Quiz 02 - Practice

COMP 110: Introduction to Programming
Spring 2024

Thursday March 7, 2024

Name: ____

9-digit PID: ____

Do not begin until given permission.

Honor Code: I have neither given nor received any unauthorized aid on this quiz.

Signed: ____

Question 1: Multiple Choice Completely fill in the bubble next to your answer using a pencil. Each question should have exactly one filled-in bubble.

1.1. A variable's value should not be reassigned after initialization.

○ `True`
● `False`

1.2. The first time a variable is bound to a value is referred to with which of the following special names?

○ Assignment
● Initialization
○ Relative Reassignment
○ Declaration

1.3. Which side of the following statement should be evaluated first?

```
x = y
```

○ `x`
● `y`

1.4. The following two statements are equivalent to one another and interchangeable:

```
x = y
y = x
```

● `False`
○ `True`

1.5. The following statement increments `x`'s value by `1`.

```
x + 1 = x
```

● `False`
○ `True`

1.6. The following statement increments `x`'s value by `1`.

```
x += 1
```

○ `False`
● `True`

1.7. Tuples and lists can both be mutated after creation.

● `False`
○ `True`

1.8. When accessing an index of a list that does not exist, what kind of error is encountered?

○ `NameError`
○ `KeyError`
● `IndexError`
○ `StackOverflowError`

1.9. When *accessing* an element of a list, what kind of value most generically describes what is found inside the subscription notation's square brackets. E.g. `a_list[HERE]`

○ Integer Literal
○ Data Type
● Integer Expression
○ Integer Variable Name

1.10. Generally, to avoid an infinite `while` loop, each iteration of the loop body should change a variable involved in the `while` loop's test condition bringing it closer to `False`:

○ False
● True

1.11. Consider a function named `f` with a `while` loop. In the `while` loop's body, there is a `return` statement. At most, how many times will this `return` statement be evaluated in a single function call to `f`?

● 1
○ As many times as the loop iterates
○ Infinite

1.12. Which of the following describes a test written to demonstrate an expected usage of a function?

○ Edge Case
● Use Case

1.13. What is the evaluation of the following expression:

```
[10, 20, 30][[0, 1, 2][3 - 1]]
```

○ 10
○ 20
● 30
○ IndexError

Question 2: Respond to the following questions

Consider the following code listing:

```
animals: list[str] = ["fox", "bear", "rabbit"]
ints: list[~~str~~ int] = [1, 1, 1, 1]
two_d: list[list[int]] = [[10, 20], [30, 40], [50, 60]]
```

2.1. Write an expression that evaluates to `"bear"`, making use of the `animals` variable.

Solution: `animals[1]`

2.2. Write a method call that adds the value `"mouse"` to the `animals` list.

Solution: `animals.append("mouse")`

2.3. Write a function call expression that evaluates to the quantity of values in the `animal` list.

Solution: `len(animals)`

2.4. Write an expression that increments the 3rd value in `ints` to be one greater than its previous value (regardless of what the previous value was).

Solution: `ints[2] = ints[2] + 1` or `ints[2] += 1`

2.5. Write a sequence of 3 assignment statements that will swap the values of the 0 and 1 index in `animals`. You will need to declare and initialize a temporary variable.

Solution:

```
temp: ~~int~~ str = animals[1]
animals[1] = animals[0]
animals[0] = temp
```

2.6. Write an expression that accesses the value `40` stored in the `two_d` variable.

Solution: `two_d[1][1]`

2.7. Write an expression that accesses the list `[50, 60]` stored in the `two_d` variable.

Solution: `two_d[2]`

2.8. Write an expression that removes the item at index 1 from `animals`.

Solution: `animals.pop(1)`

Question 3: Memory Diagram Trace a memory diagram of the following code listing.

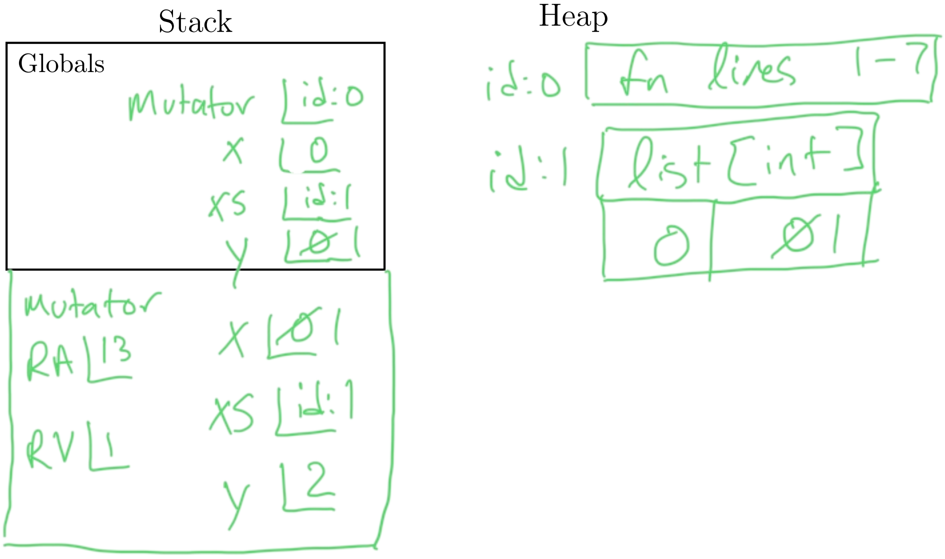
```
1  def mutator(x: int, xs: list[int]) -> None:
2    """An impure function..."""
3    x += 1
4    xs[0] += 1
5    y: int = x + 1
6    print(f"mutator x: {x}, xs: {xs}, y: {y}")
7    return x
8
9  x: int = 0
10 xs: list[int] = [0]
11 y: int = 0
12 print(f"global before x: {x}, xs: {xs}, y: {y}")
13 y = mutator(x, xs)
14 print(f"global after x: {x}, xs: {xs}, y: {y}")
```

Output

Solution: global before x: 0, xs: [0], y: 0

mutator x: 1, xs: [1], y: 2

global after x: 0, xs: [1], y: 1

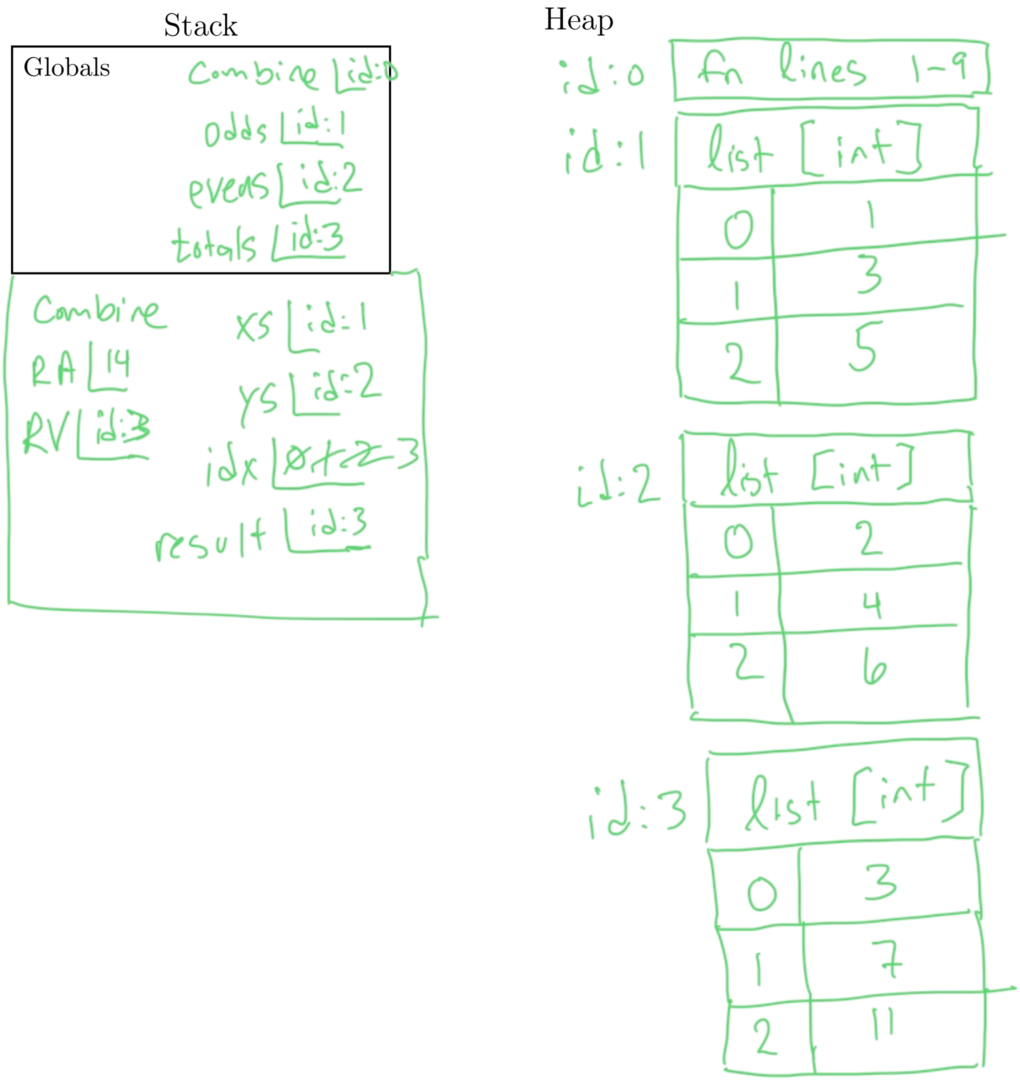


Question 4: Memory Diagram Trace a memory diagram of the following code listing.

```
1  def combine(xs: list[int], ys: list[int]) -> list[int]:
2    """Add the items of two lists item-wise."""
3    assert len(xs) == len(ys)
4    idx: int = 0
5    result: list[int] = []
6    while idx < len(xs):
7      result.append(xs[idx] + ys[idx])
8      idx += 1
9    return result
10
11
12 odds: list[int] = [1, 3, 5]
13 evens: list[int] = [2, 4, 6]
14 totals: list[int] = combine(odds, evens)
15 print(totals)
```

Output

Solution: [3, 7, 11]



Question 5: Memory Diagram Trace a memory diagram of the following code listing.

```
1  def sort(xs: list[int]) -> None:
2    """Sort with the insertion sort algorithm."""
3    N: int = len(xs) # Number of items
4    i: int = 1       # "current index"
5    x: int           # "current value"
6    si: int          # "shift index" searching backward
7
8    while i < N:
9      print(xs)
10     x = xs[i]      # store current value
11     si = i
12     while si > 0 and x < xs[si - 1]:
13       xs[si] = xs[si - 1] # shift greater value forward one
14       si -= 1
15     xs[si] = x      # *insert* (assign) "current value" in correct position
16     i += 1
17
18
19 values: list[int] = [40, 10, 30, 20]
20 sort(values)
21 print(values)
```

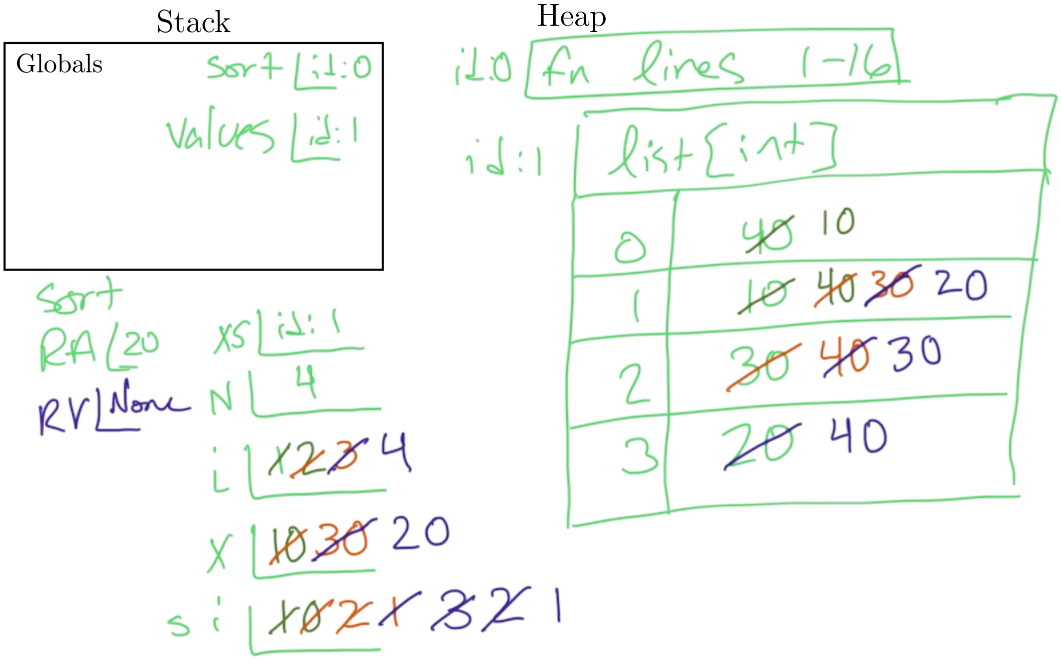
Output

Solution: [40, 10, 30, 20]

[10, 40, 30, 20]

[10, 30, 40, 20]

[10, 20, 30, 40]

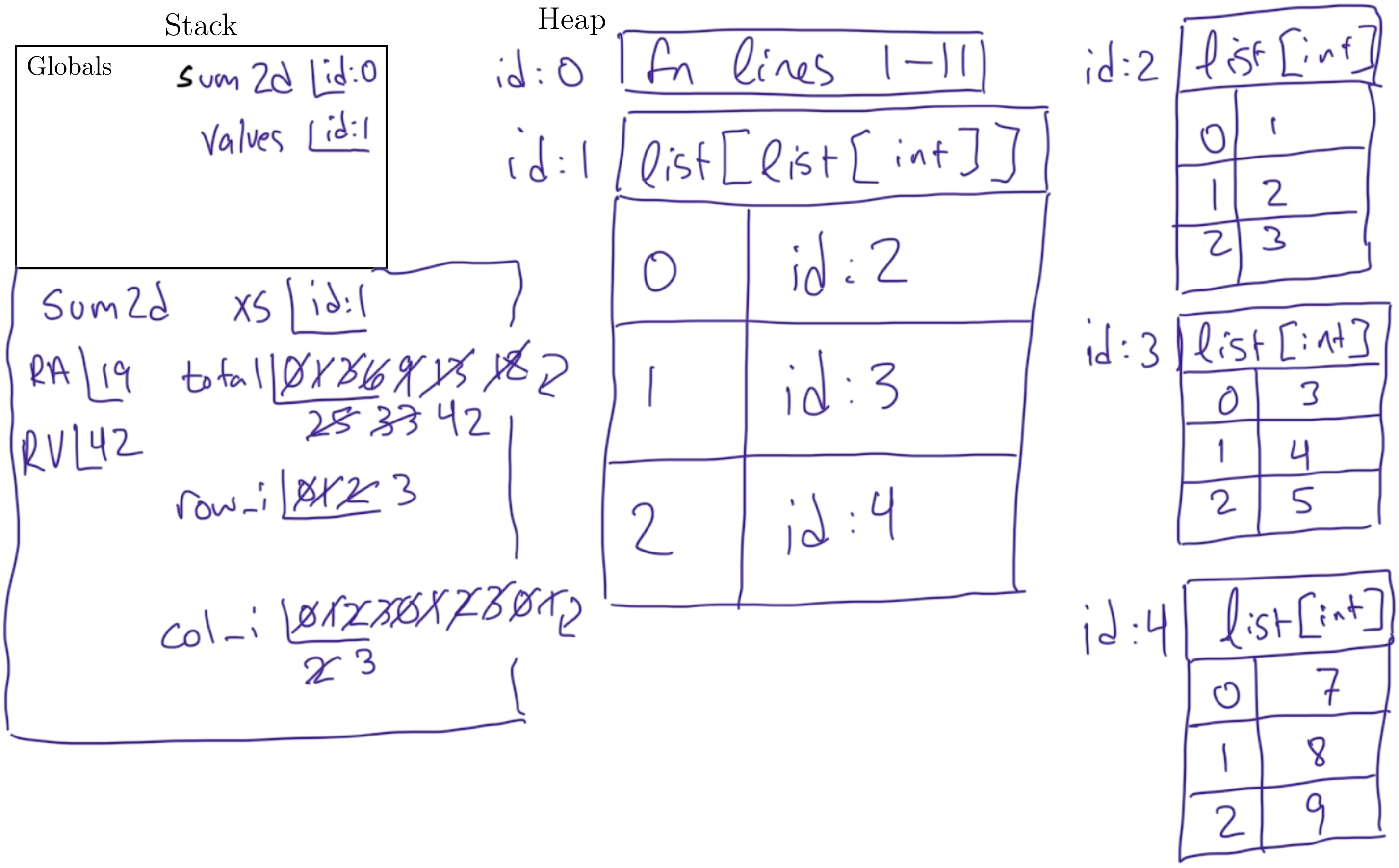


Question 6: Memory Diagram Trace a memory diagram of the following code listing and then answer the sub-questions. You do not need to diagram the sub-questions.

```
1  def sum2d(xs: list[list[int]]) -> int:
2    """Calculate the sum of a 2-dimensional list of lists."""
3    total: int = 0
4    row_i: int = 0
5    while row_i < len(xs):
6      col_i: int = 0
7      while col_i < len(xs[row_i]):
8        total += xs[row_i][col_i]
9        col_i += 1
10     row_i += 1
11   return total
12
13
14 values: list[list[int]] = [
15   [1, 2, 3],
16   [3, 4, 5],
17   [7, 8, 9]
18 ]
19 print(sum2d(values))
```

Output

Solution: 42

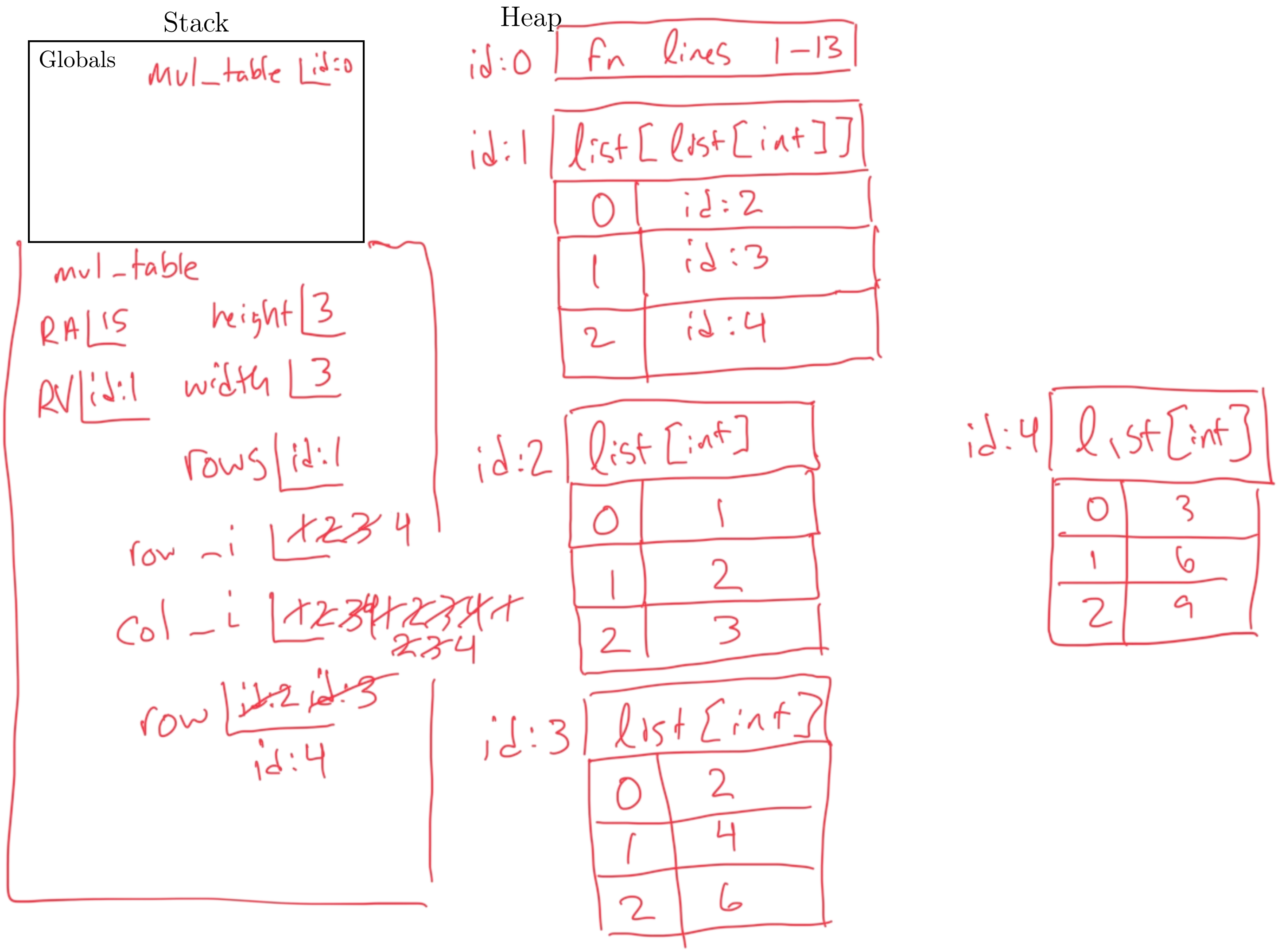


Question 7: Memory Diagram Trace a memory diagram of the following code listing.

```
1  def mul_table(height: int, width: int) -> list[list[int]]:
2    """Generate a multiplication table."""
3    rows: list[list[int]] = []
4    row_i: int = 1
5    while row_i <= height:
6      col_i: int = 1
7      row: list[int] = []
8      while col_i <= width:
9        row.append(row_i * col_i)
10       col_i += 1
11     rows.append(row)
12     row_i += 1
13   return rows
14
15 print(mul_table(3, 3))
```

Output

Solution: [[1, 2, 3], [2, 4, 6], [3, 6, 9]]



Question 8: Function Writing Write a function definition for `reverse` with the following expectations:

- The `reverse` function should accept a `list[str]` parameter and return a `list[str]`.
- The returned `list` should have every item of the parameter list in reversed order, such that the first value of the returned list was the last value of the input list, the second value of the returned list was the second to last value of the input list, and so on.
- The function *must not mutate* its parameter.
- The function *must not use* the `copy`, `reverse`, or `insert` methods of `list`.
- You should explicitly type all variables, parameters, and return types.

8.1. Write your function definition for `reverse` here.

Solution: One possible solution, of many possible valid solutions:

```
def reverse(xs: list[str]) -> list[str]:
  """Reverse elements of input list without mutation."""
  reversed: list[str] = []
  idx: int = len(xs) - 1
  while idx >= 0:
    reversed.append(xs[idx])
    idx -= 1
  return reversed

```

8.2. Write a test function for a use case that demonstrates expected usage with at least three values in the list.

Solution: One possible test function, of many possible valid test functions:

```
def test_reverse_3() -> None:
  """Test reversal of three element list."""
  assert reverse(["one", "two", "three"]) == ["three", "two", "one"]

```

Question 9: Function Writing Write a function definition for `flip_flop` with the following expectations:

- The `flip_flop` function should accept a `list[str]` parameter and return `None`.
- The function *must mutate* its parameter such that pairs of subsequent indices are swapped. For example, index 0's value should be swapped with index 1's value. Index 2's value should be swapped with index 3's value, and so on. If there are an odd number of indices, leave the final element in its place.
- You should explicitly type all variables, parameters, and return types.

9.1. Write your function definition for `flip_flop` here.

Solution: One possible solution, of many possible valid solutions:

```
def flip_flop(strs: list[str]) -> None:
  idx: int = 1
  while idx < len(strs):
    temp: str = strs[idx]
    strs[idx] = strs[idx - 1]
    strs[idx - 1] = temp
    idx += 2

```

9.2. Write a test function for a use case that demonstrates expected usage with at least three values in the list.

Solution: One possible test function, of many possible valid test functions:

```
def test_flip_flop_5() -> None:
  """Test flip flop with 5 elements"""
  letters: list[str] = ["a", "b", "c", "d", "e"]
  flip_flop(letters)
  assert letters == ["b", "a", "d", "c", "e"]

```

This page intentionally left blank. Do not remove from quiz packet.