**INTRODUCTION**

In this section, we will research and compare important token categories in programming languages. We will focus on Python and Java as our selected languages for comparison.

1. **KEYWORDS**

Definition: Keywords are reserved, predefined syntax(words) that holds a special meaning and are not used as identifiers in the computer programming.

1. **Python Keywords**

There are 35 keywords in Python that can’t be used for variable/ function names. They serve to specify the language's structure and grammar. Key Categories of Python Keywords:

* Boolean and Special Literals: False, None, True.
* Logical Operators: not, or, and.
* Control Flow: if, elif, else, while, for, continue, break, return, try, except, finally.
* Function/Class Definitions: def, class, lambda.
* Exception Handling: raise, assert.
* Importing and Scope Management: import, from, global, nonlocal.
* Asynchronous Programming: async, await.

*Table 1. Token Representation of Python Keywords*

|  |  |  |
| --- | --- | --- |
| **Category** | **Keyword Identified** | **Token Representation** |
| Asynchronous Programming | async | (keyword, "async") |
| await | (keyword, "await") |
| Boolean and Special Literals | False | (keyword, "False") |
| None | (keyword, "None") |
| True | (keyword, "True") |
| Context Management | with | (keyword, "with") |
| Control Flow | break | (keyword, "break") |
| continue | (keyword, "continue") |
| elif | (keyword, "elif") |
| else | (keyword, "else") |
| for | (keyword, "for") |
| if | (keyword, "if") |
| return | (keyword, "return") |
| try | (keyword, "try") |
| while | (keyword, "while") |
| Deletion | del | (keyword, "del") |
| Exception Handling | assert | (keyword, "assert") |
| except | (keyword, "except") |
| finally | (keyword, "finally") |
| raise | (keyword, "raise") |
| Function/Class Definitions | class | (keyword, "class") |
| def | (keyword, "def") |
| lambda | (keyword, "lambda") |
| Generator Function | yield | (keyword, "yield") |
| Importing and Scope Management | from | (keyword, "from") |
| global | (keyword, "global") |
| import | (keyword, "import") |
| nonlocal | (keyword, "nonlocal") |
| Keyword for Aliases/Contexts | as | (keyword, "as") |
| Logical Operators | and | (keyword, "and") |
| in | (keyword, "in") |
| is | (keyword, "is") |
| not | (keyword, "not") |
| or | (keyword, "or") |
| Null Operation | pass | (keyword, "pass") |

1. **Java Keywords**

Java has a set of **51 keywords** that also have reserved meanings. Key Categories of Java Keywords:

* Control Flow: if, else, while, for, switch, continue, break.
* Modifiers: private, protected, public, static, final, abstract.
* Data Types: int, char, boolean, double, float.
* Class and Object: class, interface, extends, implements.
* Exception Handling: try, catch, finally, throw, throws.
* Threading: synchronized, volatile.
* Access Modifiers: public, private, protected.

*Table 2. Token Representation of Java Keywords*

|  |  |  |
| --- | --- | --- |
| **Category** | **Keyword Identified** | **Token Representation** |
| Access Modifiers | private | (keyword, "private") |
| protected | (keyword, "protected") |
| public | (keyword, "public") |
| Class and Object | class | (keyword, "class") |
| enum | (keyword, "enum") |
| extends | (keyword, "extends") |
| implements | (keyword, "implements") |
| import | (keyword, "import") |
| instanceof | (keyword, "instanceof") |
| interface | (keyword, "interface") |
| new | (keyword, "new") |
| package | (keyword, "package") |
| super | (keyword, "super") |
| this | (keyword, "this") |
| Control Flow | break | (keyword, "break") |
| case | (keyword, "case") |
| continue | (keyword, "continue") |
| default | (keyword, "default") |
| do | (keyword, "do") |
| else | (keyword, "else") |
| for | (keyword, "for") |
| goto | (keyword, "goto") |
| if | (keyword, "if") |
| return | (keyword, "return") |
| switch | (keyword, "switch") |
| while | (keyword, "while") |
| Data Types | boolean | (keyword, "boolean") |
| byte | (keyword, "byte") |
| char | (keyword, "char") |
| double | (keyword, "double") |
| float | (keyword, "float") |
| int | (keyword, "int") |
| long | (keyword, "long") |
| null | (keyword, "null") |
| short | (keyword, "short") |
| void | (keyword, "void") |
| Exception Handling | catch | (keyword, "catch") |
| finally | (keyword, "finally") |
| throw | (keyword, "throw") |
| throws | (keyword, "throws") |
| try | (keyword, "try") |
| Modifiers | abstract | (keyword, "abstract") |
| assert | (keyword, "assert") |
| const | (keyword, "const") |
| final | (keyword, "final") |
| native | (keyword, "native") |
| static | (keyword, "static") |
| strictfp | (keyword, "strictfp") |
| transient | (keyword, "transient") |
| Threading | synchronized | (keyword, "synchronized") |
| volatile | (keyword, "volatile") |

1. **IDENTIFIERS**

**Definition:** Variables that are used to hold a value(s) in the programming language is referred to as Identifiers. These are symbols used to treat a particular value, and can be called/ used whenever and wherever it is needed, provided that the call should be within the scope of that variable.

1. **Similarities Between Java and Python**

Both Java and Python have several common rules for identifiers:

* 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 (though there may be practical limits).
* 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.).

1. **Differences Between Java and Python**

* Starting Characters:
* Java: Allows identifiers to start with letters (a-z, A-Z), underscores (\_), or dollar signs ($).
* Python: Identifiers can only start with letters (a-z, A-Z) or underscores (\_). Dollar sign ($) is not allowed 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 in Python (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.
* Type-less Variable Names:
* Java: Identifiers are linked to specific types and need to follow strict type conventions.
* Python: Identifiers are not type-bound, and the language allows a more flexible approach for variable names.

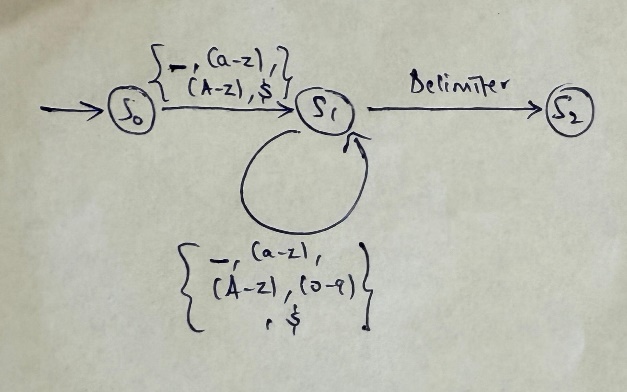
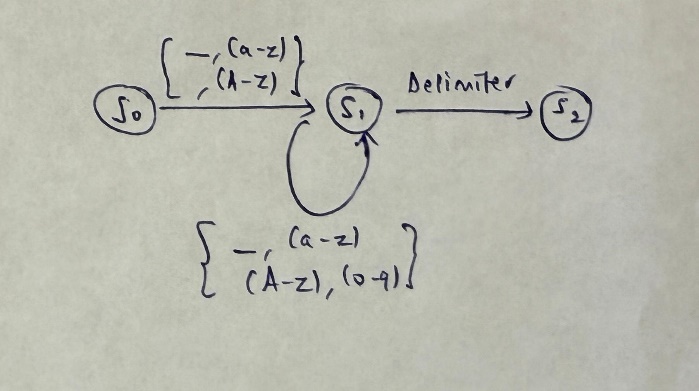
1. **Summary Table: Java Vs. Python Identifiers**

*Table 3. 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 |

1. **FINITE AUTOMATA REPRESENTATION**

**Definition:** A mathematical model of computation used to depict and examine the behavior of systems that can exist in a finite number of states is called a finite automaton (plural: automata). Based on the rules established by the automaton, it evaluates an input string of symbols (characters) and decides whether the string is approved or refused.



*Fig 1. Finite Automata for Identifiers in Python and Java (Scanner Action)*

**States:**

* Start State (S0): Where the automaton begins.
* Valid Start Character (S1): For characters like a-z, A-Z, \_ and $ (in Java).
* Valid Continuation Character (S2): For characters like a-z, A-Z, 0-9, \_, $(in Java).
* 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.

**Token Classification after Identification of Token Category:**

* Keyword Matching:
* After a token is scanned, it is first checked against the keyword table (a predefined list of reserved words for the language).
* If the token matches any entry in the keyword table, it is classified as a keyword.
* 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.
* 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.

1. **PROGRAM EXAMPLE**
2. **Python Code/ Program:**

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   
**Literals:** "Alice", 30, "Hello, {self.name}", "Hello, World!"  
**Operators:** =, ;, ,, .  
**Functions:** print()

**Note:** 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.

1. **Java Code/ Program:**

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

**Literals**: "Alice", 30, "Hello, ", "Hello, World!", null

**Operators**: =, +, !=, .

**Functions**: System.out.println()