Eating Candy Bar Answer

Question)

Here is a program.

```
response = input("Do you want to eat a candy bar (Y/N)? ")

cost_of_candy_bar = 7
number_of_candy_bars = 0
total_cost = 0

SEE_QUESTION
    number_of_candy_bars = number_of_candy_bars + 1
    total_cost = total_cost + cost_of_candy_bar

    response = input("Do you want to eat one more (Y/N)? ")

print("You ate", number_of_candy_bars, "candy bars.")
print("They cost you", total_cost, "dollars.")
```

Here is the output showing the text input and output when the program is run. The text in bold and italic is the text entered by the user.

```
Do you want to eat a candy bar (Y/N)? Y
Do you want to eat one more (Y/N)? Y
Do you want to eat one more (Y/N)? Y
Do you want to eat one more (Y/N)? Y
Do you want to eat one more (Y/N)? N
You ate 4 candy bars.
They cost you 28 dollars.
```

The question is: **How many** of the following lines of code can be used to replace **SEE_QUESTION**? You just need to count how many of the lines of code shown below could be used to replace **SEE_QUESTION**.

For your answer, enter an integer number. Don't use a full stop or anything like that. Your answer will be a number in the range 0 to 10 inclusive.

- `while response == "Y":`
- `while response != "Y":`
- `while response == "N":`

- `while response != "N":`
- `while number_of_candy_bars != 4:`
- `while number_of_candy_bars == 4:`
- `while number_of_candy_bars < 4:`
- `while total_cost != 28:`
- `while total_cost < 35:`
- `while total_cost >= 28:`

Correct answer(s):

- 5

Explanation:

- This question focusses on producing the required output using one of the provided while conditions
- If you look at the given output, you know the loop should run its content 4 times to produce the output
- That means the while condition is evaluated 5 times, producing `True`, `True`, `True`, `True` and `False` when the loop is run
- Here are the values of each condition when the loop is run:
    - `response == "Y"`

      `True`, `True`, `True`, `True` and `False`
    - `response != "Y"`

      `False`, `False`, `False`, `False` and `True`
    - `response == "N"`

      `False`, `False`, `False`, `False` and `True`
    - `response != "N"`

      `True`, `True`, `True`, `True` and `False`
    - `number_of_candy_bars != 4`

      `True`, `True`, `True`, `True` and `False`
    - `number_of_candy_bars == 4`

      `False`, `False`, `False`, `False` and `True`
    - `number_of_candy_bars < 4`

      `True`, `True`, `True`, `True` and `False`
    - `total_cost != 28`

      `True`, `True`, `True`, `True` and `False`

- `total_cost < 35`

  `True`, `True`, `True`, `True` and `True`

- `total_cost >= 28`

  `False`, `False`, `False`, `False` and `True`

- As you can see, 5 of the conditions allow the loop to run its content four times

Lottery Game Answer

Question)

Chris wants to play a lottery game.

Here is how the game works. The lottery game draws **six unique numbers randomly from 1 to 49**. Anyone who guesses all six drawn numbers correctly wins the jackpot.

But Chris is too lazy to guess the numbers so he has written a Python program to help him, which is shown below.

```
import random

# Add the 49 balls in a list
balls = []
for number in range(1, 50):


    balls._____________________

# For displaying the output
order = ["first", "second", "third", \
         "fourth", "fifth", "sixth"]

# Randomly draw the balls
for ball in range(6):

    # Draw a ball from the remaining balls


    index = random.randint(0, _____________________ )

    number = balls[index]


    balls._____________________

    print("The", order[ball], "number is", number)
```

Here is an example of running the program:

Example

```
The first number is 27
The second number is 3
The third number is 38
The fourth number is 40
The fifth number is 11
The sixth number is 34
```

You need to fill in the three blanks in the program so that it can work as described above.

Correct answer(s):

- First blank:

  ```
  balls.append(number)
  ```

- Second blank:

  ```
  index = random.randint(0, len(balls) - 1)
  ```

- Third blank:

  ```
  balls.remove(number)
  ```

Explanation:

- For the first blank:
  - The objective is to add the numbers, from 1 to 49, to the `balls` list
  - One way is to simply add each number to the end of the list and therefore the `.append()` method can be used
  - Alternatively, you can also use the `.insert()` method to add each number, e.g. using `balls.insert(0, number)`
- For the second blank:
  - The objective is to randomly pick a number from the `balls` list
  - The index variable should contain the item index of the randomly picked number
  - Since the indices of the items in the list start from 0 to `len(balls)-1`, you should use `len(balls)-1` in the blank so that `random.randint()` covers the entire range of the indices
- For the third blank:
  - The objective is to remove the number that gets randomly picked so that the number will not be repeated later
  - That can be achieved by removing the number from the `balls` list and hence the use of the `.remove()` method

Number Indexing Answer

Question)

You know when you use *`list`*`.index(`*`item`*`)` on a list, it can only return the index of the first occurrence of *`item`*.

Here is a program demonstrating how to improve *`list`*`.index(`*`item`*`)`. It works by looking for all occurrences of a number from a list of numbers.

```
numbers = [ \
    28, 32, 30, 12, 6,  \
    6, 7, 9, 10, 17,    \
    37, 12, 41, 21, 37, \
    28, 25, 7, 30, 45   \
]

print("The list is:")
print(numbers)

number = int(input("Give me the number: "))

size = len(numbers)
while len(numbers) > 0:
    if numbers[0] == number:
        print(size - len(numbers), end=" ")

    numbers = numbers[ SEE_QUESTION ]
```

As you can see, the content of **`SEE_QUESTION`** is not shown in the above code.

Here are some examples of using the program. The text in bold and italics (e.g. ***`hello`***) is the text entered by the user.

Example 1

```
The list is:
[28, 32, 30, 12, 6, 6, 7, 9, 10, 17, 37, 12, 41, 21, 37, 28,
25, 7, 30, 45]
Give me the number: 6
4 5
```

Example 2

```
The list is:
[28, 32, 30, 12, 6, 6, 7, 9, 10, 17, 37, 12, 41, 21, 37, 28,
25, 7, 30, 45]
Give me the number: 30
2 18
```

Example 3

```
The list is:
[28, 32, 30, 12, 6, 6, 7, 9, 10, 17, 37, 12, 41, 21, 37, 28,
25, 7, 30, 45]
Give me the number: 28
0 15
```

What is the missing code that needs to replace **SEE_QUESTION**? You need to enter the missing code.

Correct answer(s):

- `1:`
- `1::`
- `1::1`
- `1:len(numbers)`
- `1:len(numbers):1`

Explanation:

- By examining the loop condition:

  ```
  while len(numbers) > 0:
  ```

  you can see the loop runs when there are items inside the `numbers` list
- In other words, the loop stops when the list is empty
- Therefore, the loop content needs to remove some items from `numbers` so that the loop condition can work correctly
- The loop does that using the last line of code inside the loop, which is the one that you need to work on:

  ```
  numbers = numbers[ SEE_QUESTION ]
  ```

- To remove items from the list, one way to do that is by using slicing, i.e.:

  ```
  numbers = numbers[ <start> : <end> : <step> ]
  ```

- From the if statement at the start of the loop, the number that the user is interested in, i.e. the `number` variable, is always compared against the first item of the list
- To make the program work,
  - you need to make sure the number is compared against every item in the list, not only the first number
  - that means the first item of the list cannot remain the same throughout the loop
- You can achieve this by removing the first item from the `numbers` list
- The simplest way to do that is by writing this slice notation:

  ```
  numbers = numbers[ 1 : ]
  ```

- You do not need to provide the end number and the step number because by default, slicing will extract items until the end of the list with a step number of 1
- You can also use the alternative answers as shown above in the correct answers