Python

Week 2 Assignment Solutions

PW SKILLS

1. **What are the Operators? And explain all the 7 Operators in detail with examples.**

Solution:

In Python, operators are special symbols or keywords that perform operations on variables and values. There are several types of operators in Python, including arithmetic operators, comparison operators, assignment operators, logical operators, bitwise operators, membership operators, and identity operators.

Here's an overview of each type of operator along with examples:

a. Arithmetic Operators:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, division, etc.

- `+` (Addition): Adds two operands.
- `-` (Subtraction): Subtracts the second operand from the first.
- `*` (Multiplication): Multiplies two operands.
- `/` (Division): Divides the first operand by the second (always results in a float).
- `%` (Modulus): Returns the remainder of the division of the first operand by the second.
- `//` (Floor Division): Returns the quotient of the division, removing the decimal part.
- `**` (Exponentiation): Raises the first operand to the power of the second operand.

Example:

```
a = 10
b = 3
print(a + b)   # Output: 13
print(a - b)   # Output: 7
print(a * b)   # Output: 30
print(a / b)   # Output: 3.3333333333333335
print(a % b)   # Output: 1
print(a // b)  # Output: 3
print(a**b)  # Output: 1000
```

b. Comparison Operators:

Comparison operators are used to compare values. They return True or False based on the comparison result.

- `==` (Equal): Returns True if both operands are equal.
- `!=` (Not Equal): Returns True if operands are not equal.
- `>` (Greater Than): Returns True if the left operand is greater than the right.
- `<` (Less Than): Returns True if the left operand is less than the right.
- `>=` (Greater Than or Equal To): Returns True if the left operand is greater than or equal to the right.
- `<=` (Less Than or Equal To): Returns True if the left operand is less than or equal to the right.

Example:

```
 x = 10
y = 5
print(x == y)   # Output: False
print(x != y)   # Output: True
print(x > y)    # Output: True
print(x < y)    # Output: False
print(x >= y)   # Output: True
print(x <= y)   # Output: False
```

c. Assignment Operators:

Assignment operators are used to assign values to variables.

- `=` (Assignment): Assigns the value on the right to the variable on the left.
- `+=`, `-=`, `*=`, `/=`, `%=` etc. (Compound Assignment): Performs arithmetic operation and assigns the result to the variable.

Example:

```
a = 10
b = 5
a += b   # Equivalent to: a = a + b
print(a)   # Output: 15
```

d. Logical Operators:

Logical operators are used to combine conditional statements.

- `and`: Returns True if both statements are true.
- `or`: Returns True if one of the statements is true.
- `not`: Returns True if the statement is false (negates the statement).

Example:

```
x = 5
print(x > 0 and x < 10)   # Output: True
print(x > 0 or x < 2)     # Output: True
print(not(x > 0 and x < 10))  # Output: False
```

e. Bitwise Operators:

Bitwise operators perform operations on binary representations of integers.

- `&` (Bitwise AND)
- `|` (Bitwise OR)
- `^` (Bitwise XOR)
- `~` (Bitwise NOT)
- `<<` (Left Shift)
- `>>` (Right Shift)

Example:

- a = 2 # binary representation: 0010
- b = 3 # binary representation: 0011
- print(a & b) # Output: 2 (binary: 0010)
- print(a | b) # Output: 3 (binary: 0011)

f. Membership Operators:

Membership operators are used to test if a sequence is presented in an object.

- `in`: Returns True if a sequence with the specified value is present in the object.
- `not in`: Returns True if a sequence with the specified value is not present in the object.

Example:

```
my_list = [1, 2, 3, 4, 5]
print(3 in my_list)    # Output: True
print(6 not in my_list) # Output: True
```

g. Identity Operators:

Identity operators are used to compare the memory locations of two objects.

- `is`: Returns True if both variables are the same object.
- `is not`: Returns True if both variables are not the same object.

Example:

```
x = [1, 2, 3]
y = [1, 2, 3]
z = x
print(x is z)   # Output: True
print(x is y)   # Output: False
print(x is not y)  # Output: True
```

These are the different types of operators in Python along with examples illustrating their usage and functionality.

2. Which of the following is Incorrect?

a)print(20//3)=6
b)print (-15//2)=-7
c)print (15.0//2)=7.0
d)print (-15.0//2)=-8.0

Solution:

Option b is the incorrect choice.
Explanation:b) print (-15//2)=-8: This is a floor division operation that divides -15 by 2. The result is -7.5, but since we are using floor division, the result is rounded down to the nearest integer, which is -8.

3. What will be the output of the following program?Which of the following is Incorrect?

```
a = float("-inf")
b = float("inf")
print(a > b)
print(a < b)
print(a == b)
print(a != b)
print(a >= b)
print(a <= b)
```

Output:

False
True
False
True
False
True

4. what will be the Output of the following code? Explain it.

a)a=144

print(a>>1,a>>2,a>>3,a>>4,a>>5)

Output: 72 36 18 9 4

b)a=9

print(a<<1,a<<2,a<<3,a<<4,a<<5)

Output: (18, 36, 72, 144, 288)

c)a,b=14,6

(a>>1)+3*(b)-36/12

Solution: 22.0

d)a,b=14,6

```
a,b=b,a
temp2=(a>>1)+3*(b)-b<<2
print(temp2)
```

Solution: 124

5. Number Conversion.

a) Convert 27 in Binary Number.

Solution:

Steps to be followed for converting Decimal Number to Binary Number.

1.Divide the decimal number by 2.
2. Record the remainder (either 0 or 1).
3. Repeat steps 1 and 2 with the quotient obtained from the previous division until the quotient becomes 0.
4. The binary representation will be the sequence of remainders obtained in reverse order.

Step 1: 27 ÷ 2 = 13, remainder 1
Step 2: 13 ÷ 2 = 6, remainder 1
Step 3: 6 ÷ 2 = 3, remainder 0
Step 4: 3 ÷ 2 = 1, remainder 1
Step 5: 1 ÷ 2 = 0, remainder 1

Now, reading the remainders from bottom to top: 11011

b) Convert 56.30) in Binary Number.

Solution:

So, Here it is divided into two parts :56+0.30
So, the Binary conversion of 56 is :111000
And Binary Conversion of 0.30 is calculated as 👍

0.30*2=0.6
0.6*2=1.20
0.20*2=0.4
0.4*2=0.8
0.8*2=1.6
0.6 - - - It repeating so, there is no final answer for this:
Hence 0.30= 01001

So, the final answer is :111000.01001

c) Convert 1010101) into decimal Number.

Solution:

To convert the binary number 1010101 to decimal, you can use the positional notation method. This method involves multiplying each digit of the binary number by increasing powers of 2, starting from the rightmost digit. Then, sum up all the products to get the decimal equivalent.

```
Binary: 1010101

Position:    6   5   4   3   2   1   0
Binary:      1   0   1   0   1   0   1

Calculation: (1 * 2^6) + (0 * 2^5) + (1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 *
2^0)

Result:       (1 * 64) + (0 * 32) + (1 * 16) + (0 * 8) + (1 * 4) + (0 * 2) + (1 * 1)
            = 64 + 0 + 16 + 0 + 4 + 0 + 1
            = 85
```

d) Convert 111111) into Decimal Number.

Solution: 63

Programming Assignments:

Note: for Programming Assignments Please refer to the colab Notebook.
Programming-Solution