Week-3, Graded

This document has 11 questions.

Question-1

Statement

What is the output of the following snippet of code?

```python
for char in 'a1b2c3d4e5':
    if char in 'abcde':
        print('|', end = '') # there is no space between the quotes
        continue
    print(char, end = '')  # there is no space between the quotes
print('|')
```

Options

(a)

```
12345
```

(b)

```
|||||
```

(c)

```
|1|2|3|4|5
```

(d)

```
1|2|3|4|5|
```

(e)

```
|1|2|3|4|5|
```

Answer

(e)

Feedback

Whenever you hit a letter, print `|`. If not, print the character as it is.

As a side note, we are not using the `range` function to iterate over the indices of the string here. Instead, we are directly iterating over the characters of the string.

Question-2

Statement

Select the correct implementation of a program that accepts a positive integer x as input and prints the maximum value of the integer y such that $2^y \leq x$.

Sample Test Cases

Input	Output
100	6
256	8

Options

(a)

```python
x = int(input())
y = 0
while x > 1:
    x = x // 2
    y = y + 1
print(y)
```

(b)

```python
x = int(input())
y = 0
while x >= 1:
    x = x // 2
    y = y + 1
print(y)
```

(c)

```python
x = int(input())
y = 0
while x > 1:
    x = x / 2
    y = y + 1
print(y)
```

(d)

```python
x = input()
y = 0
while x > 1:
    x = x // 2
    y = y + 1
print(y)
```

Answer

(a)

Feedback

The basic mathematical idea is as follows. For any positive integer x, we can always find a y such that:

$$2^y \leq x < 2^{y+1}$$

That is, every number should lie between two consecutive powers of 2. From this it follows that:

$$2^{y-1} \leq \left\lfloor \frac{x}{2} \right\rfloor < 2^y$$

The exercise for you is to connect this mathematical insight with the option (a). We have used floor function in the inequality. This corresponds to the `//` operator in Python. Why is it important to do floor division? Take the case of a number that is not a power of 2, say 10, and run the code given in option-(c). What happens?

Now, for the wrong options:

- Option-(b) is quite close to option-(a), but has the `>=` sign in the while condition. This will make the loop go through one more iteration than what is needed.
- Option-(c) does normal division and not floor division.

Why is option-(d) wrong? This is left as an exercise for you.

Question-3

Statement

Consider the following snippet of code:

```python
x = int(input())
i = 0
while x % (10 ** i) != x:
    i = i + 1
print(i)
```

What will the variable `i` represent at the end of execution where `x` is a positive integer?

Options

(a)

Number of zeros in `x`

(b)

Number of ones in `x`

(c)

Number of digits in `x`

(d)

Number of non-zero digits

Answer

(c)

Feedback

To understand the answer, you have to look at two mathematical facts:

- If a positive integer x has n digits, then it has to lie between these two powers of 10:

$$10^{n-1} \leq x < 10^n$$

- If $y \leq x$, then the remainder when x is divided by y will always be less than x. On the other hand, if $y > x$, then the remainder when x is divided by y will always be equal to x.

Try to work with these two facts to see why the answer is option-(c).

Question-4

Statement

What is the output of the following snippet of code?

```python
alpha = 'abcdefghijklmnopqrstuvwxyz'
shift = 5
word = 'python'
encoded_word = ''  # there is no space between quotes
for char in word:
    shifted_index = (alpha.index(char) + shift) % 26
    encoded_char = alpha[shifted_index]
    encoded_word += encoded_char
print(encoded_word)
```

Options

(a)

```
stmydu
```

(b)

```
tcxlsr
```

(c)

```
veznut
```

(d)

```
udymts
```

Answer

(d)

Feedback

Each character is shifted right by five units. If this code is not clear to you, check out the video on Caesar cipher that was covered in week-2. The only difference here is that the encoding process is done for each character of a string in a loop.

Question-5

Statement

Consider the following snippet of code.

```python
name = input()
nick = ''     # there is no space between the quotes
space = ' ' # there is one space between the quotes
first_char = True
for char in name:
    if first_char == True:
        nick = nick + char
        first_char = False
    if char == space:
        first_char = True
print(nick)
```

What is the output for the following input?

```
Albus Percival Brian Wulfric Dumbledore
```

Options

(a)

```
Albus
```

(b)

```
Dumbledore
```

(c)

```
AP
```

(d)

```
APBWD
```

Answer

(d)

Feedback

When does `first_char` become `True`? When does it become `False`? If you can answer these two questions, then you are halfway on the path to understanding the code. The names of variables can often help you figure out their purpose in the code. The two variables names — `first_char` and `nick` — have a clear meaning. Can you make an informed guess as to what they stand for?

Note

This method of using a Boolean variable such as `first_char` to determine the **state** of the code is explained in detail in approach-2 of PPA-4 in the PPA-hints document. Refer to it for more details. Once you read that section, get back to this problem and identify what the state stands for here.

Common Data for questions 6, 7

Statement

Common Data for questions (6) and (7)

Consider the following snippet of code:

```
word = input()
valid = True
for i in range(len(word)):
    char = word[i]
    if i % 2 == 0 and char not in 'aeiou':
        valid = False
print(valid)
```

Question-6

Statement

Select all inputs for which the code prints `True` as the output. (MSQ)

Options

(a)

```
abet
```

(b)

```
enamel
```

(c)

```
eatery
```

(d)

```
onetime
```

Answer

(a), (b), (d)

Feedback

According to the code, a `word` is valid if and only if every character at an even index is a vowel.

Question-7

Statement

Assume that a ten letter word is passed as input to the code. If the output is `True`, then which of the following statements about the input word are true?

Options

(a)

The word has exactly five vowels.

(b)

The word has have at least five vowels.

(c)

The letters at even indices are vowels. Assume that we use zero-based indexing.

(d)

Every vowel in the word appears only at even indices. Assume that we use zero-based indexing.

Answer

(b), (c)

Feedback

Options (a) and (d) are clearly not supported. A good counter example is the string input: `aaaa`

Question-8

Statement

There is a collection of boxes, each box containing certain number of coins. This information is represented as a string such as this: `'|1|4|1|5|9|'`. Here, there are five boxes. The first box has one coin, second has four coins and so on. Assume that each box has at least one coin and at most nine coins.

Select the correct implementation of a snippet of code that computes the average number of coins across the boxes and stores it in a variable named `avg`. Assume that the string `boxes` is already given to you and that there is at least one box in the collection.

Options

(a)

```python
# Sample initialization of boxes: '|1|4|1|5|9|'
# Assume that boxes is provided to you
num = 0
total = 0
for i in range(len(boxes)):
    if i % 2 == 0:
        continue
    coins = boxes[i]
    total += coins
    num += 1
avg = total / num
```

(b)

```python
# Sample initialization of boxes: '|1|4|1|5|9|'
# Assume that boxes is provided to you
num = 0
total = 0
for i in range(len(boxes)):
    if i % 2 == 0:
        continue
    coins = int(boxes[i])
    total += coins
    num += 1
avg = total / num
```

(c)

```python
# Sample initialization of boxes: '|1|4|1|5|9|'
# Assume that boxes is provided to you
num = 0
total = 0
for coins in boxes:
    total += coins
    num += 1
avg = total / num
```

(d)

```python
# Sample initialization of boxes: '|1|4|1|5|9|'
# Assume that boxes is provided to you
num = 0
total = 0
for i in range(len(boxes)):
    if i % 2 == 0:
        break
    coins = int(boxes[i])
    total += coins
    num += 1
avg = total / num
```

Answer

(b)

Feedback

A key observation is this: the vertical bars appear at even indices. Do you now see the relevance of these two lines in the code?

```python
#################
    if i % 2 == 0:
        continue
#################
```

Why are we using the keyword `continue` and not `break`? Can we replace `continue` with `break`? Will the code still work?

The rest of the code is quite easy to parse and a patient perusal of the same will convince you of the correctness of option-(b) Some of the mistakes that other options have:

- Option-(a) `boxes` is a string and `boxes[i]` is also a string. You can't directly add an integer and a string. `boxes[i]` must first be converted into an integer.
- Option-(c) the separator `|` is completely ignored here.
- Option-(d) the keyword should be `continue` and not `break`.

Question-9

Statement

The first five terms of the Fibonacci sequence is given below.

$$F_1 = 1$$
$$F_2 = 1$$
$$F_3 = 2$$
$$F_4 = 3$$
$$F_5 = 5$$

We wish to write a program that accepts a positive integer n as input and prints F_n as the output. Select all correct implementations of this program. (MSQ)

Options

(a)

```python
n = int(input())
F_prev = 1
F_curr = 1
count = 2
while count < n:
    temp = F_prev + F_curr
    F_prev = F_curr
    F_curr = temp
    count += 1
print(F_curr)
```

(b)

```python
n = int(input())
if n <= 2:
    print(1)
else:
    F_prev = 1
    F_curr = 1
    count = 2
    while count < n:
        temp = F_prev + F_curr
        F_prev = F_curr
        F_curr = temp
        count += 1
    print(F_curr)
```

(c)

```python
n = int(input())
F_prev = 1
F_curr = 1
for i in range(n):
    temp = F_prev + F_curr
    F_prev = F_curr
    F_curr = temp
print(F_curr)
```

(d)

```python
n = int(input())
F_prev = 1
F_curr = 1
for i in range(n - 2):
    temp = F_prev + F_curr
    F_prev = F_curr
    F_curr = temp
print(F_curr)
```

Answer

(a), (b), (d)

Feedback

`F_prev` and `F_curr` correspond to two consecutive terms in the sequence. `F_prev + F_curr` is the next term in the sequence. So, we need to perform the following updates:

(1) `F_prev + F_curr` will be the new `F_curr` and

(2) `F_prev` will be assigned the current value of `F_curr`

If this is not clear, consider the following table:

	Current	**Next**
`F_prev`	`x`	`y`
`F_curr`	`y`	`x + y`

If these operations are done sequentially, say (1) followed by (2), update (2) would fail as we no longer have access to the old value of `F_curr`. On the other hand, if (2) is performed first followed by (1), the first operation would fail as we no longer have access to the old value of `F_prev`.

In some sense, both these updates have to be done simultaneously. This is where a `temp` variable comes in handy. Convince yourself that the use of `temp` variable circumvents the problems posed by sequential updates.

As a final point, the interesting thing is that you don't need the `if-else` condition for $n = 1$ and $n = 2$. Options (a) and (d) show the way to do it. Option-(b) is also correct, but makes the code less elegant. Why is option-(c) wrong? This is left as an exercise for you.

Common Data for questions 10, 11

Statement

Common Data for questions (10) and (11)

Consider the following snippet of code.

```python
for x in range(100):
    for y in range(100):
        if x != y:
            print(f'{x},{y}')
```

Question-10

Statement

When the code given above is executed, how many lines will the output have? (NAT)

Answer

9900

Feedback

In a nested loop that has two levels where each loop iterates through the same number of elements, say n, a normal statement inside the inner loop will usually be executed n^2 times. Here, line-3 is executed n^2 times. However, line-4 is within an if-statement and it will not be executed whenever $x == y$, so, we need to subtract n from n^2. There are n occasions when $x == y$. Setting $n = 100$, we have: $100 \times 100 - 100 = 9900$ as the final answer.

Question-11

Statement

Among the code blocks given below, select the correct equivalent implementation of the code given in the common data.

Options

(a)

```python
x, y = 0, 0
while x < 100:
    while y < 100:
        if x != y:
            print(x, y)
```

(b)

```python
x, y = 0, 0
while x < 100:
    while y < 100:
        if x != y:
            print(x, y)
        x += 1
        y += 1
```

(c)

```python
x, y = 0, 0
while x < 100:
    while y < 100:
        if x != y:
            print(x, y)
    y += 1
    x += 1
```

(d)

```python
x, y = 0, 0
while x < 100:
    while y < 100:
        if x != y:
            print(x, y)
        y += 1
    x += 1
```

(e)

```python
x, y = 0, 0
while x < 100:
    y = 0
    while y < 100:
        if x != y:
            print(x, y)
        y += 1
    x += 1
```

Answer

(e)

Feedback

The last option is the correct one. Now start from option (a) and go all the way up to option (d). Compare each of these options with option (e). This will reveal all the errors to you. A brief demonstration is given for your reference:

- In some ways, option-(a) has some very serious errors. `x` and `y` are never incremented!
- There is an attempt to correct this mistake in option-(b). Unfortunately, `x` and `y` are being incremented in the wrong place.
- The same problem persists in option-(c).
- Option-(d) corrects the position of increment operations. However, it still misses one crucial step, that of resetting `y` to zero within the inner loop.