DS&AI

Python for Data Science

DPP: 1

Basics of Python

Q1 The output of below python code segment is _____

```
a = 5.5
b = 2.0
c = 2
d = –2.0
e = a // b
f = d // c
print (e + f)
```

(A) 0 (B) 0.0
(C) 1.0 (D) –1.0

Q2 The ascending order Of Precedence of below Operators is _____

1. not in
2. <<
3. not
4. ^

(A) 3, 2, 1, 4 (B) 3, 1, 2, 4
(C) 3, 1, 4, 2 (D) 2, 4, 1, 3

Q3 The result after evaluating the below expression in Python is _____

result = 14 & 4 + 5 << 2 ^ 19 + 3 // 7 – 3 >> 4

Q4 Match The Following Operators with their associativity.

LIST-I	LIST-II
(A) ** (Exponentiation)	1. Left To Right
(B) & (Bitwise AND)	2. Right To Left
(C) is not (Identity)	
(D) = (Assignment)	

(A) A-1, B-2, C-2, D-1
(B) A-2, B-2, C-1, D-1
(C) A-2, B-1, C-2, D-2
(D) A-2, B-1, C-1, D-2

Q5 The output of below code segment is _____

```
a = 0° 63
b = a<<2
c = a>>3
print(b+c)
```

Q6 What will be printed by below Python Code?

```
i = 0 x A E 1
j = i & 152
k = j | 100
print (k)
```

(A) 344 (B) 00344
(C) 0xe4 (D) 228

Q7 What will be the value of result in the below code?

```
x = 0b01010110
y = 0x123
z = 42
result=x+y-z
```

(A) 517 in octal
(B) 517 in decimal
(C) 1f4 in hexa decimal
(D) 335 in in decimal

Q8 What will be the value of k in the below code?

```
i = –13.5
j = 5
k = i % j
print (k)
```

(A) 0.0 (B) 1.5
(C) 3.5 (D) –1.5

Answer Key

Q1	(C)
Q2	(C)
Q3	5
Q4	(D)
Q5	210
Q6	(D)
Q7	(A, D)
Q8	(B)

Hints & Solutions

Q1 Text Solution:

- **Compute `e`**
  - `a = 5.5`
  - `b = 2.0`
  - We need to compute `e = a // b`, where `//` is the floor division operator.
    In Python, floor division of floating-point numbers results in a floating-point number that is floored.
    - `5.5 // 2.0` evaluates to `2.0`, because `5.5 / 2.0 = 2.75`, and the floor of `2.75` is `2.0`.
- **Compute `f`**
  - `d = -2.0`
  - `c = 2`
  - We need to compute `f = d // c`.
    - `-2.0 // 2` evaluates to `-2.0`, because `-2.0 / 2 = -1.0`, and the floor of `-1.0` is `-1.0`. However, since the floor division result for negative numbers using float in Python is floored to the next smallest integer value, the result is `-2.0`.
- **Calculate `e + f`**
  - `e = 2.0`
  - `f = -2.0`
  - Adding these values together: `e + f = 2.0 + (-2.0) = 0.0`

Q2 Text Solution:

To determine the ascending order of precedence of the given operators, let's first review the precedence levels of each operator in Python:

1. **`not`**: Logical negation operator
2. **`in`**: Membership operator
3. **`<<`**: Bitwise left shift operator
4. **`^`**: Bitwise XOR operator

Precedence Levels

1. **`not`**: Logical NOT has a higher precedence than most operators but lower than comparison operators and some arithmetic operators.
2. **`in`**: Membership operators have precedence that is similar to comparison operators but lower than bitwise shift operators.
3. **`<<`**: Bitwise shift operators (<<, >>) have higher precedence than bitwise XOR (^) and logical operators.
4. **`^`**: Bitwise XOR has lower precedence than bitwise shift operators but higher than logical operators like `not`.

Summary of Precedence

- **`not`** has the highest precedence among the given operators.
- **`in`** has lower precedence than bitwise shift operators.
- **`<<`** (bitwise left shift) has higher precedence than bitwise XOR.
- **`^`** (bitwise XOR) has the lowest precedence among the given operators.

Ascending Order of Precedence

1. **`^`** (lowest precedence)
2. **`<<`**
3. **`in`**
4. **`not`** (highest precedence)

Therefore, the ascending order of precedence is:
`^ < << < in < not`
Correct Answer
(C)

Q3 Text Solution:

1. **Handle bitwise XOR and AND:**
   - **Bitwise AND (`&`)**: `14 & 20` evaluates to `4`. (In binary, `14` is `1110` and `20` is `10100`. The bitwise AND results in `100` which is `4`).

- **Bitwise XOR (^)**: `4 ^ 19` evaluates to 15. (In binary, 4 is `100` and 19 is `10011`. The bitwise XOR results in `1111` which is 15).

Conclusion

The result of the expression `14 & 4 + 5 << 2 ^ 19 + 3 // 7 - 3 >> 4` is 5.

Q4 Text Solution:

Operators and Their Associativity

1. **Exponentiation (`**`)**
   - **Associativity**: Right to Left
   - **Explanation**: Exponentiation in Python is evaluated from right to left, meaning `2 ** 3 ** 2` is evaluated as `2 ** (3 ** 2)`.
2. **Bitwise AND (`&`)**
   - **Associativity**: Left to Right
   - **Explanation**: Bitwise AND is evaluated from left to right, meaning `a & b & c` is evaluated as `(a & b) & c`.
3. **Identity (`is not`)**
   - **Associativity**: Right to Left
   - **Explanation**: Identity operators like `is not` are evaluated from left to right, but they are usually considered in terms of comparisons. For practical purposes in Python, comparisons like `is` and `is not` are evaluated left to right.
4. **Assignment (`=`)**
   - **Associativity**: Right to Left
   - **Explanation**: Assignment operators are evaluated from right to left, meaning `a = b = c` is evaluated as `a = (b = c)`.

Matching Operators with Their Associativity

1. **`**` (Exponentiation)**: Right to Left
2. **`&` (Bitwise AND)**: Left to Right
3. **`is not` (Identity)**: Left to Right
4. **`=` (Assignment)**: Right to Left

Thus, the correct match from LIST-I to LIST-II is:

- **A (Exponentiation)**: 2 (Right to Left)
- **B (Bitwise AND)**: 1 (Left to Right)
- **C (Identity)**: 1 (Left to Right)
- **D (Assignment)**: 2 (Right to Left)

Q5 Text Solution:

- **Left Shift (`a << 2`)**:
  - `63 << 2` means shifting the bits of 63 (which is `00111111` in binary) left by 2 positions.
  - Result: `11111100` in binary, which is 252 in decimal.
- **Right Shift (`a >> 3`)**:
  - `63 >> 3` means shifting the bits of 63 (which is `00111111` in binary) right by 3 positions.
  - Result: `00000111` in binary, which is 7 in decimal.

Q6 Text Solution:

1. **Hexadecimal to Decimal Conversion**
   - `i = 0xAE1` represents a hexadecimal number. Convert it to decimal:
     - `0xAE1` in hexadecimal is equal to 2785 in decimal.
2. **Bitwise AND Operation**
   - `j = i & 152`
   - First, convert 152 to binary:
     - 152 in binary is `10011000`.
   - Perform the bitwise AND operation between 2785 (in binary: `10101110 0001`) and 152 (in binary: `00000000 10011000`):
     - In binary: `10101110 0001 & 00000000 10011000` = `00000000 00001000` (binary) = 8 in decimal.
3. **Bitwise OR Operation**
   - `k = j | 100`
   - Convert `100` to binary:

- `100` in binary is `01100100`.
- Perform the bitwise OR operation between `8` (in binary: `00001000`) and `100` (in binary: `01100100`):
  - In binary: `00001000 | 01100100` = `01101100` (binary) = `108` in decimal.

4. **Print the Result**
   - The result of `k` is `228` in decimal.

Conclusion
The output of the code segment is `228`

Q7 **Text Solution:**
a, d,

- **Octal Representation:**
  - Decimal `335` in octal is `517`.
- **Hexadecimal Representation:**
  - Decimal `335` in hexadecimal is `0x14F`.

Q8 **Text Solution:**

- **Calculate the Division**
  - Divide `i` by `j`: `-13.5 / 5 = -2.7`.
- **Calculate the Floor of the Division**
  - The floor of `-2.7` is `-3`. This is the largest integer less than or equal to `-2.7`.
- **Calculate the Product of the Floor Value and the Divisor**
  - Multiply the floor value by `j`: `-3 * 5 = -15`.
- **Calculate the Modulus**
  - Subtract this product from `i`: `-13.5 - (-15) = -13.5 + 15 = 1.5`.

  So, the modulus operation `-13.5 % 5` yields `1.5`.