

## Quiz 03 - Practice

COMP 110: Introduction to Programming  
Spring 2024

Thursday April 11, 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** For each of the next questions, select all of `set`, `list`, `dict`, and/or `tuple` for which the statement describes. Bubble in ALL squares that apply.

- 1.1. Which of the following data structures are sequences?  
 tuple    list    set    dict
- 1.2. Select all data structures that are mutable.  
 tuple    list    set    dict
- 1.3. Select all data structures that can contain duplicate elements.  
 tuple    list    set    dict
- 1.4. Which of these data structures use key-value pairs for storing data?  
 tuple    list    set    dict
- 1.5. Which of the following data structures does not guarantee the order of elements? (The `dict` data structure is intentionally omitted; in Python, order is maintained. However, generally, `dict`-like data structures do not guarantee ordering.)  
 tuple    list    set
- 1.6. Which data structures allow indexing via subscription notation to access individual elements directly?  
 tuple    list    set    dict
- 1.7. If you need to store a collection of items and frequently check whether an item is in the collection, which data structure is most efficient?  
 tuple    list    set    dict
- 1.8. To ensure the order of elements is maintained and allow for duplicates, which data structure would you choose?  
 tuple    list    set    dict
- 1.9. For a fixed collection of elements that should not be altered, which data structure is the most appropriate?  
 tuple    list    set    dict
- 1.10. To store a sequence of elements that you intend to iterate over and modify, which data structure offers the best performance?  
 tuple    list    set    dict
- 1.11. For associating student PIDs to their respective email addresses, which data structure provides the most efficient lookup?  
 tuple    list    set    dict
- 1.12. Which of the following could use use as a *key type* in a `dict`? (Hint: keys must be *immutable*)  
 tuple    list    set    dict
- 1.13. Which data structure's *literal syntax* is enclosed within parentheses?  
 tuple    list    set    dict
- 1.14. Which data structure's *literal syntax* is enclosed within curly braces?  
 tuple    list    set    dict
- 1.15. Which data structure's *literal syntax* is enclosed within square brackets?  
 tuple    list    set    dict
- 1.16. Which data structures can you iterate over using a `for..in` loop?  
 tuple    list    set    dict
- 1.17. Which data structures allow the use of the `len` function to determine the *number of elements* it contains?  
 tuple    list    set    dict
- 1.18. Which of the following data structures is best when you want to find the *intersection*, *union*, or *difference* of two collections of values?  
 tuple    list    set    dict
- 1.19. If you were creating a messaging app, where you want to maintain a list of messages in the order they were received, which data structure would you use?  
 tuple    list    set    dict
- 1.20. When trying to count the frequency of words in a document, which data structure would allow you to efficiently store and update counts?  
 tuple    list    set    dict

**Question 2: Respond** to the following questions

Consider the following function signatures:

```
1 def a(x: float, y: float) -> float: ...
2 def b(a: str) -> int: ...
3 def c(x: int) -> bool: ...
```

2.1. What is the `Callable` type of `a`?

**Solution:** `Callable[[float, float], float]`

2.2. What is the `Callable` type of `b`?

**Solution:** `Callable[[str], int]`

2.3. What is the `Callable` type of `c`?

**Solution:** `Callable[[int], bool]`

**Question 3: Respond** to the following questions

Consider the following generic `Callable` type aliases and function signatures:

```
1 Transform = Callable[[T], U]
2 Predicate = Callable[[T], bool]
3 BinaryFunc = Callable[[T, U], V]
4
5 def f(x: int) -> bool: ...
6 def g(x: int) -> double: ...
7 def h(x: float, y: float) -> float: ...
8 def a(x: str, y: int) -> bool: ...
9
10 def hof(t: Transform[int, double]) -> bool: ...
```

3.1. Which of the function names conform to the `Transform` type?

**Solution:** `f, g`

3.2. Which of the function names conform to the `Predicate` type?

**Solution:** `f`

3.3. Which of the function names conform to the `BinaryFunc` type?

**Solution:** `h, a`

3.4. Given the function signatures defined above, write a function call to the ‘`hof`’ function:

**Solution:** `hof(g)`

**Question 4: Respond** to the following questions using Python's builtin `filter` and `map` functions.

Consider the following functions:

```
1 def a(x: float) -> bool:  
2     return x >= 0.0  
3  
4 def b(x: bool) -> bool:  
5     return not x  
6  
7 def c(x: float) -> str:  
8     return f"-> {x} <-"  
9  
10 def d(x: str) -> float:  
11     return float(x)
```

4.1. What is the evaluation of `list(map(a, [1.0, 0.0, -1.0, 2.0]))` in list literal notation?

**Solution:** [True, True, False, True]

4.2. What is the evaluation of `list(filter(a, [1.0, 0.0, -1.0, 2.0]))` in list literal notation?

**Solution:** [1.0, 0.0, 2.0]

4.3. What is the evaluation of `list(map(b, [True, False, True]))` in list literal notation?

**Solution:** [False, True, False]

4.4. What is the evaluation of `list(filter(b, [True, False, True]))` in list literal notation?

**Solution:** [False]

4.5. What is the evaluation of `list(map(c, [110.0, 210.0]))` in list literal notation?

**Solution:** ["-> 110.0 <-", "-> 210.0 <-"]

4.6. What is the evaluation of `list(map(d, ["110.0", "210.0"]))` in list literal notation?

**Solution:** [110.0, 210.0]

4.7. What is the evaluation of `list(filter(a, map(d, ["-100.0", "110.0"])))` as a list literal?

**Solution:** [110.0]

4.8. What is the evaluation of `list(map(c, map(d, ["-100.0", "110.0"])))` as a list literal?

**Solution:** ["-> -100.0 <-", "-> 110.0 <-"]

**Question 5: Memory Diagram** Trace a memory diagram of the following code listing. For the purposes of diagramming, you can ignore the imports, TypeVars, and type aliases.

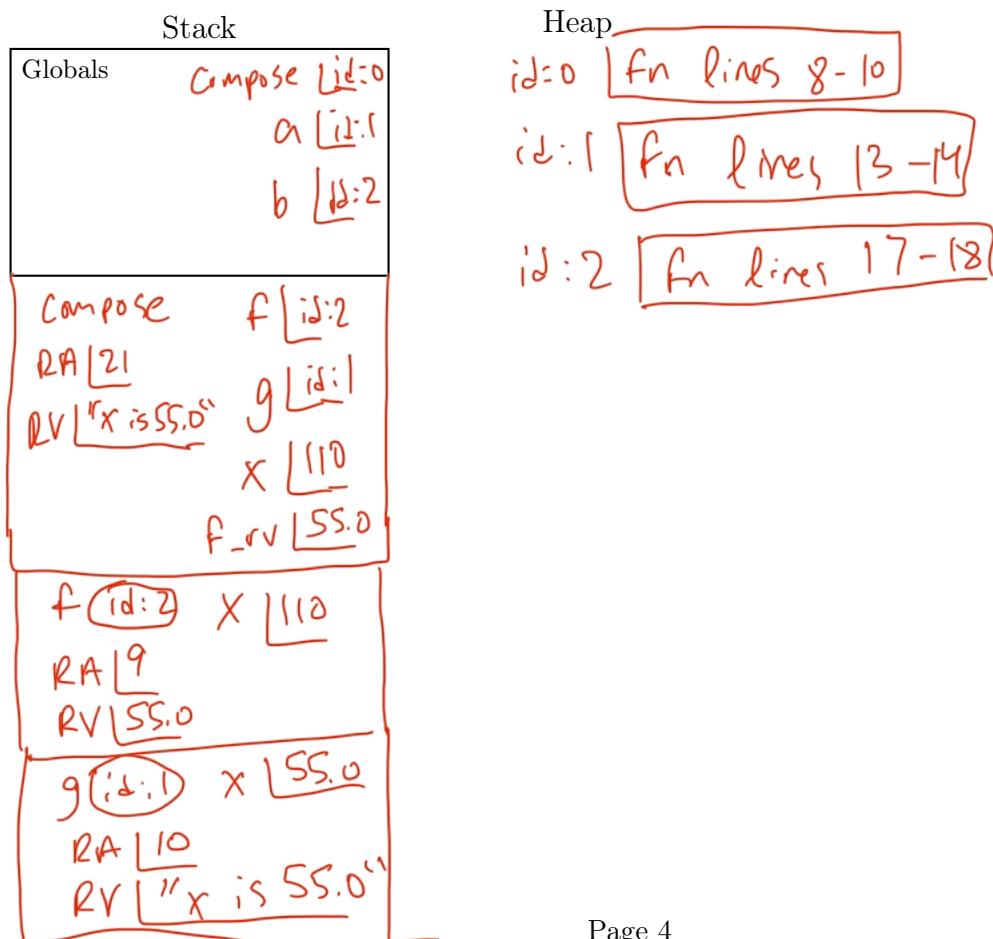
```

1 from typing import Callable, TypeVar
2
3 T = TypeVar("T")
4 U = TypeVar("U")
5 Transform = Callable[[T], U]
6
7
8 def compose(f: Transform[int, float], g: Transform[float, str], x: int) -> str:
9     f_rv: float = f(x)
10    return g(f_rv)
11
12
13 def a(x: float) -> str:
14     return f"x is {x}"
15
16
17 def b(x: int) -> float:
18     return x / 2.0
19
20
21 print(compose(b, a, 110))

```

Output

Solution: ~~XXXXXX~~ X is 55.0



**Question 6: Memory Diagram** Trace a memory diagram of the following code listing. For the purposes of diagramming, you can ignore the imports, TypeVars, and type aliases.

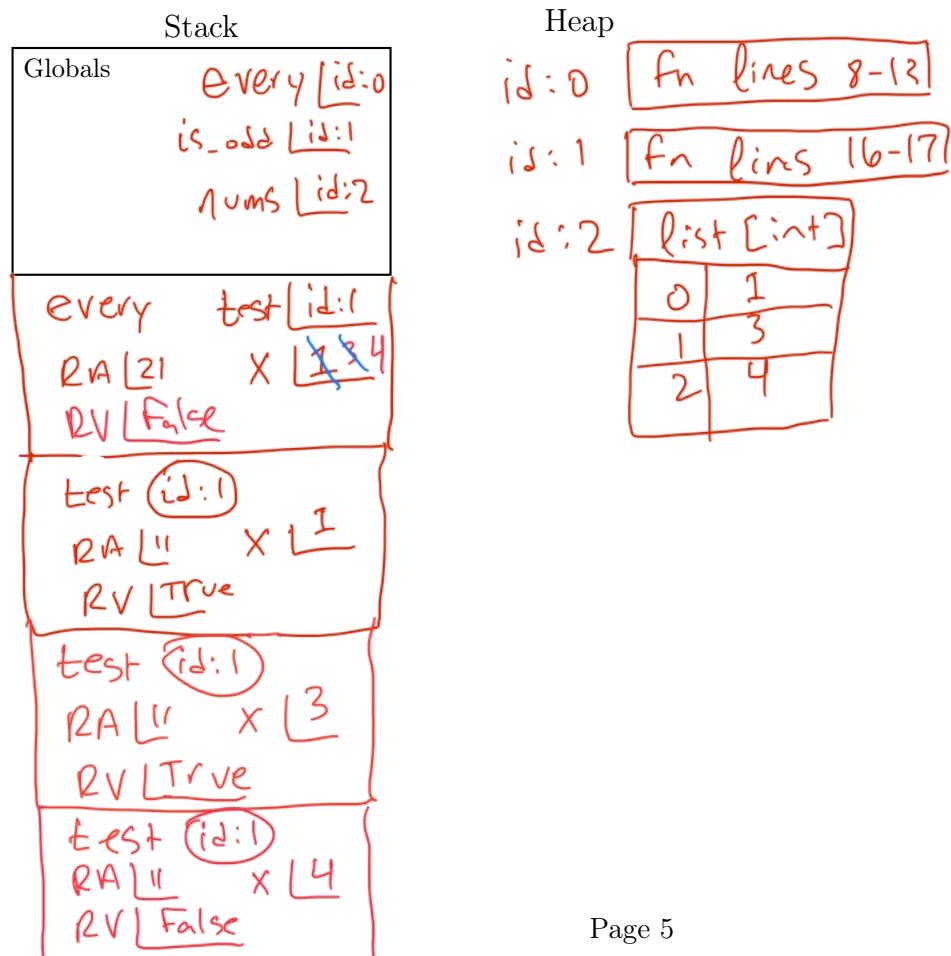
```

1 from typing import TypeVar, Callable
2 from collections.abc import Iterable
3
4 T = TypeVar("T")
5 Predicate = Callable[[T], bool]
6
7
8 def every(test: Predicate[T], xs: Iterable[T]) -> bool:
9     """A mysterious higher-order function..."""
10    for x in xs:
11        if not test(x):
12            return False
13    return True
14
15
16 def is_odd(x: int) -> bool:
17     return x % 2 == 1
18
19
20 nums: list[int] = [1, 3, 4]
21 print(every(is_odd, nums))

```

Output

**Solution:** False



**Question 7: Memory Diagram** Trace a memory diagram of the following code listing. For the purposes of diagramming, you can ignore the imports, TypeVars, and type aliases.

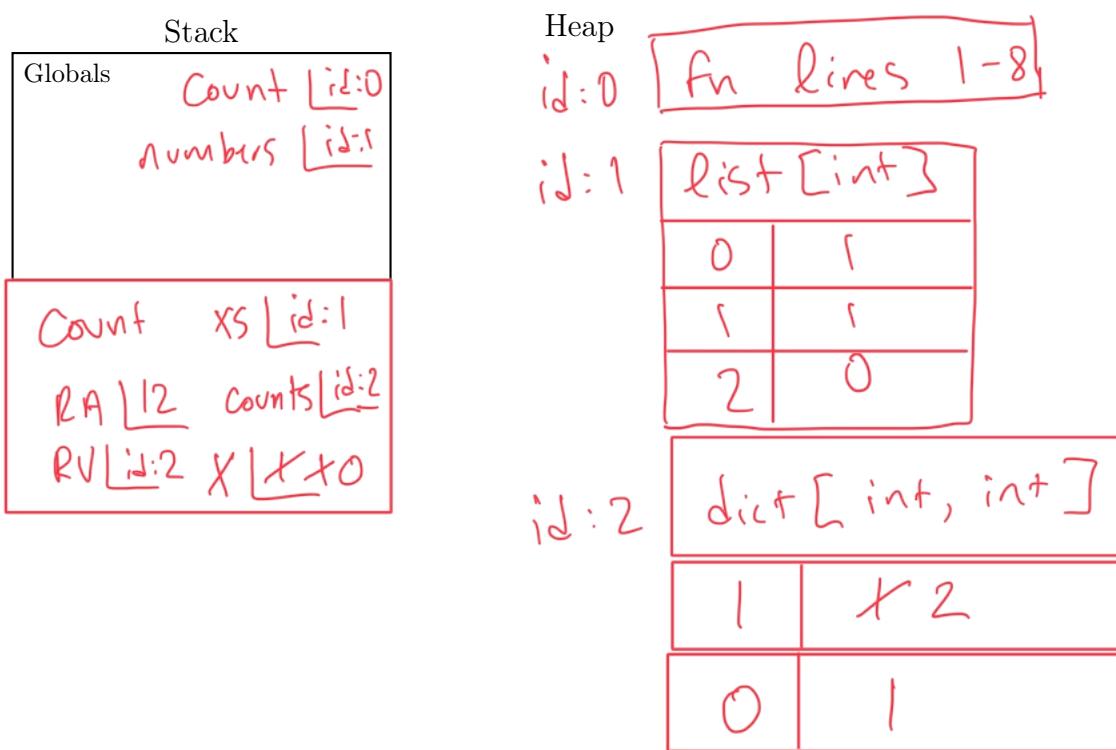
```

1 def count(xs: list[int]) -> dict[int, int]:
2     counts: dict[int, int] = {}
3     for x in xs:
4         if x in counts:
5             counts[x] += 1
6         else:
7             counts[x] = 1
8     return counts
9
10
11 numbers: list[int] = [1, 1, 0]
12 print(count(numbers))

```

Output

**Solution:** {1:2, 0:1}



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

- The `any` function should accept a `Callable[[str], bool]` "predicate" test function and a `list[str]` as parameters. It should return a `bool`.
- The function should return `True` if *any str* item in the list parameter, when used as an argument to call the callable predicate parameter, returns `True`. Otherwise, this function should return false.
- You should explicitly type all variables, parameters, and return types.

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

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

```
1 def any(test: Callable[[str], bool], xs: list[str]) -> bool:  
2     for x in xs:  
3         if test(x):  
4             return True  
5     return False  
6
```

8.2. Write a valid function that could be used with `any` and returns whether a given string is greater than 3 characters long.

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

```
1 def longerThan3(s: str) -> bool:  
2     return len(s) > 3  
3
```

8.3. Write an example function call to `any` making use of the function defined above and a list of length 2 that will result in a `False` value being returned by `any`.

**Solution:** One possible call, of many possible valid calls:

```
1 any(longerThan3, ["a", "bb", "ccc", "d"])  
2
```

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

- The `count_lens` function should accept a list of string values and return a dictionary where the key type is `int` and the value type is `int`.
- The function should *count the frequencies* of strings in the parameter list of the same length(s). For example, `["a", "b", "cc", "d"]` should return `{1: 3, 2: 1}` because there were three strings of length 1 and one string of length 2.
- You should explicitly type all variables, parameters, and return types.

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

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

```
1 def count_lens(strs: list[str]) -> dict[int, int]:  
2     counts: dict[int, int] = {}  
3     for s in strs:  
4         if len(s) in counts:  
5             counts[len(s)] += 1  
6         else:  
7             counts[len(s)] = 1  
8     return counts  
9 
```

9.2. Write a test function for a use case that demonstrates expected usage with at least three values in the list. Your input should be different from the prompt's sample input.

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

```
1 def test_count_lens() -> None:  
2     """Test flip flop with 5 elements"""  
3     letters: list[str] = [{"a", "b", "cc", "dd"}]  
4     assert count_lens(letters) == {1: 2, 2: 2} 
```

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