Editorial-Mock : Mid Module A Quiz - 2

Question 1

In a list of customer IDs, which search algorithm would divide the list in half at each step to quickly locate a specific ID?

Options:

- A) Linear Search
- B) Bubble Sort
- C) Binary Search
- D) Merge Sort

Answer: C) Binary Search

Explanation:
Binary Search works by repeatedly dividing a sorted list in half. It compares the target value with the middle element and eliminates half of the remaining elements based on whether the target is greater or less than the middle element. Linear Search (A) checks each element sequentially, while Bubble Sort (B) and Merge Sort (D) are sorting algorithms, not search algorithms.

Question 2

When the user wants to get the work done faster, which sorting algorithm has the best time complexity for sorted data?

Options:

- A) Bubble Sort (unoptimized version)
- B) Quick Sort
- C) Insertion Sort
- D) Merge Sort

Answer: C) Insertion Sort

Explanation:
For already sorted data, Insertion Sort has a time complexity of O(n) because it requires only one pass to confirm that the list is sorted, making minimal swaps. In contrast, Bubble Sort (A) is O(n^2) (unless optimized), Quick Sort (B) averages O(n log n) but still performs partitions, and Merge Sort (D) always operates in O(n log n).

Question 3

Given we want to print some numbers with the help of a loop and condition, what will be the output of the following Python code snippet?

even_numbers = [x for x in range(20) if x % 2 == 0]

print("Even numbers:", even_numbers)

Options:

- A) Even numbers: [1, 3, 5, 7, 9]
- B) Even numbers: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
- C) Even numbers: [0, 1, 2, 3, 4]
- D) Even numbers: [2, 4, 6, 8, 10]

Answer: B) Even numbers: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Explanation:
The list comprehension iterates over numbers 0 to 19 and selects only those where x % 2 == 0 (even numbers). Thus, it produces even numbers from 0 up to 18.

Question 4

If you are reading a file in binary mode using open('file.bin', 'rb') and the file has 1024 bytes, what will be the length of f.read(512) if called once?

Answer: 512

Explanation:
In binary mode, f.read(512) reads exactly 512 bytes from the file. Since the file is 1024 bytes long and the file pointer is at the beginning, the read returns 512 bytes.

Question 5

Given a sorted array of size n = 8, how many comparisons are made in the worst-case scenario when using the binary search algorithm to find a target value?

Options:

- A) 2
- B) 4
- C) 5
- D) 8

Answer: B) 4

Explanation:
Binary search halves the array with each comparison. For an array of size 8, the worst-case scenario takes $\log_2(8) + 1 = 3 + 1 = 4$ comparisons.

Question 6

A company maintains a **log file** (sales_data.txt) where each line contains a daily sales amount as an integer. The accounting team wants to calculate the **total sales** for the month by reading this file line by line.
Which of the following correctly describes the use of the while loop and file handling to sum integers from a file?

Options:

- A) The while loop continues reading lines from the file until it reaches the end, converting each line to an integer and adding it to the sum.
- B) The while loop continues indefinitely until manually stopped, without file handling.
- C) The while loop reads all lines at once using readlines() and stores them in a list.
- D) The while loop can only be used to write data to a file, not to read data.

Answer: A) The while loop continues reading lines from the file until it reaches the end, converting each line to an integer and adding it to the sum.

Explanation:
Using a while loop with readline() allows you to read one line at a time until reaching the end of the file. Each line is converted to an integer and accumulated in a running total.

Question 7

If a developer opens a log file using mode 'a+', what operations can they perform?

Options:

- A) Only write operations
- B) Only read operations
- C) Both read and write operations
- D) No operations

Answer: C) Both read and write operations

Explanation:
The 'a+' mode opens the file for both appending (writing) and reading. Although the file

pointer starts at the end, it can be repositioned (for example, using seek(0)) to read existing content.

Question 8

If a file named "data.txt" contains 3 lines, how many lines will it contain after adding 2 more using append mode?

Answer: 5

Explanation:
Append mode ('a') adds new content to the end of the file without altering existing data. Therefore, adding 2 new lines to an existing 3-line file results in 5 lines.

Question 9

While working on a real-time project, an engineer encounters many errors. Which of the following statements about Python's exception handling is true?

Options:

- A) The finally block is executed only if an exception is raised.
- B) The finally block is executed only if an exception is not raised.
- C) A try block can have multiple except blocks.
- D) You can define custom exceptions by inheriting from Exception.

Answer: C) & D)

Explanation:

- Option C is correct: A try block can have multiple except blocks to handle different types of exceptions.
- Option D is also correct: Custom exceptions can be created by subclassing Exception. The finally block executes regardless of whether an exception occurs, so options A and B are incorrect.

Question 10

While analyzing a product with zero sales, a programmer encounters the following code. What will be the output?

```
try:
   print(5 / 0)
except ZeroDivisionError:
    print("Error")
```

Options:

- A) 0
- B) None
- C) Error
- D) ZeroDivisionError

Answer: C) Error

Explanation:
Division by zero raises a ZeroDivisionError. The except block catches the error and prints "Error". Options A and B are not applicable, and option D would be the result only if the exception object were printed.

Question 11

You are developing a **log management system** for a web application. The system should:

1. **Read existing logs** to analyze errors.
2. **Append new log entries** as the application runs.
3. **Overwrite logs** when a new session starts.

You want to perform operations to handle files. Which of the following are valid file modes in Python?

Options:

- A) r+
- B) a
- C) write
- D) w+

Answer: A) r+, B) a, D) w+

Explanation:

- r+ opens a file for reading and writing (file must exist).
- a opens a file for appending (creates the file if it doesn't exist).
- w+ opens a file for writing and reading (overwrites the file or creates a new one). write is not a valid file mode; the correct mode for writing is w.

Question 12

Multiple search algorithms are compared based on time complexity. What is the worst-case time complexity of linear search?

Options:

- A) O(1)
- B) O(log n)
- C) O(n)
- D) O(n^2)

Answer: C) O(n)

Explanation:
In the worst-case scenario, linear search examines every element in the list once, resulting in a time complexity of O(n).

Question 13

To sort students who have scored well across the city, a developer opts for binary search. What is the primary advantage of binary search over linear search?

Options:

- A) It works on unsorted lists.
- B) It has a worst-case time complexity of O(log n).
- C) It's easier to implement.
- D) It uses less memory.

Answer: B) It has a worst-case time complexity of O(log n).

Explanation:
Binary search is much more efficient than linear search, with a worst-case time complexity of O(log n), provided that the list is sorted.

Question 14

Imagine you are debugging code that handles exceptions in a financial application. Consider the following snippet:

```
try:
   raise Exception("Error")
   print(1 / 0)
except ZeroDivisionError as e:
   print("Error:", e)
except ArithmeticError:
   print("Arithmetic error")
```

What will happen when this code is executed?

Options:

- A) No Output
- B) Arithmetic error
- C) Error: division by zero
- D) It will throw an exception error with description "Error"

Answer: D) It will throw an exception error with description "Error"

Explanation:
The line raise Exception("Error") immediately raises a generic exception before the division by zero can occur. Since neither the ZeroDivisionError nor the ArithmeticError handlers catch this generic exception, the program will crash and display "Error".

Question 15: Python Exception Handling Concepts

You are reviewing Python's exception handling mechanisms to ensure your code is robust and well-structured. During a team discussion, the following statements are proposed:

Which of the following statements are true regarding Python's exception handling?

- **A)** The except block can catch multiple exception types using a tuple.
- **B)** A finally block is optional in a try-except structure.
- **C)** Raising an exception inside an except block will suppress the original exception.
- **D)** The raise statement without an argument re-raises the last exception inside an except block.

Answer: A), B), & D)

Explanation:

- **A) True** – You can catch multiple exceptions in a single except block using a tuple, e.g., except (ValueError, TypeError).
- **B) True** – The finally block is optional; a try block can exist with just an except block.
- **C) False** – Raising an exception inside an except block does not necessarily suppress the original exception unless explicitly handled.

- **D) True** – The raise statement without arguments inside an except block re-raises the last caught exception.

Question 16

A ride-sharing company calculates a trip fare based on distance: a base fare of $5, plus $2 per mile for the first 10 miles and $1 per mile thereafter. Given the following code, what is the output if the distance is 15 miles?

```
distance = 15  # in miles

base_fare = 5

fare = base_fare + (distance * 2 if distance <= 10 else 10 * 2 + (distance - 10) * 1)

print(fare)
```

Options:

1. 35
2. 30
3. 25
4. 40

Answer: 30

Explanation:
For a 15-mile trip:

- Base fare: $5
- First 10 miles: 10 × $2 = $20
- Remaining 5 miles: 5 × $1 = $5
  Total = 5 + 20 + 5 = $30.

Question 17

Managing the Employee Database
Given the following employee database:

```
employees = {

  "101": {"name": "Alice", "position": "Engineer", "age": 28},

  "102": {"name": "Bob", "position": "Manager", "age": 34},

  "103": {"name": "Charlie", "position": "Analyst", "age": 29}

}
```

Which code snippet correctly calculates the average age of all employees?

Options:

- A)
- total_age = sum([employee["age"] for employee in employees])
- average_age = total_age / len(employees)
- B)
- average_age = sum(employee["age"] for employee in employees.values()) / len(employees)
- C)
- average_age = sum(employees["age"]) / len(employees)
- D)
- total = sum([emp.age for emp in employees.values()])
- average_age = total / len(employees)

Answer: Option B

Explanation:
Option B correctly iterates over the dictionary values using employees.values(), sums the "age" from each, and divides by the number of employees to compute the average.

Question 18

The Treasure Hunt 🏴 ☠️

A group of adventurers must gain entry into a magical cave. The guardian statue's decision-making is determined by the following code:

```
time_of_day = "night"

has_lantern = True

has_map = False

temperature = 28  # in degrees Celsius

has_key = False


if time_of_day == "day":

    if 20 < temperature <= 30:

        if has_key:
            decision = "Enter the cave"
        elif has_lantern or has_map:
            decision = "Enter with caution"
        else:
            decision = "Wait"
    else:
        decision = "Wait"
elif time_of_day == "night":
    if has_key:
        decision = "Enter the cave"
    elif has_lantern and has_map:
        if temperature > 25:
            decision = "Enter with caution"
        else:
            decision = "Wait for sunrise"
    elif has_lantern or has_map:
        if temperature < 25:
            decision = "Wait for sunrise"
        else:
            decision = "Wait"
    else:
        decision = "Locked out"
else:
    decision = "Invalid time"

print(decision)
```

Based on the code and the given inputs, what are the **possible outputs** if any of the input values (time_of_day, has_lantern, has_map, temperature, has_key) are changed?

Options:

- a) Enter the cave
- b) Enter with caution

- c) Wait
- d) Wait for sunrise
- e) Locked out
- f) Invalid time

Answer: a, b, c, d, e, f

Explanation:
Depending on the input values, the code can output "Enter the cave", "Enter with caution", "Wait", "Wait for sunrise", or "Locked out". "Invalid time" (f) would only occur if time_of_day is neither "day" nor "night".

Question 19

The Enchanted Valley 🌄 ✨
The wizard is exploring a 6x6 grid valley. Each cell contains an enchanted flower (1), a cursed flower (-1), or an empty patch (0). The counting rules are:

1. An enchanted flower is valid if no adjacent (including diagonal) cells contain a cursed flower.
2. **Corner Rule:** A valid enchanted flower in a corner counts as twice.
3. **Boundary Rule:** A valid enchanted flower on the boundary (but not a corner) counts as 1.5 times (round down after summing).

The following code calculates the count:

```
garden = [
   [1, 0, -1, 1, 0, -1],
   [0, 1, 0, 1, -1, 0],
   [1, 0, 1, 0, 0, 1],
   [-1, 1, 0, -1, 1, 0],
   [1, 0, 0, 1, 0, 1],
   [-1, 1, 0, 0, 1, -1]
]

valid_flower_count = 0

directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]
```

```
for i in range(len(garden)):
    for j in range(len(garden[i])):
        if garden[i][j] == 1:
            is_cursed_nearby = False
            # Check adjacent cells
            for dx, dy in directions:
                ni, nj = i + dx, j + dy
                if 0 <= ni < len(garden) and 0 <= nj < len(garden[i]) and garden[ni][nj] == -1:
                    is_cursed_nearby = True
                    break
            # Add to count if valid
            if not is_cursed_nearby:
                if (i == 0 and j == 0) or (i == 0 and j == len(garden[i]) - 1) or (i == len(garden) - 1 and j == 0) or (i == len(garden) - 1 and j == len(garden[i]) - 1):
                    valid_flower_count += 2  # Corner flower
                elif i == 0 or i == len(garden) - 1 or j == 0 or j == len(garden[i]) - 1:
                    valid_flower_count += 1.5  # Boundary flower
                else:
                    valid_flower_count += 1  # Regular valid flower


print(int(valid_flower_count))
```

What is the total count of valid enchanted flowers after applying the rules?

Answer: 2

Explanation:
The code checks each cell and applies the scoring rules: adding 2 for a corner flower, 1.5 for a boundary flower, and 1 for an inner flower. After processing and converting the total to an integer (rounding down), the final count is 2.

Question 20

Magical Forest Mystery: Skipping the Bad
In a mystical forest, creatures are represented by the strings "good" and "bad". If

a "bad" creature is encountered, its immediate children are ignored when counting "good" creatures; however, grandchildren and deeper descendants are still considered. Consider the following code:

```
def count_good_creatures(tree):
    if isinstance(tree, str):
        return 1 if tree == "good" else 0
    total = 0
    for subtree in tree:
        if isinstance(subtree, list):  # If the subtree is a list
            if "bad" in subtree:
                # Skip immediate children of "bad"
                total += count_good_creatures([s for s in subtree if s != "bad" and isinstance(s, str)])
            else:
                total += count_good_creatures(subtree)  # Process the list recursively
        else:
            total += count_good_creatures(subtree)  # Handle individual string elements
    return total


forest = [
    "good",
    ["bad", "good", ["good", "bad", ["good", "good"]]],
    ["good", ["bad", "good"]],
    "good"
]


print(count_good_creatures(forest))
```

What is the output of the code?

Options:

- A) 6
- B) 7
- C) 8

- D) 5

Answer: D) 5

Explanation:
The code recursively counts "good" creatures while skipping the immediate children of any "bad" occurrence. After processing the nested structure, the final output is 5.

Question 21

Delivery Route Optimizer

You work as a software engineer for a logistics company. The company wants to optimize delivery routes by sorting deliveries based on the following priorities:

1. **Priority** (highest first)
2. **Distance from depot** (closest first)
3. **Delivery Window** (earliest start time first)

Each delivery is represented as a tuple:
(delivery_id, priority, distance_from_depot, delivery_window_start_time, delivery_window_end_time)

Below is the code for sorting the deliveries:

```
def sort_deliveries(deliveries):
    # Custom sort function using priority, distance, and delivery window start time
    return sorted(deliveries, key=lambda x: (-x[1], x[2], x[3]))


# Sample deliveries list
deliveries = [
    (101, 3, 20, 9, 11),
    (102, 1, 50, 10, 12),
    (103, 2, 30, 8, 10),
    (104, 3, 10, 7, 9)
]


# Sorting the deliveries
sorted_deliveries = sort_deliveries(deliveries)
```

Displaying the sorted deliveries

print(sorted_deliveries)

After applying the sorting algorithm, what is the sorted order of deliveries?

Options:

- **A.**
  i. (104, 3, 10, 7, 9)
  ii. (101, 3, 20, 9, 11)
  iii. (103, 2, 30, 8, 10)
  iv. (102, 1, 50, 10, 12)
- **B.**
  i. (104, 3, 10, 11, 9)
  ii. (101, 3, 20, 9, 11)
  iii. (103, 2, 30, 8, 10)
  iv. (102, 1, 50, 10, 12)
- **C.**
  i. (103, 2, 30, 8, 10)
  ii. (102, 1, 50, 10, 12)
  iii. (101, 3, 20, 9, 11)
  iv. (104, 3, 10, 7, 9)
- **D.**
  i. (102, 1, 50, 10, 12)
  ii. (101, 3, 20, 9, 11)
  iii. (103, 2, 30, 8, 10)
  iv. (104, 3, 10, 7, 9)

Answer: Option A

Explanation:

- **Priority:** Deliveries (104, 3, 10, 7, 9) and (101, 3, 20, 9, 11) have the highest priority (3).
- **Distance:** Among the highest priority deliveries, (104, 3, 10, 7, 9) has a shorter distance (10) compared to (101, 3, 20, 9, 11) (distance 20), so it comes first.
- The remaining deliveries are sorted next by their priority values.

Question 22

You're explaining sorting algorithms to your friend. In selection sort, after the first pass, which element is guaranteed to be in its correct position?

Here, we wanted to arrange the elements in ascending order.

Options:

- A) The largest element
- B) The smallest element
- C) The middle element
- D) No element is guaranteed to be in its correct position

Correct Answer:

Option B) The smallest element

Explanation:

In selection sort, during each pass, the algorithm selects the smallest element from the unsorted portion of the array and places it at the beginning of the sorted portion. After the first pass, the smallest element in the entire array is guaranteed to be in its correct position at the start of the array. This process continues with each subsequent pass, gradually building the sorted portion of the array from left to right.

Question 23

Reversing a List
A programmer writes a function to reverse a list of n integers using a loop:

```
def reverse_list(lst):
    reversed_lst = []
    for i in range(len(lst) - 1, -1, -1):
        reversed_lst.append(lst[i])
    return reversed_lst
```

What is the time and space complexity of this function?

Options:

- A) Time: O(n), Space: O(1)
- B) Time: O(n), Space: O(n)
- C) Time: O(n^2), Space: O(n)
- D) Time: O(log n), Space: O(log n)

Answer: Option B

Explanation:
The function iterates through the list once (O(n) time) and creates a new list to store the reversed elements, resulting in O(n) space.

Question 24

Navan has written the following function divisibleUpto(n, m) to check divisibility and store divisors:

```
def divisibleUpto(n, m):
  l = []
  for i in range(n, 0, -1):
    if m % i == 0:
      l = [i] + l
  return l
```

Here, n and m are positive integers. For which of the following values (which equals the sum n + m) will the function return a list of size at least 4?

Options:

- A) 4
- B) 10
- C) 7
- D) 15

Answer: Option D

Explanation:
The function divisibleUpto(n, m) constructs a list of divisors of m from n down to 1, prepending each divisor to the list. The goal is to determine for which option (A, B, C, D) the sum n + m results in a list of size ≥4.

Analysis:

- **Option A (4):** Possible pairs (n, m) are (1,3), (2,2), (3,1). All result in lists of size ≤2. ❌
- **Option B (10):** All pairs (e.g., n=6, m=4) yield lists with ≤3 divisors. ❌
- **Option C (7):** Pairs (e.g., n=3, m=4) produce lists with ≤2 divisors. ❌

- **Option D (15):** The pair (n=9, m=6) works. Divisors of 6 (1, 2, 3, 6) all ≤9. The list becomes [1, 2, 3, 6], size 4. ✅

Question 25: Typecasting Error

You are a Python developer working on a data processing project. You need to perform various typecasting operations on input data. Which of the following operations will produce an error?

1. str(3.145)
2. float("13.44")
3. int("7.4")
4. bool(0)

Correct Answer:

Option 3

Explanation:

In this scenario, option 3 (int("7.4")) will produce an error. The int() function cannot directly convert strings with decimal points to integers. It expects a string representation of an integer, not a float. The other operations are valid:

- str(3.145) converts a float to its string representation.
- float("13.44") successfully converts a string representation of a decimal number to a float.
- bool(0) converts zero to False, which is a valid boolean operation.

Question 26: Function Output

As a Python tutor, you're reviewing a student's code. What will be the output of the following function?

```
def  wrong(x):
  return  8**2/2


print(wrong(4))
```

Options:

1. 8
2. Error
3. 32

4. 33

Correct Answer:

Option 3) 32

Explanation:

The function wrong(x) computes (8**2)/2, which equals 64/2 = 32. Note that:

- The parameter x is not used in the function body.
- In Python 3, the / operator performs floating-point division, but the result is still 32.0, which is printed as 32.

Question 27: Bubble Sort Characteristic

You're explaining sorting algorithms to a computer science class. In bubble sort, after the first pass, which element is guaranteed to be in its correct position?

Here, we are sorting the elements in ascending order.

Options:

- A) The smallest element
- B) No element is guaranteed to be in its correct position
- C) The middle element
- D) The largest element

Correct Answer:

Option D) The largest element

Explanation:

In bubble sort, after the first complete pass through the array, the largest element is guaranteed to "bubble up" to the last position. This is because in each comparison, the larger element is moved towards the end of the array. After one full pass:

- The largest element will have been compared with every other element.
- It will have been swapped to the right in each comparison where it was larger.
- Thus, it will end up at the rightmost position, which is its correct final position.

Question 28: Valid Function Definition

You're creating a language translation function in Python. Which of the following is a valid function definition with a default parameter?

Options:

- A) def translate(lang, text = "I like this"):
- B) def translate(text = "I like this", lang):

- C) def translate(lang, "I like this" = text):
- D) def translate("I like this" == text, lang):

Correct Answer:

Option A) def translate(lang, text = "I like this"):

Explanation:

This function definition is correct because:

- It uses the correct syntax for assigning a default value to a parameter (text = "I like this").
- The parameter with a default value (text) comes after the parameter without a default (lang).
- In Python, parameters with default values must be placed after parameters without default values in the function definition.

Question 29: Nested Function Output

You're working with nested functions in Python. What will be the output of print(cool(5)) after running the following code?

```
def  ander(x):
  z =  1
  def  bahar(y):
            return  (x ** y)
  return bahar


cool = ander(2)
```

Options:

- A) 32
- B) 10
- C) 25
- D) Error

Correct Answer:

Option A) 32

Explanation:

This code demonstrates a closure in Python:

1. The ander() function returns the inner function bahar().
2. bahar() remembers the value of x from its enclosing scope.
3. When we call ander(2), it returns bahar with x=2.
4. cool now refers to bahar with x=2.
5. Calling cool(5) is equivalent to bahar(5) with x=2, which returns 2**5 = 32.

This showcases how nested functions can "remember" values from their outer scope, even after the outer function has finished executing.

Question 30

You are programming a drone delivery system to locate items in different storage zones of a warehouse.

- **Zone A**: Items are randomly scattered (unsorted).
- **Zone B**: Items are sorted by weight in ascending order.

The drone's battery consumption depends on the number of comparisons it makes during a search. The drone needs to find an item with **ID P567**:

- Zone A has 500 items.
- Zone B has 1024 items.
- Each comparison in **Zone A** costs 2 units of battery, and each comparison in **Zone B** costs 1 unit of battery.

What is the total battery consumption in the **worst case** if the drone uses linear search in Zone A and binary search in Zone B?

Correct Answer: 1010 units