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:

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

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

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

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

Question 3: Respond to the following questions

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

```python
Transform = Callable[[T], U]
Predicate = Callable[[T], bool]
BinaryFunc = Callable[[T, U], V]

def f(x: int) -> bool: ...
def g(x: int) -> double: ...
def h(x: float, y: float) -> float: ...
def a(x: str, y: int) -> bool: ...

def hof(t: Transform[int, double]) -> bool: ...
```

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

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

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

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

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

Consider the following functions:

```
def a(x: float) -> bool:
  return x >= 0.0

def b(x: bool) -> bool:
  return not x

def c(x: float) -> str:
  return f"-> {x} <-"

def d(x: str) -> float:
  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?

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

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

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

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

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

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

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

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

```
from typing import Callable, TypeVar

T = TypeVar("T")
U = TypeVar("U")
Transform = Callable[[T], U]


def compose(f: Transform[int,float], g: Transform[float,str], x: int) -> str:
  f_rv: float = f(x)
  return g(f_rv)


def a(x: float) -> str:
  return f"x is {x}"


def b(x: int) -> float:
  return x / 2.0


print(compose(b, a, 110))
```

Output

Stack

Heap

Globals

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

```python
from typing import TypeVar, Callable
from collections.abc import Iterable

T = TypeVar("T")
Predicate = Callable[[T], bool]


def every(test: Predicate[T], xs: Iterable[T]) -> bool:
  """A mysterious higher-order function..."""
  for x in xs:
    if not test(x):
      return False
  return True


def is_odd(x: int) -> bool:
  return x % 2 == 1


nums: list[int] = [1, 3, 4]
print(every(is_odd, nums))
```

Output

Stack

Heap

Globals

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

```python
def count(xs: list[int]) -> dict[int, int]:
  counts: dict[int, int] = {}
  for x in xs:
    if x in counts:
      counts[x] += 1
    else:
      counts[x] = 1
  return counts


numbers: list[int] = [1, 1, 0]
print(count(numbers))
```

Output

Stack

Heap

Globals

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.

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

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`.

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.

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.

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