ASSIGNMENT - 1

MCQS

1. Which of the following is NOT a unary operator?

a) +

b) -

c) *

d) ~

2. What do identity operators compare?

a) If two objects have the same value.

b) If two objects have the same memory location.

c) The data type of a variable.

d) Both b and c.

5 - MARKS

1. Describe the process of "coercing an object to a new data type" in Python. What is the syntax for this operation, and what are some limitations or considerations when performing data type coercions?

- **Answer:** Coercing an object to a new data type in Python means converting the data type of an object to another. The syntax is `datatype(object)`. The changes can be stored in the same variable or a different variable

- **Limitations/Considerations:**
- Only a few coercions are accepted.
- If the value enclosed within quotes is a string that cannot be semantically converted to the target data type (e.g., converting "Ram" to a float), a `ValueError` will be raised. For example, `float("Ram")` will result in an error. However, a string containing an integer (e.g., `'1'`) can be coerced to an integer using `int('1')`.

2**.** Discuss the concept of the precedence of operators.

Precedence of Operators: The "precedence of operators" refers to the order in which different operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. This hierarchy is crucial for correctly interpreting and evaluating complex mathematical or logical expressions.

- Understanding operator precedence ensures that expressions are evaluated in the intended order, leading to the correct result.
- For the expression: `A = 7 - 2 * (27 / 3**2) + 4`

  And its evaluation is based on precedence:
  1. **Parentheses:** First, `(27 / 3**2)` is evaluated.
  2. **Exponent:** Inside the parentheses, `3**2` is evaluated first, which is `9`.
  3. **Division:** Then, `27 / 9` is evaluated, which is `3`.
  4. The expression becomes: `A = 7 - 2 * 3 + 4`
  5. **Multiplication:** Next, `2 * 3` is evaluated, which is `6`.
  6. The expression becomes: `A = 7 - 6 + 4`
  7. **Addition and Subtraction (left to right):**
     - `7 - 6` is `1`.
     - `1 + 4` is `5`.

     The final result is `5.0`. This step-by-step evaluation clearly demonstrates how the hierarchy (Parentheses > Exponent > Division/Multiplication > Addition/Subtraction) dictates the calculation order.

7- MARKS

1. Analyze the various categories of operators, detailing their purpose and providing at least one example for each type.

   Answer:

- **Arithmetic Operators:**
  - **Purpose:** Used to perform mathematical operations between two operands.
  - **Example:**
    - `+` (Addition) `-` (Subtraction) `*` (Multiplication) `/` (Division) `%` (Remainder) `**` (Exponent)
- **Assignment Operators:**
  - **Purpose:** Used to assign values to variables. They combine an arithmetic operation with an assignment.

    - **Example:**
        - = (Assign values) += (Add and assign) -= (Subtract and assign)
        - *= (Multiply and assign) /= (Divide and assign).
- **Relational or Comparison Operators:**
    - **Purpose:** Test numerical equalities and inequalities between two operands and return a boolean value (`True` or `False`). All relational operators have the same precedence.
    - **Example :**
        - < (Strictly less than): `x<y` is `True`.
        - <= (Less than equal to): `x<=y` is `True`.
        - > (Strictly greater than): `x>y` is `False`.
        - >= (Greater than equal to): `x>=y` is `False`.
        - == (Equal to equal to): `x==y` is `False`.
        - != (Not equal to): `x!=y` is `True`.
- **Logical Operators:**
    - **Purpose:** Used when operands are conditional statements and return a boolean value. In Python, they work with scalars or boolean values.
    - **Example (with x=4, y=7):**
        - `or` (Logical OR): `(x>y) or (x<y)` is `True` (since `x<y` is `True`).
        - `and` (Logical AND): `(x>y) and (x<y)` is `False` (since `x>y` is `False`).
        - `not` (Logical NOT): `not (x==y)` is `True` (since `x==y` is `False`).
- **Bitwise Operators:**
    - **Purpose:** Used when operands are integers, treating them as strings of binary digits and operating bit by bit. They can also operate on conditional statements.
    - **Example (with x=5, y=7):**
        - | (Bitwise OR on integers): `x | y` (5 binary `0101` OR 7 binary `0111`) results in 7 (binary `0111`).
        - & (Bitwise AND on integers): copies a bit to the result if it exists in both operands.
        - | (Bitwise OR on conditional statements): `(x<y) | (x==y)` is `True` (since `x<y` is `True`).
        - & (Bitwise AND on conditional statements): `(x<y) & (x==y)` is `False` (since `x==y` is `False`).

2. Explain the concept of "indexing" in sequence data types in Python. Discuss how indexing is applied to Strings, Lists, Arrays, and Tuple objects, providing an example for each where applicable.

Answer: Indexing in Python refers to the process of accessing individual elements within an ordered sequence data type (like strings, lists, or tuples) by their position. The index starts from 0 for the first element, and negative indexing can be used to access elements from the end of the sequence, where -1 refers to the last element.

1. **Strings:** Strings are sequences of characters and support indexing.

Example:
Python
```
strSample = 'learning'

print(strSample.index('l'))  # Output: 0 (finds the index of the first occurrence of 'l')

print(strSample)          # Output: 'g' (accesses the character at index 7)
```

2. **Lists:** Lists are sequences of multiple data type objects and fully support indexing.

Example:
Python
```
lstSample = [1, 2, 'a', 'sam', 2]

print(lstSample.index('sam')) # Output: 3 (finds the index of 'sam')

print(lstSample)          # Output: 'a' (accesses the element at index 2)
```

3. **Arrays:** Arrays are sequences of a constrained list of objects (all objects of the same datatype) and support indexing.

   Example:
   Python
   ```
   from array import array

   arrSample = array('i')
   ```

   - print(arrSample[-3])   # Output: 2 (accesses the third to last element)
4. **Tuples:** Tuples are sequences of compound data and fully support indexing.

Example:
Python
```
tupSample = (1, 2, 3, 4, 3, 'py')

print(tupSample.index('py'))  # Output: 5 (finds the index of 'py')
```