15-112 F21 Practice Midterm 2

80 minutes

Note: This practice midterm was written by TAs. It may not reflect the exact difficulty or length of the actual midterm.

1. **Short Answer & Multiple Choice**

True or False: It is only possible to hash mutable types.

True or False: In a fractal, each recursive call should have a greater level than the current function call.

True or False: Binary search must be performed on a list that is sorted.

True or False: You can only have one base case in a recursive function.

Short Answer: Order the following big-Ohs from **slowest to fastest runtime**.

```
O(n), O(112), O(log(n**5)), O(2**(0.001 * n)), O(n**0.5)
```

Multiple Choice: Which of the following is true about inheritance?

A. An instance of the parent class can call a method defined in the child class.
B. A child class can override and add to the __init__ method of the parent class.
C. A child class is unable to overwrite an existing parent method.
D. A child class is not allowed to have any of the same methods as the parent class.

2. **Code Tracing**: Indicate what ct1, ct2, and ct3 print. Place your answers (and nothing else) in the boxes beside/below the code. **Assume all relevant imports have been made.** If you are asked to print a set or dictionary, you may print its keys/elements **in any order**.

```
class Cookie(object):
    def __init__(self, flavor):
        self.flavor, self.toppings = flavor, []
    def addTopping(self, topping):
        self.toppings.append(topping)
    def __repr__(self):
        return f'{self.flavor} cookie with {len(self.toppings)} toppings'
    def __eq__(self, other):
        return isinstance(other, Cookie) and repr(self) == repr(other)

class SmolCookie(Cookie):
    def addTopping(self, topping):
        if len(self.toppings) == 2: self.toppings.pop(0)
        super().addTopping(topping)

def ct1():
    c1 = Cookie('Chocolate')
    c2 = SmolCookie('Chocolate')
    print(c1)
    c1.addTopping('garlic')
    c2.addTopping('cheese')
    print(c1 == c2, c2 == c1)
    for i in range(10):
        c2.addTopping('mayo')
    print(c2.toppings)
ct1()
```

Write your answer for ct1 in the box below.

```
def ct2(L):
    s = set(L)
    d1 = {}
    d2 = dict()
    for elem in s:
        d1[elem] = L.count(elem)
    print(d1)
    for key in d1:
        if d1[key] in s:
            val = d1[key]
            d2[val] = d2.get(val, set())
            d2[val].add(key)
    return d2

print(ct2([1,1,1,2,4,4,5]))
```

Write your answer for `ct2` in the box below.

```
def ct3(L, s=1, depth=1):
    if L == []:
        result = 0
    elif len(L) == 1:
        result = L[0] * s
    else:
        mid = len(L) // 2
        if mid % 2 == 0:
            newS = s
        else:
            newS = -s
        a = ct3(L[:mid], s, depth+1)
        b = ct3(L[mid:], newS, depth+1)
        result = a + b
    print('*' * depth, result)
    return result

L = [15, 112, 10]
print(ct3(L))
```

Write your answer for ct3 in the box below.

3. **Free Response:** `valuesMatchingCounts(L)`

Write the function `valuesMatchingCounts` (vmc for short) which takes in a list of integers L and returns a set of all elements from L whose count in L is themself. For example, 1 would be in the set if it appears 1 time in L, 3 would be in the set if it appears 3 times in L, etc.

This function must run in O(n), where n is the length of L.

Below are a few examples:

```
assert(vmc([1, 2, 3, 4, 5]) == {1})
assert(vmc([1, 5, 1, 1, 2]) == set())
assert(vmc([5, 2, 5, 2, 5, 6, 5, 1, 5, 0]) == {1, 2, 5})
```

4. **Free Response:** File System Classes

Write the `File`, `PythonFile`, and `Folder` classes such that the following test cases pass. You **must use inheritance** to receive full credit.

```
def testFileFolder():
    print("Testing File, Python File and Folder...", end="")
    a = File("A", 15)
    b = File("B", 1)
    c = File("C", 12)

    assert(a.getSize() == 15)
    assert(a == File("A", 15))
    assert(a != File("A", 16))
    assert(a != File("B", 15))
    assert(a != "Don't Crash Here")

    assert(str([b, c]) == "[File('B', 1), File('C', 12)]")

    # Hint: do not write more methods than you need to...
    h = PythonFile("Hi", 100)
    assert(isinstance(h, File))
    assert(not isinstance(a, PythonFile))
    assert(h.getSize() == 100)
    assert(repr(h) == "PythonFile('Hi', 100)”)
    # Running a Python file always outputs the string "Hello World!"
    assert(h.runFile() == "Hello World!")

    # These work just like actual files and folders...
    F0 = Folder([])
    F1 = Folder([a, F0])
    F2 = Folder([F1])
    F3 = Folder([b, h])
    F4 = Folder([F2, c, F3])

    # Hint: this method is recursive...
    assert(F0.getFileNames() == set())
    assert(F1.getFileNames() == {"A"})
    assert(F2.getFileNames() == {"A"})
    assert(F3.getFileNames() == {"B", "Hi"})
    assert(F4.getFileNames() == {"A", "B", "C", "Hi"})
    print("Passed!")
```

You may write your File System Classes solution on this page.

5. **Free Response:** `onlyPrimes(L)`

Write a **recursive** function `onlyPrimes` which takes in a list of integers L and returns a list consisting of only the elements of L that are prime (in the order that they appear in L).

Do not use loops, strings, sets, dictionaries, or O(n) list functions or methods (no comprehensions, min, max, sorted, etc). List indexing and slicing are allowed.

Note that **isPrime must also be written recursively**. As a hint, you may want to write a recursive helper function that checks factors one by one, starting from 2, until it reaches some max factor.

Below are a few examples:

```
assert(onlyPrimes([]) == [])
assert(onlyPrimes([1, 5, 1, 1, 2]) == [5, 2])
assert(onlyPrimes([1, 2, 3, 4, 5, 6, 7, 8, 9]) == [2, 3, 5, 7])
```

6. **Free Response:** `kSuperSplit(L, k, n)`

Write the backtracking function `kSuperSplit` which takes in a list of positive integers L, a positive integer k, and a positive integer n, and partitions L into k 1D lists (each element of L is present in exactly one of the k lists) such that the sum of each 1D list is at most n. For example if we are given:

```
L = [1, 5, 1, 1, 2, 3, 4, 5]
k = 3
n = 8
```

then `kSuperSplit` needs to return 3 1D lists, each with a sum of at most 8, so the following is one possible solution:

```
[[1, 5, 1, 1],
 [3, 5],
 [2, 4]]
```

However, if `k = 6` and `n = 4` there is no possible solution, so kSuperSplit would return `None`.