**Python Identifiers**

A name in a Python program is called an identifier. It is used to identify variables, functions, classes, and other objects.

**Rules for Identifiers:**

1. **Alphabet Symbols**: Identifiers can be formed using uppercase and lowercase alphabets (e.g., myVariable, Var123).
2. **Underscore Prefix**: If an identifier starts with an underscore (\_), it signifies that the identifier is private or internal (e.g., \_privateVar).
3. **No Digits at the Start**: Identifiers cannot start with digits (e.g., 123name is invalid).
4. **Case Sensitivity**: Identifiers are case-sensitive. For example, Name and name are treated as different identifiers.
5. **Reserved Words**: Reserved keywords cannot be used as identifiers.
   * Example: def = 10 is invalid because def is a reserved keyword.
6. **No Length Limit**: There is no fixed length limit for identifiers. However, long names are discouraged for readability and best practices.
7. **Dollar Symbol**: The dollar ($) symbol is not allowed in Python identifiers.

**Python Reserved Words**

There are 33 reserved words in Python that cannot be used as identifiers.

* **Boolean & None**: True, False, None
* **Logical Operators**: and, or, not, is
* **Conditional Statements**: if, elif, else
* **Looping**: while, for, break, continue, return, in, yield
* **Exception Handling**: try, except, finally, raise, assert
* **Importing & Defining**: import, from, as, class, def, pass, global, nonlocal, lambda, del, with

**Python Data Types**

Python supports several built-in data types that can be used to represent various kinds of data.

1. **Int**: Integer values (e.g., x = 5).
2. **Float**: Floating-point values (e.g., x = 5.75).
3. **Complex**: Complex numbers (e.g., x = 3 + 5j).
4. **Bool**: Boolean values (True or False).
5. **Str**: String values (e.g., x = "Hello").
6. **Bytes**: Immutable sequences of bytes, similar to arrays.
   * Allowed values range from 0 to 256. If any value outside this range is provided, a ValueError is raised.
   * Once created, byte values cannot be changed; attempting to do so raises a TypeError.
7. **Bytearray**: Mutable version of bytes. The values of a bytearray can be modified.
8. **Range**: Represents an immutable sequence of numbers.
   * Example: range(10) generates numbers from 0 to 9.
9. **List**: Represents a collection of values where insertion order is preserved and duplicates are allowed.
   * Mutable and growable in nature.
   * Example: my\_list = [1, 2, 3, 4]
10. **Tuple**: Similar to a list but immutable. Once created, the values cannot be modified.
    * Example: my\_tuple = (1, 2, 3)
11. **Set**: Represents a collection of unique values where order is not important.
    * Duplicates are not allowed, and it is mutable.
12. **Frozenset**: Immutable version of a set. The elements cannot be changed.
13. **Dict**: Represents key-value pairs. Duplicate keys are not allowed, but values can be duplicated.
    * Example: my\_dict = {101: 'durga', 102: 'ravi'}
14. **None**: Represents the absence of a value or null value. It is used when no value is associated with a variable.

def m1(): a = 10 print(m1())

Slice Operator

my\_string = "Hello, World!" reversed\_string = my\_string[::-1] print(reversed\_string)

Typecasting: Typecasting refers to the process of converting a value from one data type to another.