CSE3500 Python Practice Problems

You're expected to know Python before taking this course, as it is included in the prerequisites. But understandably some people are a bit shaky on it, or simply due to the passing of time have forgotten some of it. These problems are meant to serve as a refresher so that you can brush up on your skills

I remember when I first started CS at UConn, and I marveled at people who were crushing the programming assignments while I struggled with some of them. I could not understand how they got to be so good and it seemed like programming just came easy to them, and I felt like I would never reach that stage. The truth was far simpler - they had just spent way more hours programming than I had. And when I had put in more time than them, I became a stronger programmer than them.

I bring this up to say if you are struggling with these and with your Python skills in general, chances are it's due to a low amount of time spent programming. I strongly recommend to practice practice practice. Split your time between leetcode style problems, personal projects, and extra learning on the side. Even just an hour a day will make a big difference. If you spend 3-4 hours a day outside of class programming, within a single semester you'll probably be in the top 10% of programmers at UConn.

The other reason I stress practice, is because this class is a really good litmus test for how you would fare in a job interview. It's tough to assess programming abilities, and it's definitely possible many in this class have received good grades up until this point, but don't know how to program well. That will become very apparent in any programming interview and it will be very tough to get a job in today's competitive environment. I've labeled these questions with corresponding difficulty. The vast majority are considered "instant-fail" questions. Where, if you can not figure out the question within 1-2 minutes, you'd be rejected in an interview. Typically interview questions involve algorithms and are even harder than these types of questions (and these types of questions will be on quizzes and exams). I've also labeled which questions were actually in interviews. I don't want to sugarcoat it, if you are struggling here, you are behind, and you will likely struggle when trying to find a job, probably to the point of not getting one at all. I don't want to frighten people, but I also don't want people to graduate and then be unable to get a job. The good news is, you can easily get better and fix this by putting in a lot of hours practicing to correct this before your job search.

Lastly, I don't see any reason to delay the answers for these problems past the deadline, as you can just look the answers up anyways. I've included them here. Just go problem by problem, BEFORE looking at the answer and checking your work. Pretend this is a real interview. Check for all of your edge cases before looking at the solution, as these will count against you in interviews. For the short answer section, use ChatGPT or your favorite AI to help your understanding.

1. Reverse a list. Difficulty: Instant fail. Real interview question: Yes

   python lists have a .reverse() I guess?

2. Consider the following code:

```
a = [1, 2, 3]
b = a
b[0] = 5
print(a)
# What will this print?
```

[5, 2, 3]

Answer:

```
a = [1, 2, 3]
b = a
b[0] = 5
print(a)
# What will this print?
# [5, 2, 3]
```

3. Change line number 2 (into any number of lines) so that this is a deep copy rather than a shallow copy. Difficulty: Instant fail. Real interview question: Yes

```
a = [1, 2, 3]
b = a  # Make this a deep copy
b[0] = 5
print(a)
```

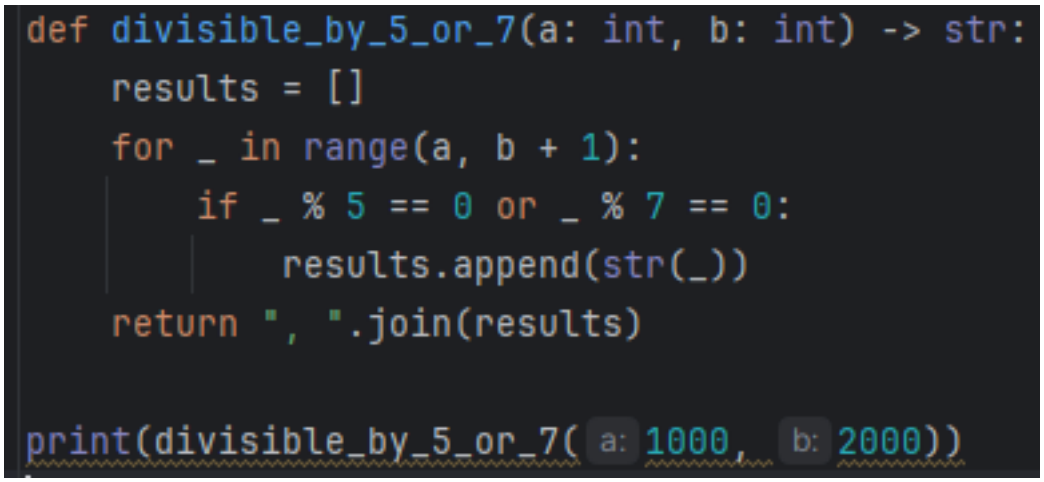
b = a.copy()

Answer:

```
a = [1, 2, 3]
# Any of these would work. And there are several others.
b = a.copy()
b = a[:]
b = list(a)
b = []
for x in a:
    b.append(x)
b[0] = 5
print(a)
# What will this print?
# [5, 2, 3]
```

4. Write a function that takes two numbers, a and b, where b is higher than a. Print every number between a and b that is divisible by 5 or 7, on one line, separated by commas. Call your function with a being 1000, and b being 2000. For example, if you were to call this function with the numbers 1 and 10, the output printed would be: 5, 7, 10. Notice there is no trailing comma at the end. Difficulty: Instant fail. Real interview Question: Yes

---



```python
def divisible_by_5_or_7(a: int, b: int) -> str:
    results = []
    for _ in range(a, b + 1):
        if _ % 5 == 0 or _ % 7 == 0:
            results.append(str(_))
    return ", ".join(results)

print(divisible_by_5_or_7( a: 1000,  b: 2000))
```

Answer:

```
def foo(a, b):
    nums = ""
    for x in range(a, b + 1):
        if x % 5 == 0 or x % 7 == 0:
            nums += str(x) + ","  # Or better, use f strings f"{x}, "
    print(nums[:-1])
```

Answer:

```
def foo(a, b):
    nums = []
    for x in range(a, b + 1):
        if x % 5 == 0 or x % 7 == 0:
            nums.append(x)
    print(', '.join(nums))
```

5. Change the above solution. Store everything in a list, and use the .join method to bring it together into a string. For example, ', '.join([1, 2, 3]) outputs "1, 2, 3". Use a list comprehension. Difficulty: Probably Instant fail. Real interview Question: Yes

```
print(", ".join([str(_) for _ in range(1000, 2000 + 1) if _ % 5 == 0 or _ % 7 == 0]))
```

Answer:

```
def foo(a, b):
    print(', '.join(x for x in range(a, b + 1) if x % 5 == 0 or x % 7 == 0))
```

6. Write a function that returns the total count of each character in a string in a dictionary, regardless of whether it is uppercase or lowercase. Difficulty: Instant fail. Real interview Question: Yes

---

```python
def count_letters_in_string(s: str) -> dict:
    letters = {"a": 0, "b": 0, "c": 0, "d": 0, "e": 0, "f": 0, "g": 0, "h": 0, "i": 0, "j": 0, "k": 0, "l": 0, "m": 0,
               "n": 0, "o": 0, "p": 0, "q": 0, "r": 0, "s": 0, "t": 0, "u": 0, "v": 0, "w": 0, "x": 0, "y": 0, "z": 0}
    for l in s.lower():
        letters[l] += 1
    return letters
print(count_letters_in_string("Writeafunctionhatreturnsthetalcountofachcharacterinastringndictionary"))
```

Answer:

```
def foo(string):
    freq = {}
    for c in string:
        if c in freq:
            freq[c] += 1
        else:
            freq[c] = 1
    return freq
```

Answer:

```
def foo(string):
    freq = {}
    for c in string:
        freq[c] = freq.get(c, 0) + 1
    return freq
```

Answer:

```
from collections import defaultdict

def foo(string):
    freq = defaultdict(int)
    for c in string:
        freq[c] += 1
    return freq
```

Answer:

```
from collections import Counter

def foo(string):
    return Counter(string)
```

7. Write a class Student that has an attribute for grade (which is a float), and a function that prints whether or not the student is passing or failing. If the student has above a 60, they pass, otherwise they fail. Subclass this student in a new class called CSEStudent. CSEStudent should have an additional attribute for how many CS classes they have taken. Change the function that determines whether or not the student is passing by making it so that a 70 is required to pass. Initialize a CSEStudent object with 5 classes taken and a grade of 65. Difficulty: Instant fail. Real interview Question: Yes

---

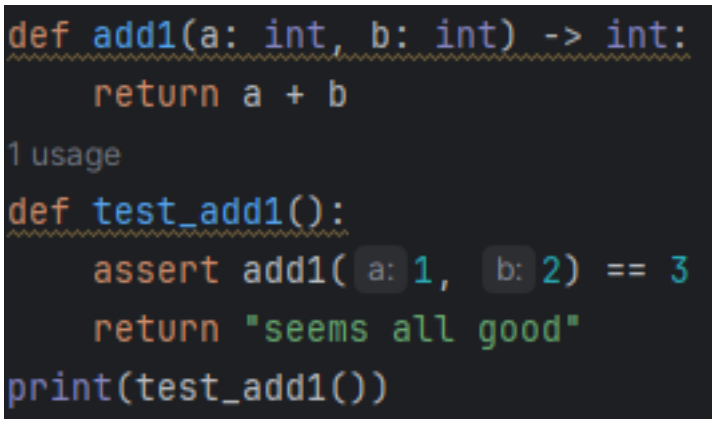
```python
class Student:
    def __init__(self, grade: float):
        self.grade = grade
        self.passing = 60.0

    1 usage
    def pass_fail(self):
        if self.grade < self.passing:
            return "Fail"
        return "Pass"


1 usage
class CSEStudent(Student):
    def __init__(self, grade: float, CSE_classes: int):
        super().__init__(grade)
        self.CSE_classes = CSE_classes
        self.passing = 70.0

sad_student = CSEStudent(grade=65.0, CSE_classes=5)
print(sad_student.pass_fail())
```

Answer:

```
class Student:
    def __init__(self, grade: float):
        self.grade = grade

    def print_status(self):
        if self.grade > 60:
            print("Passing")
        else:
            print("Failing")


class CSEStudent(Student):
    def __init__(self, grade: float, cs_classes_taken: int):
        super().__init__(grade)
        self.cs_classes_taken = cs_classes_taken

    def print_status(self):
        if self.grade > 70:
            print("Passing")
        else:
            print("Failing")


# Initialize a CSEStudent with 5 CS classes taken and a grade of 65
student = CSEStudent(grade=65, cs_classes_taken=5)
student.print_status()
```

8. Write a unit tests in Python for a function that adds two numbers Difficulty: Instant-fail. Real interview Question: Yes

---



Answer:

```
def add(a, b):
    return a + b

def test_add():
    assert add(1, 2) == 3, "Add fails"
```

9. You have a CSV. The headers are "name", "birthday", and "year". The years are all valid years. Add 1 to each year and overwrite the original CSV file. The input file is data.csv, and the output file is new_data.csv. Difficulty: Instant-fail. Real interview Question: The actual question was harder, this is a simplified version.

```
def csv_editor1():
    csv = pd.read_csv("dns/data.csv")
    csv["year"]+=1
    csv.to_csv("dns/new_data.csv")
csv_editor1()
```

Answer: Cons to this approach: header changes can break code, if files are large you are reading everything before writing

```
import csv

rows = []

with open("data.csv", newline="") as f:
    reader = csv.reader(f)
    headers = next(reader)
    for row in reader:
        row[2] = str(int(row[2]) + 1)  # year is column index 2
        rows.append(row)

with open("new_data.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(headers)
    writer.writerows(rows)
```

Answer: Cons to this approach: if files are large you are reading everything before writing

```
import csv

rows = []

with open("data.csv", newline="") as f:
    reader = csv.DictReader(f)
    fieldnames = reader.fieldnames
    for row in reader:
        row["year"] = str(int(row["year"]) + 1)
        rows.append(row)

with open("new_data.csv", "w", newline="") as f:
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(rows)
```

Answer (imo optimal):

```
import csv

with open("data.csv", newline="") as infile, open(data_new, "w", newline="") as outfile:
    reader = csv.DictReader(infile)
    writer = csv.DictWriter(outfile, fieldnames=reader.fieldnames)

    writer.writeheader()
    for row in reader:
        row["year"] = str(int(row["year"]) + 1)
        writer.writerow(row)
```

Answer with pandas:

```
import pandas as pd

df = pd.read_csv("data.csv")
df["year"] = df["year"] + 1
df.to_csv("data.csv", index=False)
```

10. The same question as before, except this time, use no external libraries. Assume there are no commas in any names. Additionally, explain why this solution is dangerous. You have a CSV. The headers are "name", "birthday", and "year". The years are all valid years. Add 1 to each year and overwrite the original CSV file. The input file is data.csv, and the output file is new_data.csv. Difficulty: Instant-fail. Real interview Question: No, but I actually like this one quite a bit. It tests basic string manipulation and file reading

---

```
def vanilla_csv_editor1():
    with open("dns/b.csv", "r") as csv:
        file = csv.readlines()
        data = file[1:]
        new_data = []
        for line in data:
            n, b, y = line.strip().split(",")
            y = str(int(y) + 1)
            new_data.append(f"{n},{b},{y}\n")
    with open("dns/b.csv", "w") as csv:
        csv.write(file[0])
        csv.writelines(new_data)


vanilla_csv_editor1()
```

Answer:

```
lines = []

with open("data.csv") as f:
    lines = f.readlines()

headers = lines[0]
rows = lines[1:]

new_rows = []
for row in rows:
    name, birthday, year = row.strip().split(",")
    year = str(int(year) + 1)
    new_rows.append(f"{name},{birthday},{year}\n")

with open("data.csv", "w") as f:
    f.write(headers)
    f.writelines(new_rows)
```

11. Design and implement a model of a standard deck of playing cards using object-oriented programming in Python. Create a Card class that represents an individual playing card with a suit and a rank, and ensure that a card has a human-readable string representation when printed. Create a Deck class that represents a full deck of 52 unique cards, one for each combination of standard suits and ranks. The deck should be able to be shuffled, should allow cards to be dealt and removed from the deck, and should behave like a typical Python collection by supporting the use of len() to return the number of remaining cards, iteration in a for loop, and indexing to access a specific card. Use appropriate Python dunder methods to achieve this behavior, and structure your solution in a clean, idiomatic, and object-oriented way. Difficulty: Medium. Real interview question: Yes

---

```python
class Card:
    def __init__(self, suit: str, rank: str):
        self.suit = suit
        self.rank = rank

    def __repr__(self) -> str:
        return f"{self.rank} of {self.suit}"
```

```python
class Deck:
    def __init__(self):
        self.suits = ["Hearts", "Spades", "Diamonds", "Clubs"]
        self.ranks = ["Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen",
                      "King"]
        self.reset()

    def __len__(self):
        return len(self.cards)

    def __getitem__(self, i: int):
        return self.cards[i]

    def __iter__(self):
        return iter(self.cards)

    1 usage
    def reset(self, shuffle=True):
        self.cards = [Card(s, r) for s in self.suits for r in self.ranks]
        if shuffle:
            self.shuffle()

    1 usage
    def shuffle(self):
        r.shuffle(self.cards)

    def print_deck(self):
        for card in self.cards:
            return card

    def deal_card(self):
        self.cards.pop(0)
```

12. Build a cache decorator similar to the lru cache decorator in Python. Bonus points if you can support max size limitations. Difficulty: In-between medium and hard. Real interview Question: Yes. While you won't be required to be able to write a decorator in this course, a decorator is merely a function that returns another function. You should know that and be able to do this problem at this point in your curriculumn based on what you've already covered.

Answer:

```
from functools import wraps

def simple_cache(func):
    cache = {}

    @wraps(func)  # not necessary for credit
    def wrapper(*args, **kwargs):
        key = (args, tuple(sorted(kwargs.items())))
        if key not in cache:
            cache[key] = func(*args, **kwargs)
        return cache[key]
    return wrapper
```

Answer with maxsize:

```
from functools import wraps

def lru_cache_custom(maxsize=128):
    def decorator(func):
        cache = {}

        @wraps(func)  # not necessary for credit
        def wrapper(*args, **kwargs):
            key = (args, tuple(sorted(kwargs.items())))

            if key in cache:
                # Move key to end (most recently used)
                value = cache.pop(key)
                cache[key] = value
                return value

            result = func(*args, **kwargs)
            cache[key] = result

            if len(cache) > maxsize:
                # Remove least recently used (first inserted key)
                oldest_key = next(iter(cache))
                del cache[oldest_key]

            return result

        return wrapper
    return decorator
```

13. Design and implement a model of a standard deck of playing cards using object-oriented programming in Python. Create a Card class that represents an individual playing card with a suit and a rank, and ensure that a card has a human-readable string representation when printed. Create a Deck class that represents a full deck of 52 unique cards, one for each combination of standard suits and ranks. The deck should be able to be shuffled, should allow cards to be dealt and removed from the deck, and should behave like a typical Python collection by supporting the use of len() to return the number of remaining cards, iteration in a for loop, and indexing to access a specific card. Use appropriate Python dunder methods to achieve this behavior, and structure your solution in a clean, idiomatic, and object-oriented way. Difficulty: Medium. Real interview question: Yes. NOTE: while you won't be required to know a lot of the dunder/magic methods in this course, you should have covered them in prior classes and understand them by now. Plenty of algorithms do involve classes (especially with any data structure involving nodes), so these things do matter even for algorithms.

```python
class Card:
    def __init__(self, suit: str, rank: str):
        self.suit = suit
        self.rank = rank

    def __repr__(self) -> str:
        return f"{self.rank} of {self.suit}"
```

```python
class Deck:
    def __init__(self):
        self.suits = ["Hearts", "Spades", "Diamonds", "Clubs"]
        self.ranks = ["Ace", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen",
                      "King"]
        self.reset()

    def __len__(self):
        return len(self.cards)

    def __getitem__(self, i: int):
        return self.cards[i]

    def __iter__(self):
        return iter(self.cards)

    1 usage
    def reset(self, shuffle=True):
        self.cards = [Card(s, r) for s in self.suits for r in self.ranks]
        if shuffle:
            self.shuffle()

    1 usage
    def shuffle(self):
        r.shuffle(self.cards)

    def print_deck(self):
        for card in self.cards:
            return card

    def deal_card(self):
        self.cards.pop(0)
```

Answer:

```
class Card:
    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit

    def __str__(self):
        return f"{self.rank} of {self.suit}"

    def __repr__(self):
        return f"Card({self.rank}, {self.suit})"

import random

class Deck:
    SUITS = ["Hearts", "Diamonds", "Clubs", "Spades"]
    RANKS = ["2", "3", "4", "5", "6", "7", "8", "9", "10",
             "Jack", "Queen", "King", "Ace"]

    def __init__(self):
        self.cards = [Card(rank, suit) for suit in self.SUITS for rank in self.RANKS]

    def shuffle(self):
        random.shuffle(self.cards)

    def deal(self):
        if not self.cards:
            raise IndexError("No cards left in the deck")
        return self.cards.pop()

    def __len__(self):
        return len(self.cards)

    def __iter__(self):
        return iter(self.cards)

    def __getitem__(self, position):
        return self.cards[position]

    def __str__(self):
        return f"Deck of {len(self)} cards"

deck = Deck()
print(deck)           # Deck of 52 cards

deck.shuffle()

card = deck.deal()
print(card)           # e.g. "Queen of Spades"

print(len(deck))      # 51

print(deck[0])        # Indexing works

for card in deck:     # Iteration works
    print(card)
```

14. Let's say your Python code is running slow. How would you fix it? Difficulty: Medium. Real interview Question: Yes. Also: this may not have been gone over in other classes, but I'm including it here anyways since it can be useful to know. You can skip this question. Don't worry if you don't know this, and it won't be a requirement for this course, but it can be good to know for programming and interviews!

Answer: Start by using cProfile to profile your code and figure out exactly where the slowdown is occuring. Determine if it's a poorly implemented algorithm, and if so, correct it. From there, if we have implemented the optimal solution and our code is still running slowly, there are several tricks we can use. We can use PyPy instead of Python to speed up our code, assuming there are no dependencies that would prevent this. We can use numba if applicable. We can write that portion of the code using C or C++ if applicable, using bindings. If it's a function that's repeatedly being called with similar inputs, we can cache them. If it involves objects, we can use slots to improve performance. We can swap out data types like lists to faster datatypes like tuples. etc.

And here are some short answer, open-ended questions that come up in interviews. I've been asked all of these, and they've been covered in your classes up to this point (as far as I know).

15. What's the difference in how Python runs vs how a compiled language runs? python runs code line by line, while a compliled language compiles all the code BEFORE running even the first line
16. What is Pep8? PEP8 is the style guidelines for python
17. How is memory and garbage collection handled in Python? memory is managed using heaps, and garbage collection is automatically done by python
18. What's the difference between a list and tuple, and when would you use one over the other? Lists are used when the data is subject to change, and tuples are used when the data is static
19. How do you handle errors in Python in your code that you are expecting? I would use a try-except block
20. How does the == work under the hood? also known as __eq__, it compares values and returns a boolean
21. What are generators and when should you use them? Generators are functions that use yield, useful if there is a large quantity of data, and you don't want to juggle a single return line
22. what's the point of the famous "`if name == "__main__"`? this basically asks: is this file ran from this file? this prevents code running more times than nessessary if assessed using imports
23. What does yield do in Python? it acts much like return, except it doesn't reset the function when run
24. What's the method resolution order in Python? Order Python searches for methods in inheritance hierarchy
25. How does multiple inheritance work in Python? Class can inherit from multiple parents, MRO determines method lookup order
26. How do you install Python packages? using pip3 install

And below are some more questions that you perhaps haven't gone over in class, but are still relevant to learn for general programming and interviews. They won't affect this class, so if you don't want to do them, don't bother.

27. What are decorators and when should you use them? functions that modify other functions
28. What are context managers and when should you use them?
29. What's the global interpreter lock (GIL)?
30. When should you use threading in Python?
31. What are property methods, static methods, and class methods, and when should you use each of them?
32. What is the constructor in Python (hint, it's not dunder init)
33. Why do people use PyTest over unittest? (and other PyTest specific questions)
34. If you want to store a Python object for later use, how do you do it?
35. General back-end questions, like what is a REST API
36. How do you automate web browsing in Python for things like UI testing?
37. How do Python virtual environments work?

38. What are regular expressions and how do they work in Python?

Lastly, you can find a lot of good open-ended interview questions that don't involve coding (which are often asked as ice-breakers) just by googling "100 Python interview Questions". This can also be good practice and can expose some holes in your knowledge.