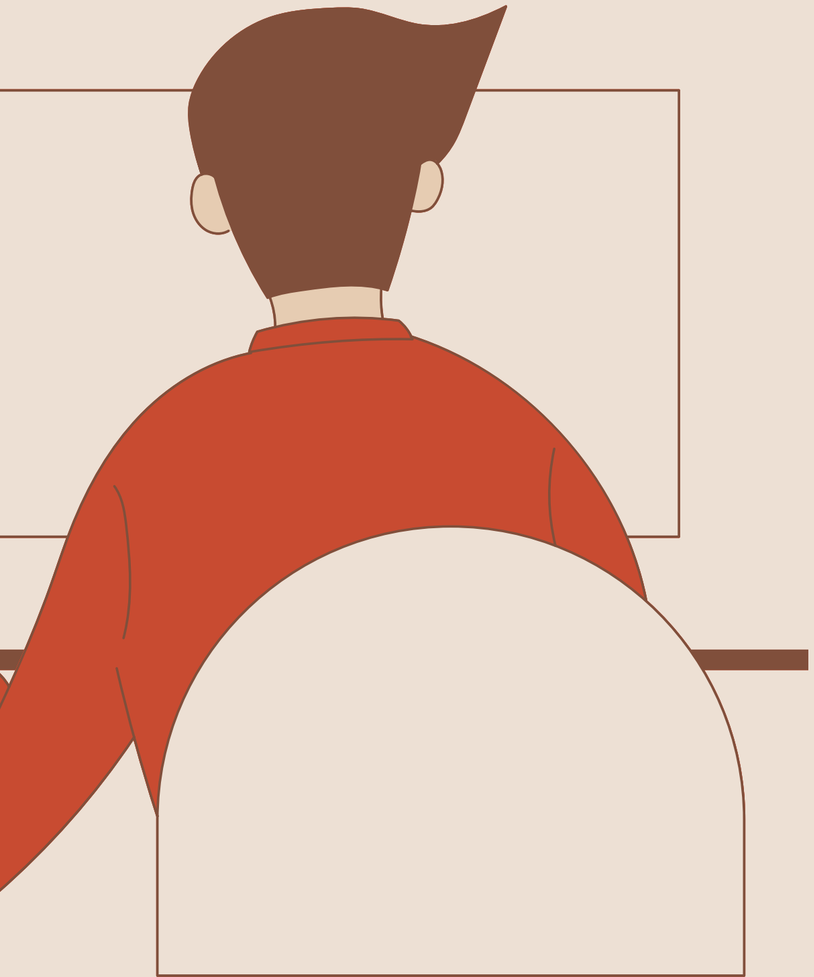
Output: A sequence of tokens passed to the parser.

Example:

Python Source Code:

```
if (x > 0):
    print(x)
```

Tokens: if, x, >, 0, :, print, (, x,).



- **Tokens – Definition and Categories**
  - Definition: Tokens are the smallest units of a program, generated during lexical analysis.
  - Categories of Tokens:
    a. Keywords (Reserved words like if, else).
    b. Identifiers (Variable or function names like x, y).
    c. Literals (Fixed values like 42, "Hello").
    d. Operators (+, -, =).
    e. Separators/Delimiters ({, }, ;).
    f. Comments (Ignored by the compiler).
    g. Whitespace (Used for formatting).

Token Representation:

(token_Category, token)

Part A:
Keywords & Identifiers

KEYWORDS

Definition: Reserved words with predefined meanings that cannot be used as identifiers.

Role: Define the syntax and control flow of a language.

Python:

Total: 35 keywords.

Examples: def, class, lambda, async...

Dynamic, focuses on readability.

TABLE 1. KEYWORDS IN PYTHON

async	continue	while	def	as
await	elif	del	lambda	and
False	else	assert	yield	in
None	for	except	from	is
True	if	finally	global	not
with	return	raise	import	or
break	try	class	nonlocal	pass

Java:

Total: 51 keywords.

Examples: public, static, synchronized, final...

Statically typed, more verbose.

TABLE 2. KEYWORDS IN JAVA

private	extends	new	case	for	while	float	void	try	native
protected	implements	package	continue	goto	boolean	int	catch	abstract	static
public	import	super	default	if	byte	long	finally	assert	strictfp
class	instanceof	this	do	return	char	null	throw	const	transient
enum	interface	break	else	switch	double	short	throws	final	Synchronized
volatile									

- # IDENTIFIERS

Definition: Definition: User-defined names for variables, functions, and classes.

Rules: The rules to write an identifiers changes from one programming language to another.

Similarities Between Java and Python

- Starting Character:

  Identifiers must begin with a letters/ character (A-Z, a-z) or underscore (_).

  Identifiers cannot start with a digit/ number (0-9).
- Subsequent Characters: From the second character in an identifier, letters, digits, or underscores can be used.
- Case Sensitivity: Yes, Identifiers are case-sensitive.
- Length: Both languages allow identifiers of arbitrary length
- Reserved Words: Identifiers cannot be keywords or reserved words in the language.
- Unicode Characters: Both Java and Python support Unicode characters in identifiers, meaning you can use letters from various languages (e.g., ñ, 你好, etc.).

IDENTIFIERS

Differences Between Java and Python

- Starting Characters:

  Java: Allows identifiers to start with letters (a-z, A-Z), underscores (_), or dollar signs ($).

  Python: Identifiers can only start with letters (a-z, A-Z) or underscores (_) and no Dollar sign ($) in Python identifiers.

- Use of the Dollar Sign ($):

  Java: The dollar sign ($) can be used in identifiers naming.

  Python: The dollar sign ($) can't be used in identifiers.

- Special Meaning of Underscore (_)

  Java: The underscore _ has no special meaning beyond being a valid character in identifiers.

  Python: The underscore _ has special meaning:

  Single leading underscore (_var): A convention for weak internal use (not enforced by the interpreter).

  Double leading underscore (__var): Used to trigger name mangling (used for private variables).

  Double leading and trailing underscores (init): Reserved for special methods (e.g., constructors, operator overloads).

- Static vs. Dynamic Typing:

  Java: A statically typed language, i.e., each identifier should be accompanied with the data type.

  Python: A dynamically typed language, so identifiers can be used with mentioning its data type.

- # IDENTIFIERS

TABLE 3. IDENTIFIERS SUMMARY TABLE (JAVA VS PYTHON)

Rule	Java	Python
Starting Character	Letters (a-z, A-Z), _, $	Letters (a-z, A-Z), _
Allowed Characters	Letters (a-z, A-Z), digits (0-9), _, $	Letters (a-z, A-Z), digits (0-9), _
Dollar Sign ($)	Allowed	Not allowed
Special Meaning of '_'	No special meaning	_var (weak internal use), __var (name mangling), __init__ (special methods)
Case Sensitivity	Case-sensitive	Case-sensitive
Length of Identifiers	No strict limit	No strict limit
Unicode Characters	Allowed	Allowed
Static vs Dynamic Typing	Statically typed (types are defined at compile-time)	Dynamically typed (types are inferred at runtime)
Reserved Keywords	Cannot use reserved words	Cannot use reserved words

- # FINITE AUTOMATA - REPRESENTATION

States:

o Start State (S0): Where the automaton begins.

o Valid Start Character (S1): For characters like a-z, A-Z, _ and $ (in Java).

o Valid Continuation Character (S2): For characters like a-z, A-Z, 0-9, _, $(in Java).

o End State (S3): A valid identifier that has reached its end.

Alphabet: Characters a-z, A-Z, 0-9, _, $(in Java).

Dead State: For both Java and Python, if one encounter a character that is not allowed in an identifier, you transition to a dead state where no further valid transitions exist.

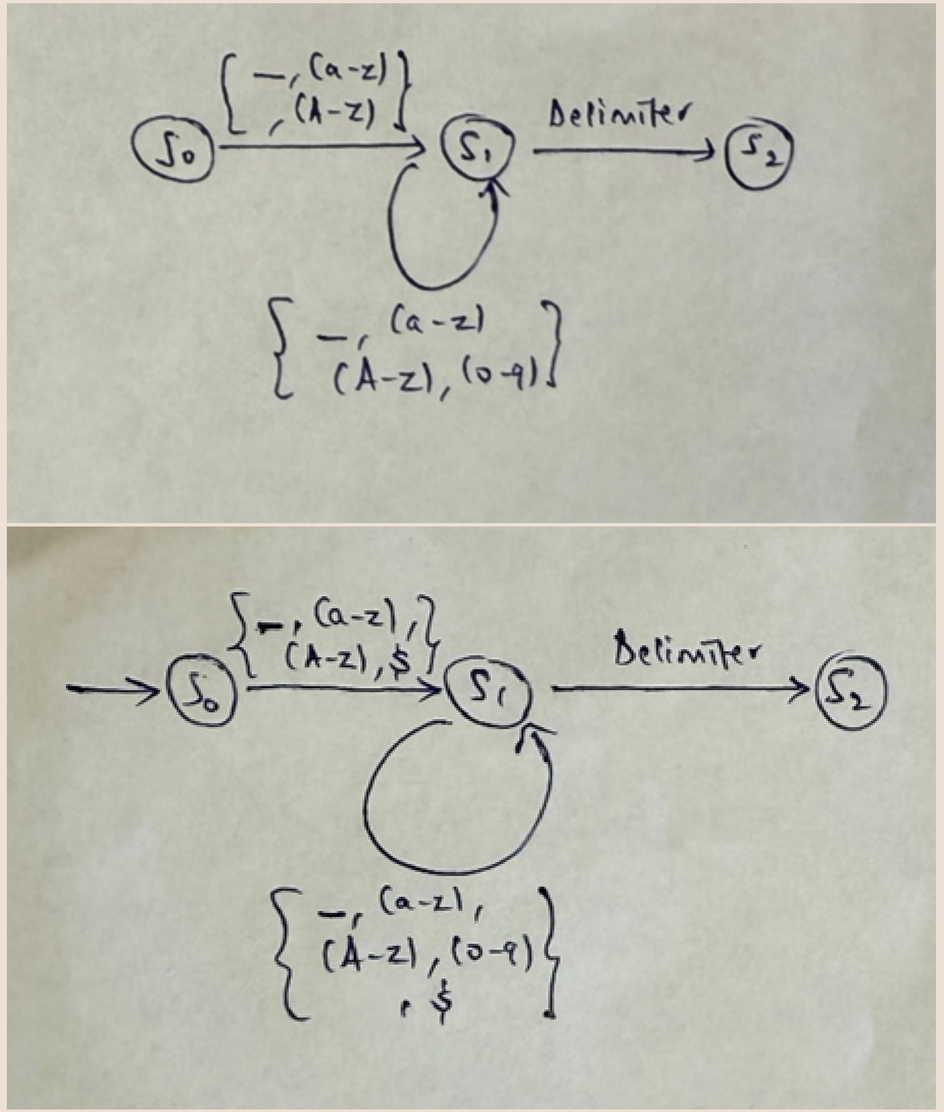


FIG 1. FINITE AUTOMATA FOR IDENTIFIERS IN PYTHON AND JAVA

- **Token Classification after Identification of Token Category**

o Keyword Matching:

After a token is scanned, it is first checked against the keyword table.

If the token matches any entry in the keyword table, it is classified as a keyword.

o Identifier Classification:

If the token does not match any entry in the keyword table, the scanner then verifies whether it satisfies the identifier rules.

If the token satisfies the identifier rules, it is classified as an identifier.

o Invalid Tokens:

If the token neither matches a keyword nor satisfies the identifier rules, it is classified as an invalid token (error) and handled accordingly.

- # PROGRAMMING EXAMPLE

PYTHON:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        # 'name' is an identifier, 'self' is a special identifier
        self.age = age
        # 'age' is an identifier
    def greet(self):
        if self.name:
            return f"Hello, {self.name}"
        else:
            return "Hello, World!"
# Creating an instance of the class
person = Person("Alice", 30)
print(person.greet())
```

Keywords:
class, def, if, else, return

Identifiers:
Person, __init__, self, name, age, greet, person

NOTE:

- **__init__** is a special identifier used for constructors.
- **print** is a function name defined in any of the python package.
- **f** in f"Hello, {self.name}" is a syntax feature for **f-strings** and doesn't belong to the category of keywords or identifiers. It's used to signify a formatted string literal in Python.

• PROGRAMMING EXAMPLE

JAVA:

```java
public class Person {
    private String name; // 'name' is an identifier
    private int age;     // 'age' is an identifier

    public Person(String name, int age) {
        // 'name' and 'age' are identifiers
        this.name = name;
        // 'this' is a special keyword
        this.age = age;
    }

    public String greet() {
        if (this.name != null) {
            // 'if' and 'return' are keywords
            return "Hello, " + this.name;
        } else {
            return "Hello, World!";
        }
    }

    public static void main(String[] args) {
        Person person = new Person("Alice", 30);
        // 'Person' and 'person' are identifiers
        System.out.println(person.greet());
    }
}
```

Keywords:
public, class, String, private, int, if, else, return, static, void, new

Identifiers:
Person, name, age, this, greet, main, args, person, System, out, println

NOTE:

- Here, **{"System", "out", "println"}** are the either name of the methods or the classes, Hence considered as identifiers.

• Java vs Python: Readability, Ease of Use, Verbosity, and Token Efficiency

TABLE 4. COMPARISON OF PYTHON AND JAVA: WRT TO TOKEN COUNT

Feature	Python	Java
Keywords	class, def, if, else, return	public, class, String, private, int, if, else, return, static, void, new
Identifiers	Person, __init__, self, name, age, greet, person	Person, name, age, this, greet, main, args, person, System, out, println
Token Count	Keywords: 5 Identifiers: 7	Keywords: 10 Identifiers: 10

To Summarize:

Java: Less readable, more verbose, harder to write.

Python: More readable, easy to write, concise, but still verbose.

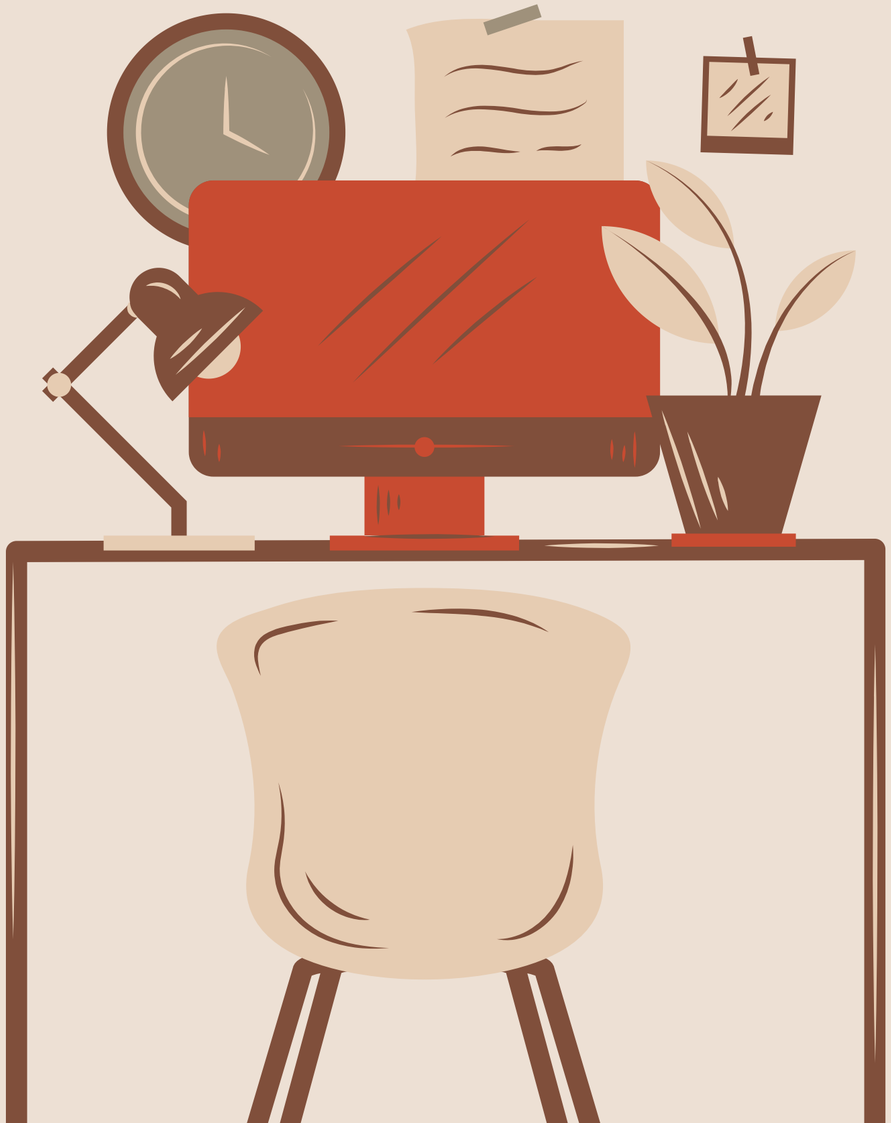
Token Efficiency Analysis:

- **PYTHON** has a total of **12 tokens**, which suggests that fewer elements (keywords and identifiers) are required to express the same functionality compared to Java. This results in more concise code, improving both readability and ease of use. The lower token count also typically translates to faster development and easier maintenance.
- **JAVA** has a total of **20 tokens**, which reflects its more verbose syntax. While this verbosity can provide benefits in terms of type safety and structure, it also increases cognitive load and development time. The higher token count generally results in more detailed and explicit code, but also reduces readability and increases complexity for smaller projects or quick tasks.

Part B: Literals & Operators

Part C: Separators/Delimiters, Comments, & Whitespace

III PART C: SEPARATORS/DELIMITERS, COMMENTS, & WHITESPACE



Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Thank you for listening!