CS-119(h) Midterm: Solutions for the Programming Questions

Question 9

```
x = 13
z = -8
y = z + 21

a = x == y
b = x == y - z
c = x > True > z

if a and b and c:
    if x - y != False:
        print("banana")
    else:
        print("ananas")
elif a or b or c:
    if x - y:
        print("penguin")
    else:
        print("unicorn")
else:
    print("pizza")
```

☐ ananas ☐ penguin ☐ banana ☒ unicorn ☐ pizza

Solution

The code initializes three variables: `x = 13`, `z = -8` and `y = z + 21 = 13`.
The code then defines three boolean variables, `a`, `b` and `c`, and assignes them to the following values:

- `a = True` because `x` and `y` are both `13`.
- `b = False` because `y - z = 13 - (-8) = 21` and `x = 13`.
- `c = True` because the expression `x > True > z` is equivalent to `(x > True) and (True > z)`. The numerical value of `True` is one. So, `(x > True) = (13 > 1)`, which is `True`. Similarly, the second expression becomes `(True > z) = (1 > -8)`, which is also `True`.

The next step is to use the values in the conditional statements.

- `if a and b and c`: This condition checks if `a`, `b`, and `c` are all `True`. Because `b = False`, we jump to the `elif` statement.
- `elif a or b or c`: This condition checks if any of `a`, `b` or `c` is `True` (it is enough to have at least one variable set to `True`, such that the condition is met). Because `a = True`, this condition holds and we jump to the next line, `if x - y`.
- `if x - y`: If we substitute the correct values for `x` and `y`, we obtain `if 0`, which translates to `if False`. Hence, this condition fails and we jump to the `else` condition, which outputs the string `"unicorn"`.

Question 10

```
numbers = [3, 6, 9, 12, 15]
result = 0

for i, v in enumerate(numbers):
    if i % 2 == 0:
        result += v
    elif i % 3 == 0:
        result -= v

print(result)
```

☑ 15 ☐ 3 ☐ 9 ☐ -9 ☐ 12

Solution

The code iterates over the list `numbers` and modifies the value of `result` based on the index `i` of each element `v`. Here is how the `result` is modified after each iteration.

- **Iteration 0:** `i = 0`, `v = 3`
  - Because `i % 2 == 0`, we add `v` to `result`.
  - `result = 0 + 3 = 3`
- **Iteration 1:** `i = 1`, `v = 6`
  - Neither `i % 2 == 0` nor `i % 3 == 0`, so `result` remains unchanged.
  - `result = 3`
- **Iteration 2:** `i = 2`, `v = 9`
  - Because `i % 2 == 0`, we add `v` to `result`.
  - `result = 3 + 9 = 12`
- **Iteration 3:** `i = 3`, `v = 12`
  - Because `i % 3 == 0`, we subtract `v` from `result`.
  - `result = 12 - 12 = 0`
- **Iteration 4:** `i = 4`, `v = 15`
  - Because `i % 2 == 0`, we add `v` to `result`.
  - `result = 0 + 15 = 15`

Hence, the program outputs the value `15`.

Question 11

```
matrix = [[i+2 * j for j in range(4)] for i in range(4)]
print(matrix)
```

- [x] `[[0, 2, 4, 6], [1, 3, 5, 7], [2, 4, 6, 8], [3, 5, 7, 9]]`
- [ ] `[[0, 1, 2, 3], [2, 3, 4, 5], [4, 5, 6, 7], [6, 7, 8, 9]]`
- [ ] `[[3, 5, 7, 9], [4, 6, 8, 10], [5, 7, 9, 11], [6, 8, 10, 12]]`
- [ ] `[[0, 2, 4, 6], [0, 3, 6, 9], [0, 4, 8, 12], [0, 5, 10, 15]]`
- [ ] `[[0, 0, 0, 0], [2, 3, 4, 5], [4, 6, 8, 10], [6, 9, 12, 15]]`

Solution

The code generates a 4×4 matrix using nested list comprehensions as follows:

- `for i in range(4)`: This loop is the outer list comprehension, creating 4 rows.
- `for j in range(4)`: For each value of `i`, this loop iterates over `j = 0, 1, 2, 3`, creating 4 columns for each row. Each element in the row is computed using the expression `i + 2j`.

We compute the values for each `i` to determine the matrix values. So, each row is as follows:

- **Iteration 0:** `i = 0`
$$[0 + 2 \cdot 0,\ 0 + 2 \cdot 1,\ 0 + 2 \cdot 2,\ 0 + 2 \cdot 3] = [0, 2, 4, 6]$$
- **Iteration 1:** `i = 1`
$$[1 + 2 \cdot 0,\ 1 + 2 \cdot 1,\ 1 + 2 \cdot 2,\ 1 + 2 \cdot 3] = [1, 3, 5, 7]$$
- **Iteration 2:** `i = 2`
$$[2 + 2 \cdot 0,\ 2 + 2 \cdot 1,\ 2 + 2 \cdot 2,\ 2 + 2 \cdot 3] = [2, 4, 6, 8]$$
- **Iteration 3:** `i = 3`
$$[3 + 2 \cdot 0,\ 3 + 2 \cdot 1,\ 3 + 2 \cdot 2,\ 3 + 2 \cdot 3] = [3, 5, 7, 9]$$

Hence, the final output is `[[0, 2, 4, 6], [1, 3, 5, 7], [2, 4, 6, 8], [3, 5, 7, 9]]`.

Question 12

```
matrix = [[1, 0, 3, 4], [5, 6, 0, 8], [9, 10, 11, 0], [0, 14, 15, 16]]
result = 0
i = 0

while i < len(matrix):
    if i:
        for j in range(len(matrix[i])):
            if matrix[i][j] == 0:
                result += matrix[j][i]
        i += 1
    i += 1
print(result)
```

- ☐ 29
- ☐ 34
- ☐ Ce programme n'affice rien car la boucle `while` ne se termine jamais
- ■ 14
- ☐ 20

Solution

The code computes a sum, stored in `result`, based on specific elements of `matrix`. For a better visualization, the `matrix` looks like the following:

$$\texttt{matrix} = \begin{bmatrix} 1 & 0 & 3 & 4 \\ 5 & 6 & 0 & 8 \\ 9 & 10 & 11 & 0 \\ 0 & 14 & 15 & 16 \end{bmatrix}$$

Initially, we have `result = 0` and `i = 0`. A `while` loop iterates over rows. The inner `for` loop only executes if `i` is non-zero. For each element in row `i`, if `matrix[i][j] == 0`, the transposed element `matrix[j][i]` is added to `result`. Be careful that `i` is incremented in two lines within the `while` loop.

The outcome of each iteration is explained below:

- **Iteration 0:** Because `i = 0`, the condition of the `if` statement fails and the code just increments `i`. We have `i = 1`.
- **Iteration 1:** The code enters the `if` body. Because `matrix[1][2] == 0`, we update `result = result + matrix[2][1] = 0 + 10 = 10`. After this iteration, we have `i = 3`.
- **Iteration 2:** Because `matrix[3][0] == 0`, we update `result = result + matrix[0][3] = 10 + 4 = 14`. After this iteration, `i = 5`, and because `i` exceeds the length of the matrix, we do not continue further.

Hence, the program outputs the value `result`, which is `14`.

Question 13

```
a = [i // 2 for i in range(1, 13, 3)]
b = [i * 2 for i in a]
a.extend(b)
result = sum(a)
for i, v in enumerate(b):
    result += v + b[i]
print(result)
```

☐ 112 ☒ 70 ☐ 77.0 ☐ 112.0 ☐ 50

Solution

The code creates the list `a` using list comprehension. The `range` generates the values `1, 4, 7, 10`.

`a = [1 // 2, 4 // 2, 7 // 2, 10 // 2] = [0, 2, 3, 5]`

The list `b` is generated by doubling each element in `a`.

`b = [0 × 2, 2 × 2, 3 × 2, 5 × 2 ] = [0, 4, 6, 10]`

The next step extends the list `a` with all the elements from `b`. Hence, `a = [0, 2, 3, 5, 0, 4, 6, 10]`. For computing the initial vlaue of `result`, we add all elements of `a`. So, `result = 0 + 2 + 3 + 5 + 4 + 5 + 10 = 30`. Then, the `for` loop iterates over `b` with `enumerate`, adding `v + b[i]` to `result`:

- **Iteration 0:** `i = 0`, `v = 0`
  - We add `v` and `b[0]` to `result`.
  - `result = 30 + 0 + 0 = 30`
- **Iteration 1:** `i = 1`, `v = 4`
  - We add `v` and `b[1]` to `result`.
  - `result = 30 + 4 + 4 = 38`
- **Iteration 2:** `i = 2`, `v = 6`
  - We add `v` and `b[2]` to `result`.
  - `result = 38 + 6 + 6 = 50`
- **Iteration 3:** `i = 3`, `v = 10`
  - We add `v` and `b[3]` to `result`.
  - `result = 50 + 10 + 10 = 70`

Hence, the program outputs the value `result`, which is `70`.

Question 14

```
x = c if (c > b) else b
x = a if (a > b and a > c) else x
y = min(-a, -b)
result = min(y, -c)
result -= x
```

- ☐ `max(a, b, c) - min(-a, -b, -c)`
- ☐ `min(-a, -b, -c) - min(-a, -b, -c)`
- ☐ `max(a, b, c) + min(a, b, c)`
- ☐ `min(a, b, c) - max(-a, -b, -c)`
- ☒ `2*min(-a, -b, -c)`

Solution

After the first line, `x = max(c, b)`. In the second line, we change the value of `x` only if `a` is greater than both `b` and `c`. Hence, the first two lines compute `x = max(a, b, c)`. With the following two lines, we obtain `result = min(-a, -b, -c)`, because `result = min(y, -c) = min(min(-a, -b), -c)`. The final line computes `result = min(-a, -b, -c) - max(a, b, c)`. Note that `min(-a, -b, -c) = - max(a, b, c)`, because it selects the most negative (i.e., the smallest) value among `-a, -b, -c`. So, we can rewrite `result = min(-a, -b, -c) - max(a, b, c) = min(-a, -b, -c) + min(-a, -b, -c) = 2*min(-a, -b, -c)`, which is the final answer.

Question 15

```
numbers = [0, 1, 2, 3, 4, 5, 6]
a = [x*x**2 for x in numbers]
b = [x and numbers[0] for x in numbers]
a.extend(b[-2:])
print(sum(a[1::3]), a[-1])
```

☑ `65 0` ☐ `257 False` ☐ `257 0` ☐ `65 216` ☐ `65 False`

Solution

The list `a` is created using list comprehension. For each element `x` in `numbers`, we compute $\texttt{x} \cdot \texttt{x}^2 = \texttt{x}^3$.

$$\texttt{a} = [0^3, 1^3, 2^3, 3^3, 4^3, 5^3, 6^3] = [0, 1, 8, 27, 64, 125, 216]$$

The list `b` is created using list comprehension. For each element `x` in `numbers`, we compute `x and numbers[0]`. Because `numbers[0] = 0`, which is interpreted as false, for any value of `x`, `x and numbers[0] = 0`. Note that the elements of `b` are zero and not `False` because `numbers[0]` is an integer, not a boolean.

$$\texttt{b} = [0, 0, 0, 0, 0, 0, 0]$$

`a.extend(b[-2:])` appends the last two elements of `b` to `a`:

$$\texttt{a} = [0, 1, 8, 27, 64, 125, 216, 0, 0]$$

`a[1::3]` starts from index 1 and selects every third element in `a`:

$$\texttt{a[1::3]} = [\texttt{a[1]}, \texttt{a[4]}, \texttt{a[7]}] = [1, 64, 0]$$

The sum of these elements is 65.
The expression `a[-1]` refers to the last element of `a`, which is `0`.
In conclusion, the final output is `65 0`.

Question 16

```
count = 0
for i in range(3):
    for j in range(3):
        if i == 1 and j == 0:
            count += 1
            break
        if i == j:
            continue
        count += 1
print(count)
```

☐ 3 ☐ 7 ☐ 4 ☑ 5 ☐ 10

Solution

The outer `for` loop iterates over `i` in the range $[0, 1, 2]$. For each value of `i`, the inner `for` loop iterates over `j` in the range $[0, 1, 2]$. In each iteration, the code does the following operations:

- **Iteration 0:** `i = 0, j = 0`
  - Because `i == j`, we `continue` and increment`j`.
  - `count = 0`
- **Iteration 1:** `i = 0, j = 1`
  - None of the conditions for the `if` statements hold. So, we only execute `count += 1` and increment`j`.
  - `count = 0 + 1 = 1`
- **Iteration 2:** `i = 0, j = 2`
  - None of the conditions for the `if` statements hold. So, we only execute `count += 1`. Moreover, because we finish the inner `for` loop, we go to the next value of `i` from the outer `for` loop and start again the inner `for` loop.
  - `count = 1 + 1 = 2`
- **Iteration 3:** `i = 1, j = 0`
  - Because `i = 1` and `j = 0`, the condition for the first `if` statement holds. So, we increment `count` and then `break`. So, we terminate the inner `for` loop and go to the next value of `i` from the outer `for` loop and start again the inner `for` loop.
  - `count = 2 + 1 = 3`
- **Iteration 4:** `i = 2, j = 0`
  - None of the conditions for the `if` statements hold. So, we only execute `count += 1` and increment`j`.
  - `count = 3 + 1 = 4`
- **Iteration 5:** `i = 2, j = 1`
  - None of the conditions for the `if` statements hold. So, we only execute `count += 1` and increment`j`.
  - `count = 4 + 1 = 5`
- **Iteration 6:** `i = 2, j = 2`
  - Because `i == j`, we `continue` and finish both `for` loops. Moreover, we do not increment `count`.
  - `count = 5`

So, the final value of `count` is `5`.