Name/Identifier – string of characters to identify some entity in program

Naming Convention in Different Programming Languages:

Java and C# - No length limit with all character significant

C++ - Does not specify limit, implementors do

PHP – Variable begin with $ sign

Perl – beginning of variable $, @, or %

Ruby – beginning of variables name @ or @@ instance or class variable

Common Naming style in Python:

Function – lower case word separate in under score, snake case

Variable – lowercase single letter, word, words separate in underscore

Class – Start with capital, using PascalCase

Method – lowercase word, separate words under score

Constant – Uppercase, singleletter, word, or words separated in underscore.

Module – short, lowercase word or words separate in underscore.

Package – short, lowercase, word or words do not seperate.

Special Words/Reserve Keywords – Make programs more readable by naming action to be performed. Syntactic part of statements

Fortran – keywords can be redefined

Variables – abstraction of computer memory cell or collection of cell

* Variable Names – identify and reference value that can change, most common name
* Variable Address – unique machine memory address
* Variable Type – range of values variable can store
* Variable Value – contents of memory cells, r-value
* Variable Lifetime – amount of time variable exist
* Variable Scope – part of program where variable can be accessed

Bindings – association between attribute and entity

Binding Time – time binding takes place

Type Bindings – Variable must be bound to data type before being reference to program

Explicit Declaration – directly specify variable when declaring it. List variable name and specifies particular type.

Implicit Declaration – associating variables with type to default conventions

Both declarations create static bindings

Python – variables are dynamically type

Binding occurs in python when value is assigned to variable

Dynamic type binding – happens when variable type is not specified by a declaration statement.

Scope – range of statement wherein variable is visible.

Determines lifetime and visibility.

Types of scope:

Global scope – accessible from anywhere in the program.

Local scope – declared inside function or block of codes, access in specific function

Enclosing scope – encloses another function

Built-in Scope – contains function an variables built into python

Scope Rules:

LEGB Rule: follows legb when looking up variable

* Local
* Enclosing
* Global
* Built in

Shadowing: variable declared in local has same name in outer, it shadows outer variable

Global Keyword: to modify global variable inside function it must have global keyword

Lexical Analysis – read source code left to right and organize into tokens.

Tokens – meaningful element for computer to understand

Lexical Analyzer – collects character into logical groupings, assign internal codes based on structure.

Lexems – logical groupings

Regular Expression – programming language tokens

Lexical analyzer use DFA to recognize tokens.

Process of Lexical Analyzer:

Input preprocessing – cleaning up input text and preparing for lexical analysis

Tokenization – breaking input text into a sequence of token

Token Classification – determine type of each token

Token Validation – check if token valid based on rule of programming

Output Generation – generate output sequence of toke or token stream

Tokens – individual words or symbols in a sentence

Classification of Tokens:

* Alphabet – All number and alphabets are considered hexadecimal alphabets
* Strings – collection of different alphabets occurring continuously
* Symbols –
* Non Tokens: Comments, blanks, tabs

Lexemes – Sequence of Characters matched by a pattern to form the token

Syntax Analysis or parsing – process of analyzing string of symbol according to the rules of formal grammar

Syntax Errors – identified and flagged in this phase must be corrected before program is success

Syntax Analyzer or parser- takes token stream analyze against production rules to detect errors

Parse tree or Abstract Syntax Tree (AST) – output of syntax analyzer

Use regex and pattern rules to identify tokens

Context-free grammar (CFG) – define syntax rules of programming language, syntax language

Parser Accomplish:

* Parsing – tokens analyzed based on grammar rules
* Error Handling – if programs contain syntax error
* Symbol Table Creation – table or data structure that store information about identifiers user in the program

Derivation – applying CFG to generate a sequence of tokens to form valid structure

2 decision

* Deciding non terminal to be replaced
* Deciding which production rules to use for replace

**Types of Derivation:**

1. **Left-Most Derivation (LMD)**
   * Always replaces the **left-most non-terminal** first.
   * The resulting sequence of steps is called a **left-sentential form**.
2. **Right-Most Derivation (RMD)**
   * Always replaces the **right-most non-terminal** first.
   * The resulting sequence of steps is called a **right-sentential form**.

Parse Tree – Graphical Representation of derivation.