COL1000: Introduction to Programming **Set A**

Major Exam
Date: 21/11/2025

Time: 120 minutes **Pages: 10** **Marks: 105**

Please carefully read the instructions below before attempting the exam:

- Write your name and entry number on each sheet.
- Rough sheet is provided in the back. Extra sheets are available. However, you must write your answers in the provided boxes. Answers outside boxes will be remain ungraded. If your writing is not legible to two graders, you will get 0. You will not be asked to explain your writing.
- No clarifications will be given. If you think a question is unclear, succintly write your assumption and then solve the question under your stated assumption.

Name: Entry No.:

MAP EACH QUESTION AND ITS GIVEN ANSWER TO THE CORRESPONDING QUESTION IN YOUR SET. LOOK OUT FOR A SLIGHT CHANGE OF VALUES IN SOME QUESTIONS.

Question 1 (3 marks) Which are the local variables, respectively, of functions A and B in the following code. Mark the local variables by writing X in the box before it on the right.

```
def A(D):
    C = D
    def B(E):
        global C
        C = D
    return B
```

A ☐ A ☐ B ☐ C ☐ D ☐ E

B ☐ A ☐ B ☐ C ☐ D ☐ E

Answer: A) B,C,D
B) D,E

Question 2 (3 marks) What is the output of the following code sequence? How many times is the function g called?

```
def f(x=3):
    def g(ff=f):
        nonlocal x
        x += ff(x-1)
    if x: g()
    return x
print(f())
```

Output:

Number of calls:

Answer: 6, 3

Question 3(10 marks) For the two code sequences given below, answer the following:

i. How many different unique objects are named `x` during the execution of each of the following code sequences. (You may assume that Python maintains only one copy of an int with a given value. Assume Iterable has been already imported from typing in each case).

ii. In each case, also list the maximum number of instances of the variable x which exist in the system (across different scopes) at the end of any statement during the execution.

iii. Circle each different variable x. Also write next to each circle, the number of different instances of x created at that point. (Note that not all occurrence of x creates a new reference. Only circle those that do.)

iv. Print the value of x at the end of the code sequence. If x is undefined at the end, write 'NameError'.

```
1 def f(x:tuple):
2     if len(x) == 0: return 0
3     return x[0] + f(x[1:])
4 x = f((1, 2, 3, 4))
```

i.

ii.

iv

Answer: 6,6,[L1*5,L4],10

```
1 def f(x:Iterable) -> int:
2     sum = 0
3     for y in x: sum += y
4     return y
5 x = 44
6 f([x])
7 print(f((x,)))
```

i.

ii.

iv

Answer: 3,2,[L1,L5],44

Question 4 (6 marks) What does the following code produce?

```
1 x = 11
2 def f():
3     global x
4     x = 100
5     print(x)
6 f()
7 print(x)
```

Answer:
100
100

```
def add1(L: list):
    for e in L:
        if type(e) == list:
            add1(e)
        elif type(e) == int:
            e += 1
L1 = [1,2]
L2 = [L1, L1, (1,2), 1, 2]
print(add1(L2))
print(L2)
```

Answer:
None
[[1, 2], [1, 2], (1, 2), 1, 2]

```
x = 10
def f():
    x = 11
    print(x)
f()
print(x)
```

Answer:
11
10

Question 5 (3 marks) The following code attempts to filter-in integer and float values in list L but fails. Identify the error and correct it by modifying a single line. What does it produce after the correction?

```
def filter_in(L: list) -> filter:
    return filter(lambda x: x is int or float , L)
print(list(filter_in([1, "1", 1.0, "one"])))
```

Modified line (Also mention line no. _____):

Prints:

Answer: Compares value to type, "or float" makes it always True
type(x) is int or type(x) is float

Question 6 (3 marks) The following code attempts to use the function f to add to g, but fails. What does it produce instead? Point out the error in g and fix it by adding a single line. What does it produce after the correction?

```
g = 100
def f(x):
    g += x
    return g
print(g, f(g))
```

Original Output:

Error:

Fix:

Answer: g is modified inside f and local by default. This causes an UnboundLocalError. Add 'global g; to the function f. It produces:
100, 200

Question 7 (5 marks) The following function reads a file and returns a list. Write an iterative algorithm to achieve the same result using readline() function. (Exact Python syntax not required.)

```
def filelist(filename:str)->list[str]:
    with open(filename, "r") as f:
      return list(filter(lambda x:x!="", map(lambda x:x.strip(), f.
                                                    readlines())))
```

```
def filelist(filename:str)-$>$list[str]:
    L = []
    with open(filename, "r") as f:
         while True:
            line = f.readline()
            if line == "": break
            line = line.strip()
            if line == "": continue
            L += [line]
    return L
```

Question 8 (3 marks) A program occasionally crashes with TypeError or IndexError in the following function, which is designed to add elements in a given part of list L. How would you discover the source of those errors by making changes in this function. (Note that f could be called from many places in the program, which may not be possible to determine in advance.)

```
def f(L, b, e):
    sum = 0
    for i in range(b,e):
        sum += L[i]
    return sum
```

TypeError:

IndexError:

Answer: Typerror is best detected by using typing and a static type-checker. asserting on type is also ok. Type of L:list[int—float], b:int, e:int. Indexerror could be is i is outside the range of L. assert that b and e are in range -len(L), len(L)-1 or b $\geq$ e.

Question 9 (2 marks) Which of the following statements about exceptions in Python are correct? Fill the box for True statement(s) with X. If any False statement is marked, you will lose all marks.

A. Exceptions interrupt the normal flow of execution ☐

B. All exceptions must be caught, or the program will crash ☐

C. A single try block can have multiple except clauses ☐

D. The order of except clauses does not matter; Python always picks the most specific matching exception type automatically. ☐

E. The finally block executes whether or not an exception occurs ☐

Answer: A,B,C,E

Question 10 (2 marks) Which statement(s) about a precondition in a function specification is (are) true? Fill the box for True statement(s) with X. If any False statement is marked, you will lose all marks.

A. The statement must hold after the function returns ☐

B. The statement must hold before the function is called ☐

C. It is a statement that describes the function's time complexity ☐

D. It is a statement that documents the exceptions the function may raise ☐

E. Preconditions are obligations on the calling function ☐

Answer: B, D

Question 11 Inspect the code below to determine its intent and answer the following questions.

```
1 def StrOperation(s: Any) -> str | None:
2     for ch in s:
3         if ch.isupper():
4             return ch
```

A. **(2 Marks)** Specify the expected post condition of the program.

B. **(4 Marks)** List two exceptions and give one example input that triggers each.

(a) Exception: ☐ Input: ☐

(b) Exception: ☐ Input: ☐

Answer:

A. The function returns the first character in the input string, which is uppercase or None otherwise .

B. (i) TypeError – if the input is not an iterable object; (ii) Attribute error if the input is indeed iterable but does not have the isUpper() attribute;

Question 12 Given the following logical specification, answer the below questions. Note that `index_of(x, lst)` returns an integer representing an index of `x` in `lst`. Similarly, `first_index_of(x,lst)` returns an integer representing the position of the first occurrence of `x` in `lst`. If no valid index exists, assume it to be -1.

- Input: *lst* is a list of comparable elements (can be empty, can also have multiple instances of elements)
- Output: *result* is a list such that:

A. **(2 Marks)** for all $x, y \in result$: if $(index\_of(x, result) < index\_of(y, result))$, then $(first\_index\_of(x, lst) < first\_index\_of(y, lst))$. Consider $lst = [5, 2, 8, 2, 3]$. Which of the following is a valid result based on the above specification and explain your answer :
(i) $[5, 2, 8, 3]$, (ii) $[2, 5, 8, 3]$

Valid: ☐ Explain: ☐

Answer: (i) is valid , in (ii) the ordering is not preserved

B. **(2 Marks)** for all $i, j \in [0, len(result))$: If $i \neq j$, then $result[i] \neq result[j]$.

i. If $lst = [5, 5, 5]$, what is the maximum length *result* can have while satisfying this specification? Explain your answer.

☐

Answer: The size of `result` could be infinite. But assuming that result is drawn from the `lst` then the length is 1 (1 Mark). Unique elements in `result` are preserved.

C. **(2 Marks)** for every distinct element $e \in lst$, e is also in *result*. Given $lst = [7, 2, 9, 2, 5]$, which of these results do not violate this specification? (Fill x in the appropriate boxes.)

i. $[7, 2, 9, 5]$ ☐
ii. $[2, 9, 5]$ ☐
iii. $[7, 2, 9, 5, 2]$ ☐
iv. $[7, 2, 9, 5, 11]$ ☐

Answer: i, iii, and iv..

D. **(2 Marks)** for all $x \in result$: $index\_of(x, result)$ corresponds to $first\_index\_of(x, lst)$. Consider $lst = ['x', 'y', 'x', 'z']$. If $result[1] = 'x'$, then does it satisfy the specification? Explain.

☐

Answer: If $result[1] = 'x'$: Index of $'x'$ in $result = 1$, but $first\_index('x', lst) = 0$, so this is in violation of the spec. Otherwise the specification may hold if $result = \{'x', 'x'\}$

Question 13 (3 marks) What will be the output of the following program:

```
def f(n):
    if n == 0:
        print(0, end=" ")
        return
    f(n-1)
    print(n-1, end=" ")
f(3)
```

Output:

Answer: 0 0 1 2

Question 14 (4 marks) Consider the following function:

```
def s(n):
    if n == 0:
        return 1
    return n * s(n-1)
```

- Is the function $s(n)$ tail-recursive?
- If yes, justify your answer. If not, write $s2$, a tail-recursive version of $s(n)$. $s2(n)$ must produce the same result as $s(n)$ for every n.

If yes:

If no:

Answers (i) No (ii)

```
def s2(n, acc=1):
    if n == 0:
        return acc
    return s2(n-1, acc * n)
```

Question 15 (3 marks) Rewrite the following function using recursion instead of while loop. Your recursive function must produce exactly the same output for every input integer x, and must not use loops.

```
def compute(x):
    result = 1
    while x > 0:
        if x % 2 == 0:
            result = result + x
        else:
            result = result * x
        x = x - 1
    return result
```

Answer

```
def compute_rec(x, result=1):
    if x <= 0:
        return result
    if x % 2 == 0:
        return compute_rec(x-1, result + x)
    else:
        return compute_rec(x-1, result * x)
```

Question 16 (5 marks) In an tall apartment complex, someone left their water on and it's flooding all units below it. Unfortunately, all apartment doors are locked, and the only way to find (and fix) the leak is to break the lock and check inside. Provide an algorithm to find the leak without breaking up to a constant fraction of all the locks in the worst case. List how you will decide which lock to break next and when to stop. (Exact python is not required, but steps must be clear.)

Next floor = Middle of the remaining floors
Break lock of Next floor and check
If middle apartment is the source of the leak: Repair and complete
Otherwise, if floor is dry: Recurse on the lower half
Otherwise, recurse on the upper half
(Definitive condition for source of the leak: this floor is wet and the one above is dry.)

Question 17 (10 marks) Write an efficient algorithm to find all elements in a list of integers that are unique. (Steps of your algorithm should be like python statements, but need not be syntactically precise. You may directly refer to any code or algorithm that appears elsewhere in this test.) Provide its run-time analysis. (You are expected to devise an algorithm better than one that is proportional to n^2.) **Rubric: recognize sort: 2 + iteration: .5, check same: .5, actions if same or not: 1 each. Should mention about how to report the unique elements found. O(n^2) but correct == 2.5 mark.**

Both interpretations allowed: filter out singletons, or filter out a single copy of each
Mergesort(L), where L is the input list
unique = []
for i in range(len(L)):
· if isunique(L[i]): unique.append(e), where
isunique(L) is (i==0 or L[i-1] != L[i]) and (i==len(L)-1 or L[i] != L[i+1])
Correctness: Sorting ensures that all elements with the same value are contiguous. A value is unique if it is not preceded nor succeeded by a copy.

Run-time analysis:
Merge sort is proportional to $n \log n$ (see analysis in class notes, and the Mergesort question below.)
The loop above has n (== length of L) iterations. Each statement outside the loop (and after the sort) takes bounded time independent of n. If the statements inside the loop is assumed to take bounded time each, the total number of steps in the loop is proportional to n. Arithmetic, Relational, and Logical operations as well as Indexing all may be assumed to take bounded time. Caveat: Append is a doubtful case, and could take variable time dependent on the size of unique. However with the assumption that it takes bounded time, the total time would be bounded by $kn \log n$ for some n

Question 18 Merge Sort is a classic divide-and-conquer algorithm that sorts a list by recursively dividing it into halves, sorting each half, and merging the sorted results. The algorithm uses two functions—`mergeSort` (which performs the recursive splitting and sorting) and `merge` (which combines two sorted lists). Their specifications are provided below.

Function Specifications

```
def merge(left: list[int], right: list[int]) -> list[int]:
    """Merges two sorted lists into one sorted list.

    Preconditions:
    - Both 'left' and 'right' are sorted in ascending order.

    Postconditions:
    - Returns a new list containing all elements from 'left' and 'right',
      sorted in non-decreasing order.
    - The length of the returned list is len(left) + len(right)."""
```

```
def mergeSort(arr: list[int]) -> list[int]:

    """
     Sorts a list of integers using the merge sort algorithm.

    Preconditions:
     - arr is a list of integers (possibly empty).

```

```
    Postconditions:
    - Returns a new list that contains the same integers as 'arr',
       sorted in ascending order.
    - The input list 'arr' should not be modified.
    """

  # TODO: Implement this function
```

- **Base Case:** If `arr` contains 0 or 1 element, return `arr`.
- **Recursive Case:** If `arr` contains 2 or more elements:
  1. Split `arr` into `leftHalf` and `rightHalf`.
  2. First call `mergeSort(leftHalf)`, then call `mergeSort(rightHalf)`.
  3. After both recursive calls return sorted lists, merge them using `merge(left, right)` and return the merged result.

A. **(5 marks) Implementation Task**: Complete the body of `mergeSort` according to the specifications above (space provided above). You will not lose marks for incorrect syntax. However, try to stay as close as possible to Python.

B. Given the input list, XX = $[8, 3, 2, 9, 7, 1, 5, 4]$, answer the following questions.

i) **(2 marks)** How many total calls to `mergeSort` are made by `mergeSort`(XX) on this list?

Include the initial call and all recursive calls in your count.

ii) **(2 marks)** How many calls to `merge` are made?

iii) **(5 marks)** List all `mergeSort(arr)` calls in the order they are invoked. Write each call with the exact list it receives as input. Assign each call an invocation rank (1 = first called). For example, if the first call is on sublist $[3, 2, 9]$, you should write "`1: [3,2,9]`." You may list them on multiple lines, each line will be read left to right.

iv) **(3 marks)** For the same calls, assign a completion rank indicating the order in which each call returns its result to its caller (1 = first to complete). For example, if it returns first on sublist $[3, 2, 9]$, then write "`1:  [3,2,9]`". Write left to right and then next line.

C. **(3 marks)** If $len(arr) = 257$, what is the maximum number of simultaneously pending (i.e., not yet completed) `mergeSort` calls at any moment during execution?

Solution for Merge Sort Assessment Q18

1. **Total number of `mergeSort` calls** (2 points)
   - Correct Answer: 15
2. **Total number of `merge` calls** (2 points)
   - Correct answer 7
3. **Invocation order of all `mergeSort` calls**

   Solution:

   1. $[8, 3, 2, 9, 7, 1, 5, 4]$
   2. $[8, 3, 2, 9]$
   3. $[8, 3]$
   4. $[8]$
   5. $[3]$
   6. $[2, 9]$
   7. $[2]$
   8. $[9]$
   9. $[7, 1, 5, 4]$
   10. $[7, 1]$
   11. $[7]$
   12. $[1]$
   13. $[5, 4]$
   14. $[5]$
   15. $[4]$
4. **Completion order of all `mergeSort` calls**

Solution:

1. [8]
2. [3]
3. [8, 3]
4. [2]
5. [9]
6. [2, 9]
7. [8, 3, 2, 9]
8. [7]
9. [1]
10. [7, 1]
11. [5]
12. [4]
13. [5, 4]
14. [7, 1, 5, 4]
15. [8, 3, 2, 9, 7, 1, 5, 4]

5. **Maximum number of pending `mergeSort` calls for $n = 257$**

   - Correct answer: 10

6. **Python implementation of `merge` and `mergeSort`**

   - See class notes

Question 19 You are the manager of a company that offers three services: **plumbing**, **cleaning**, and **carpentry**.

Worker	Plumbing	Cleaning	Carpentry
A	100	10	10
B	50	80	7
C	45	55	90

Table 1: Expertise scores for each worker–job pair

You have three workers—A, B, and C—each capable of performing all three jobs, but with different levels of expertise. The expertise scores (higher is better) for each worker–job pair are given in Table 1.

You must assign exactly one worker to each job. Each worker may be assigned to **at most one** job. The quality of an assignment plan is defined as the sum of the expertise scores of the worker–job pairs in the plan. For example, the plan

$$\{\langle A, \text{Cleaning}\rangle,\ \langle B, \text{Plumbing}\rangle,\ \langle C, \text{Carpentry}\rangle\}$$

has quality $10+50+90 = 150$. The goal is to find the assignment plan with the **maximum total quality**. The company uses the following greedy job assignment algorithm:

- While there exists an unassigned job:
  (a) Select the available worker–job pair $\langle w, j\rangle$ with the **highest expertise score**.
  (b) Assign job j to worker w.
  (c) Mark worker w as unavailable.

(d) Mark job j as assigned.

Now answer the following questions:

A. **(3 marks)** List the **sequence** of worker–job assignments the given greedy algorithm produces.

B. **(3 marks)** Provide a 3×3 expertise matrix (with non-negative entries) showing a **counterexample** where the greedy algorithm fails to find the optimal plan, i.e., **maximum total expertise**.

Worker	Plumbing	Cleaning	Carpentry
A			
B			
C			

1 Solution for Greedy Worker-Job assignment Q19

1. **Algorithm output:** Solution- A to Plumbing, C to Carpentry, B to Cleaning.
2. **Proof of optimality.** Greedy will not always be correct. Any counter-example gets full points. One counter-example provided below. Greedy will produce A-Plumbing, B-Cleaning, C-carpentry given

Worker	Plumbing	Cleaning	Carpentry
A	100	99	99
B	99	2	0
C	99	0	1

total expertise of 103. Instead, A-Cleaning, B-Plumbing, C-Carpentry gives 199.