Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Midterm 2 Review

Question 1

Implement a program which replaces instances of a user-provided string in a file with another.

Mac
python3
Windows
python

```
$ python replace.py my_script.py fib fibonacci
def fibonacci(n):
  if n <= 1:
     return n
  return fibonacci(n - 1) + fibonacci(n - 2)
```

sys.argv[2]

sys.argv[1]

where `my_script.py` contains:

```
def fib(n):
 if n <= 1:
     return n
 return fib(n - 1) + fib(n - 2)
```

```
import sys
def replacer(filename, old, new):
    with open(filename, 'r') as f:
        for line in f:
            print(line.strip('\n').replace(old,new))
if __name__ == "__main__":
    replacer(sys.argv[1], sys.argv[2], sys.argv[3])
```

Question 2

Write out the contents of x and y after the following code executes.

```
x = [[1, 2], 3]
y = x[:] + [3] * 2
y[1] = 5
```

$y = [[1,2], 3] + \underbrace{[3] * 2}_{[3,3]}$

$y = [[1,2], 3, 3, 3]$

(0 1 2 3)

$y[1] = 5 \rightarrow y = [[1,2], 5, 3, 3]$

$x = [[1,2], 3]$

example to modify x via modifying y?

$y[0][1] = 3$

Question 3

For the following kinds of data, describe what data structure, e.g., list, set, dictionary, or tuple, would be the *most* appropriate to use.

1. Storing students in a class along with their grades.
2. Storing a shopping list optimized for efficient traversal of the supermarket.
3. Storing unique x, y coordinates.

dictionary

keys = student names
values = grades

list

order is important

set of tuples would be appropriate

set of lists? X → TypeError lists are mutable

```
>>> s = set()
>>> s.add([1,2])
Traceback (most recent call last):
  File "<python-input-1>", line 1, in <module>
    s.add([1,2])
    ~~~~~^^^^^^^
TypeError: unhashable type: 'list'
```

Question 4

In the following code

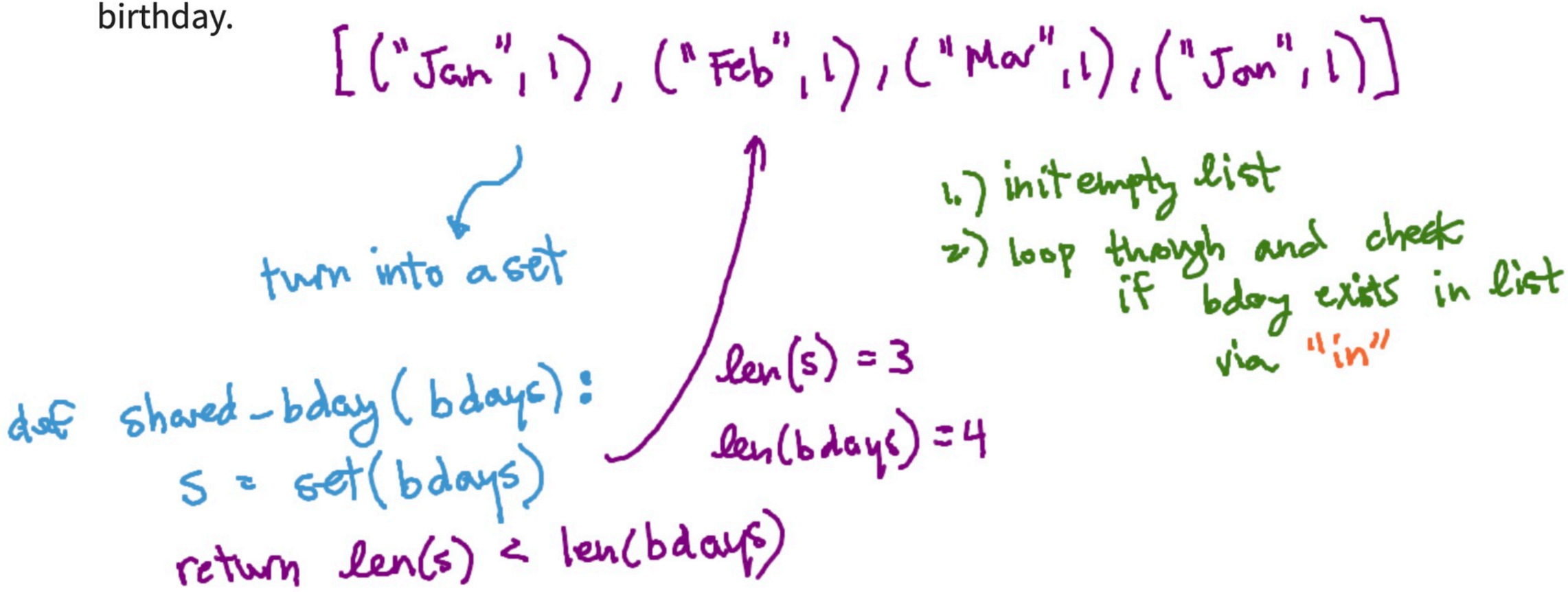
```
d = { 0: "0", 1: "I", 2: "II", 3: "III", 4: "IV", 5: "V" }
print(d[i])
```

which alternate definitions of `d` below would print the same for any value of `i` in 0-5, inclusive? *Select all that apply.*

1. `d = ["0", "I", "II", "III", "IV", "V"]`
2. `d = ("0", "I", "II", "III", "IV", "V")`
3. `d = {"0", "I", "II", "III", "IV", "V"}` X no indexing in set
4. `d = "0IIIIIIIVV"` X wrong result

Question 5

Write a function named `shared_bday` that takes a list of tuples representing birthdays, e.g., `("January",1)`, for a group of individuals and returns `True` if any share a birthday.



Question 6

Write a function that takes two parameters: a dictionary and a number. The function should update the dictionary by adding the number to each value in the dictionary.

```
def add_num(d, num):

    for key in d:
        d[key] += num
```

```
for value in dict.values():
    value += num
```

doesn't update values stored in dictionary

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> for value in d.values():
...     print(id(value))
...     value += 3
...     print(id(value))
...
4360710584
4360710680
4360710616
4360710712
4360710648
4360710744
>>> d
{'a': 1, 'b': 2, 'c': 3}
>>> for key in d:
...     d[key] += 3
...
>>> d
{'a': 4, 'b': 5, 'c': 6}
>>> for k,v in d.items():
...     d[k] = v + 1
...
>>> d
{'a': 5, 'b': 6, 'c': 7}
```

Question 7

What does the following function do (in one sentence) assuming x is a list:

```
def mystery(x):
    if len(x) <= 1:
        return True
    else:
        if x[0] < x[1]:
            return False
        else:
            return mystery(x[1:])
```

check if sorted in descending order.

Question 8

What is the shape drawn by the following function when invoked as `mystery(100,4)`, assuming the turtle starts at the origin facing to the right?

```
from turtle import *
def mystery(a, b):
    if b > 0:
        for i in range(3):
            forward(a)
            left(120)
        forward(a)
        mystery(a/2, b-1)
```

Question 9

Write a *recursive* function `all_upper` that takes a list of strings as a parameter and returns a list of booleans with `True` for strings in the list that are all uppercase, `False` otherwise. Recall that the string class has an `isupper` method that checks if it is all uppercase. For example:

```
>>> all_upper(["YES", "no"])
[True, False]
```

base case

recursive case

↳ approaches base case.

```
def all_upper(lst):
    if len(lst) == 0:
        return []
    else:
        return [lst[0].isupper()] + all_upper(lst[1:])
```

remember to use ()

indentation

Question 10

There are several problems with this recursive implementation of `fibonacci`. What are they? Recall that the Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13 ..., i.e. $F_n = F_{n-1} + F_{n-2}$ with $F_0 = 0$ and $F_1 = 1$.

```
def fibonacci(n):
 """ Return nth fibonacci number """
 if n == 0 or 1:
  return n
 else:
  fibonacci(n[1:]) + fibonacci(n[2:])
```

Question 11

We can compute the average without saving the values in memory by maintaining a "running" sum and count of the number of values observed. Implement a class `RunningAverage` that maintains a running average of values with an `add` method:

```
>>> mean = RunningAverage()
>>> for val in range(1, 5):
    mean.add(val)

>>> mean.average()
2.5
>>> mean.add(5)
>>> mean.average()
3.0
```

avg = (1 + 2 + 3) / 3 = num / den

```
class RunningAverage:
    def __init__(self):
        self.num = 0
        self.den = 0        # use count instead
    def average(self):      # check if self.den == 0 ?
        return self.num/self.den
```

```
def add(self, value):
    self.num += value
    self.den += 1
```

Question 12

It is also possible to compute a "running" variance using Welford's algorithm!

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)$$

$$\sigma_n^2 = \frac{M_{2,n}}{n}$$

where $M_{2,1} = 0$. Implement a class `RunningVariance` that derives from `RunningAverage` and computes the variance without storing all of the data.