Editorial-Solution-W1A2: Python Fundamentals and Cryptography

---

Question 1

You are creating a program to categorize numbers.
The rules are:

If the number is divisible by both 3 and 5, print "FizzBuzz".

else If the number is divisible by 3, print "Fizz".

else If the number is divisible by 5, print "Buzz".

Otherwise, print the number itself.

Which of the following code snippets correctly implements this logic?

Options:

1.

```
if num % 3 == 0 and num % 5 == 0:

    print("FizzBuzz")

elif num % 3 == 0:

    print("Fizz")

elif num % 5 == 0:

    print("Buzz")

else:

    print(num)
```

2.

```
if num % 3 == 0:

  print("Fizz")
```

```
elif num % 5 == 0:

  print("Buzz")

elif num % 3 == 0 and num % 5 == 0:

  print("FizzBuzz")

else:

    print(num)
```

3.

```
if num % 3 == 0 or num % 5 == 0:

    print("FizzBuzz")

elif num % 3 == 0:

    print("Fizz")

elif num % 5 == 0:

    print("Buzz")

else:

    print(num)
```

4.

```
if num % 3 == 0 and num % 5 == 0:
```

```
        print("FizzBuzz")

    if num % 3 == 0:

        print("Fizz")

    if num % 5 == 0:

        print("Buzz")

    if num % 3 != 0 and num % 5 != 0:

        print(num)
```

Answer and Explanation:

- Correct option no. : **1**
- Run these in colab and try out with different numbers.

Question 2 (MSQ)

You want to create a list of even numbers from 2 to 20 (inclusive). Which of the following code snippets correctly creates this list?

Options:

1.

```
even_numbers = [x for x in range(2, 21) if x % 2 == 0]
```

2.

```
even_numbers = list(range(2, 21, 2))
```

3.

```
even_numbers = []

for i in range(2, 21):
```

```
if i % 2 == 0:


    even_numbers.append(i)
```

4.

```
even_numbers = [2 * i for i in range(1, 11)]
```

Answer and Explanation:

- Correct option no. : **1, 2, 3, 4**
- All four options correctly create a list of even numbers from 2 to 20:
    - Option 1 uses list comprehension with a filter
    - Option 2 uses range with a step of 2
    - Option 3 uses traditional for loop with conditional append
    - Option 4 uses list comprehension with multiplication

Question 3 (MSQ)

Which of the following statements about integer datatypes in Python are correct?

Options:

1. Integers in Python also, includes decimal numbers
2. Python supports both positive and negative integers
3. int() function can be used to convert a float or string to an integer
4. int() function always converts any datatype variable present inside it into a integer datatype, never giving an error.

Answer and Explanation:

- Correct option no. : **2, 3**
- Explanation:
    - Option 1 is incorrect
    - Option 2 is correct: Python supports arbitrary-size integers, both positive and negative
    - Option 3 is correct: int() can convert compatible floats and strings to integers
    - Option 4 is incorrect: int() does not always succeed—it raises errors on invalid input (e.g., int("abc")).

Question 4 (MSQ)

You want to get user input for their age and store it as an integer. Which of the following code snippets correctly accomplishes this?

Options:

1.

```
age = input("Enter your age: ")
```

2.

```
age = int(input("Enter your age: "))
```

3.

```
age = input("Enter your age: ")


age = int(age)
```

4.

```
age = int("Enter your age: ")
```

Answer and Explanation:

- Correct option no. : **2, 3**
- Explanation:
    - Option 1 is incorrect: Stores the age as a string, not an integer
    - Option 2 is correct: Directly converts the input to an integer
    - Option 3 is correct: Converts the input to an integer in two steps
    - Option 4 is incorrect: Causes a TypeError because it tries to convert a non-numeric string.

Question 5 (MSQ)

Which of the following statements about typecasting in Python are correct?

Options:

1. Typecasting can be used to convert between different data types
2. The str() function can be used to convert a number to a string
3. Typecasting from float to int always rounds up to the nearest integer
4. The float() function can be used to convert an integer or string to a float

Answer and Explanation:

- Correct option no. : **1, 2, 4**
- Explanation:
    - Option 1 is correct: Python supports conversion between various data types
    - Option 2 is correct: str() can convert numbers and other types to strings
    - Option 3 is incorrect: float to int conversion truncates instead of rounding up.
    - Option 4 is correct: float() can convert integers and valid strings to float values

---

Question 6 (MSQ)

You are creating a program to determine the season based on the month number. Which of the following code snippets correctly implements the seasonal logic rules?

The rules are:

- Months 3-5 (inclusive): Spring
- Months 6-8 (inclusive): Summer
- Months 9-11 (inclusive): Autumn
- Months 12, 1, 2: Winter

(Assume that month variable can only take integer values from 1 to 12(inclusive) wherever, month variable is not defined prior.)

Options:

1.

```
if 3 <= month <= 5:

    season = "Spring"

elif 6 <= month <= 8:

    season = "Summer"

elif 9 <= month <= 11:
```

```
    season = "Autumn"

else:

    season = "Winter"
```

2.

```
if month in [3, 4, 5]:

    season = "Spring"

elif month in [6, 7, 8]:

    season = "Summer"

elif month in [9, 10, 11]:

    season = "Autumn"

elif month in [12, 1, 2]:

    season = "Winter"
```

3.

```
season = "Winter" if month in [12, 1, 2] else \

        "Spring" if 3 <= month <= 5 else \

        "Summer" if 6 <= month <= 8 else \

        "Autumn"
```

4.

```
season = ["Winter", "Winter", "Spring", "Spring", "Spring",

          "Summer", "Summer", "Summer", "Autumn", "Autumn", "Autumn",
"Winter"][month - 1]
```

Answer and Explanation:

- Correct option no. : **1, 2, 3, 4**
- All options correctly implement the seasonal logic:
    - Option 1 uses range comparison with conditional statements
    - Option 2 uses list membership testing
    - Option 3 uses nested ternary operators
    - Option 4 uses list indexing with month number

Question 7 (MSQ)

You want to create a list of the first 10 square numbers (1, 4, 9, 16, ..., 100). Which of the following code snippets correctly creates this list?

Options:

1.

```
squares = [x**2 for x in range(1, 11)]
```

2.

```
squares = [(x-1)*(x-1) for x in range(2, 12)]
```

3.

```
squares = []


for i in range(1, 11):


    squares.append(i*i)
```

4.

```
squares = [i*i for i in range(10)]
```

Answer and Explanation:

- Correct option no. : **1, 2, 3**

- Explanation:
    - Options 1, 2, and 3 correctly generate squares from 1 to 10
    - Option 4 is incorrect as it starts from 0, generating [0, 1, 4, ..., 81], missing 100.

Question 8 (MSQ)

Which of the following operations with integers in Python are valid(won't give an error)?

Options:

1. 5 / 2 results in 2.5
2. 5 // 2 results in 2
3. 5 % 2 results in 1
4. -5 // 2 results in -3

Answer and Explanation:

- Correct option no. : **1, 2, 3, 4**
- All operations are valid and produce the stated results:
    - Option 1: Regular division (/) produces a float
    - Option 2: Floor division (//) rounds down to nearest integer
    - Option 3: Modulo (%) gives the remainder
    - Option 4: floor division rounds down.

Question 9 (MSQ)

You want to get user input for a list of numbers. Which of the following code snippets correctly accomplishes this, assumning the input is numeric value only?

Options:

1.

```
numbers = input("Enter numbers separated by spaces: ").split()


numbers = [int(x) for x in numbers]
```

2.

```
numbers = list(map(int, input("Enter numbers separated by spaces: ").split()))
```

3.

```
numbers = [int(x) for x in input("Enter numbers separated by spaces: ").split()]
```

4.

```
numbers = input("Enter numbers separated by spaces: ").split()


numbers = list(map(int, numbers))
```

Answer and Explanation:

- Correct option no. : **1, 2, 3, 4**
- All options correctly convert space-separated input into a list of integers.

Question 10

Which of the following typecasting operations in Python will produce an error?

Options:

1. int("3.14")
2. float("3.14")
3. str(3.14)
4. bool(0)

Answer and Explanation:

- Correct option no. : **1**
    - Option 1: int() cannot directly convert strings with decimal points
    - Option 2: float() can convert string representations of decimal numbers
    - Option 3: str() converts float to string representation
    - Option 4: bool() converts zero to False

Question 11 (MSQ)

You are creating a program to determine if a year is a leap year. Which of the following code snippets correctly implements this logic?

Options:

1.

```
if year % 4 == 0:


    print("Leap year")
```

```
else:

    print("Not a leap year")
```

2.

```
if year % 400 == 0:

    print("Leap year")

elif year % 100 == 0:

    print("Not a leap year")

elif year % 4 == 0:

    print("Leap year")

else:

    print("Not a leap year")
```

3.

```
print("Leap year" if year % 4 == 0 and year % 100 != 0 or year % 400 == 0 else "Not a leap year")
```

4.

```
is_leap = year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

print("Leap year" if is_leap else "Not a leap year")
```

Answer and Explanation:

- Correct option no. : **2, 3, 4**
- Option 1 is incorrect: Example: year = 1900 is not a leap year, but year = 2000 is. Try it in Colab and see different results

---

Question 12 (MSQ)

Which of the following options correctly describes the Caesar cipher?

Options:

1. It's a substitution cipher that replaces each letter with the letter a fixed number of positions down the alphabet.
2. It's an encryption technique where each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
3. It's a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers based on the letters of a keyword.
4. It's a highly complex encryption used in modern systems.

Answer and Explanation:

- Correct option no. : **1, 2**
- Options 1 and 2 correctly describe the Caesar cipher.

Question 13 (MSQ)

Which of the following Python functions correctly implements the Caesar cipher encryption?

Options:

1.

```
def caesar_encrypt(text, shift):

    return ''.join([chr((ord(char) - 65 + shift) % 26 + 65) if char.isupper() else

                    chr((ord(char) - 97 + shift) % 26 + 97) if char.islower() else char

                    for char in text])
```

2.

```
def caesar_encrypt(text, shift):

    return ''.join([chr((ord(char) - 65 - shift) % 26 + 65) if char.isupper() else
```

```
            chr((ord(char) - 97 - shift) % 26 + 97) if char.islower() else char

            for char in text])
```

3.

```
def caesar_encrypt(text, shift):

    return ''.join([chr((ord(char) + shift - 65) % 26 + 65) if char.isupper() else

            chr((ord(char) + shift - 97) % 26 + 97) if char.islower() else char

            for char in text])
```

4.

```
def caesar_encrypt(text, shift):

    return ''.join([chr(ord(char) + shift) if char.isalpha() else char for char in text])
```

Answer and Explanation:

- Correct option no. : **1, 3**
- Options 1 and 3 correctly implement the Caesar cipher encryption. Option 2 decrypts instead of encrypts. Option 4 doesn't handle wraparound correctly.

Question 14 (MSQ)

Which of the following statements about the Caesar cipher are true?

Options:

1. The Caesar cipher is considered a strong encryption method for modern use.
2. The Caesar cipher has 25 possible keys.
3. The Caesar cipher is vulnerable to frequency analysis attacks.
4. The Caesar cipher is a type of symmetric encryption.

Answer and Explanation:

- Correct option no. : **2, 3, 4**

- The Caesar cipher has 25 possible keys (shifts 1-25), is vulnerable to frequency analysis, and is symmetric. However, it's not considered strong for modern use.
- Search these terms on internet to get a basic idea about them.

Question 15 (MSQ)

Which of the following are true about breaking the Caesar cipher?

Options:

1. Brute-force attack is impractical due to the large number of possible keys.
2. Frequency analysis can be used to break the Caesar cipher without knowing the key.
3. The Caesar cipher can be broken by trying all 25 possible shifts.
4. Known-plaintext attack is ineffective against the Caesar cipher.

Answer and Explanation:

- Correct option no. : **2, 3**
- The Caesar cipher can be broken by frequency analysis or by trying all 25 shifts. Brute-force is practical due to the small key space, and known-plaintext attack is effective.
- Search these terms on internet to get a basic idea about them.