CSCI 121: Practice Final Exam

Exam: 9am-12pm, Wednesday, May 13th, Eliot 314

Spring 2026

The next pages give practice problems for the final exam being held next week. The exam is comprehensive and covers these topics:

- scripting with `input` and `print`
- variables and assignment
- integer arithmetic, boolean connectives, integer comparisons
- strings and string operations
- integer division using `%` and `//`
- printing versus returning, the `None` type
- conditional statements and loops
- function definitions
- recursive functions
- higher-order functions and `lambda`
- lists and dictionaries
- object-orientation and inheritance
- linked lists and binary search trees
- sorting and searching, efficiency and running time
- BONUS PROBLEM: file I/O and exceptions

You can use these to test your knowledge in preparation for taking the exam

1. Define a function `maxDigit` that takes a list of integers. It should find the largest digit found amongst all of the integers of that list. If the list is empty, it should return `None`.

```
>>> maxDigit([123, 45, 854, 78, 0])
8
>>> maxDigit([1111, 11, 1, 1117111, 6])
7
>>> print(maxDigit([]))
None
```

2. Define a function `unzipDict` that takes a dictionary as a parameter. It should return a list of all its keys along with a list of all their associated values. These should be returned as a list of two lists of the same length. The first key should be what was associated with the first value, the second key should be what was associated with the second value, and so on. For example:

```
>>> unzipDict({'abe': 13, 'bob': 72, 'cat':13, 'dog': 1})
[['abe', 'bob', 'cat', 'dog'], [13, 72, 13, 1]]
>>> unzipDict({})
[[], []]
```

3. Write a Python script that takes a time period in seconds. It should report that time in hours, minutes, and seconds. It should use as many hours as possible to express it, and it should not report more than 59 seconds. A time unit's word should not be plural when there is only one of it. It should work as shown below:

```
% python3 time.py
Enter a time period in seconds: 7213
2 hours 0 minutes 13 seconds

% python3 time.py
Enter a time period in seconds: 3661
1 hour 1 minute 1 second

% python3 time.py
Enter a time period in seconds: 121
0 hours 2 minutes 1 second
```

4. Below are the definitions of two classes for a linked list:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None
class LinkedList:
    def __init__(self):
        self.first = None
    def prepend(self, value):
        newNode = Node(value)
        newNode.next = self.first
        self.first = newNode
    def output(self):
        current = self.first
        while current is not None:
            print(current.value)
            current = current.next
```

Write a linked list method `appendSum`. It should append a new node at the end whose value is the sum of the prior nodes' values. If the list is empty it should append a 0. For example:

```
>>> ll = LinkedList()
>>> ll.appendSum()
>>> ll.output()
0
>>> ll.prepend(3); ll.prepend(12)
>>> ll.output()
12
3
0
>>> ll.appendSum()
>>> ll.output()
12
3
0
15
```

5. Write a Python function `makeStringFormatter` that takes a single parameter `s` that is expected to be a string. It should then return a function. That function should, when given a positive integer `n`, return a string of length `n`. If the length of `s` is smaller than `n` then it returns the prefix of `s` of that length. If the length of `s` is larger than `n` then it returns `s` followed by enough spaces to be length `n`.

```
>>> f = makeStringFormatter("hello")
>>> f(3)
'hel'
>>> f(7)
'hello  '
>>> f(5)
'hello'
```

6. Write the code for a class called `Socialite`. Socialites have a first name and an email address, and the email address of every socialite is unique. These are both represented as text strings. When they meet another socialite, they each exchange their information and add it to their collection of contacts. This happens with their method `meet`. They only add that contact if they haven't already met that socialite. You can ask a socialite to tell you all their contacts with the `contacts` method. Below is a Python interaction with some socialite objects.

```
>>> a = Socialite('abe', 'abe@abc.com')
>>> b = Socialite('bob', 'bob@blahblah.biz')
>>> d = Socialite('dog', 'woof@gmail.com')
>>> d.meet(a)
>>> a.meet(b)
>>> a.contacts()
bob@blahblah.biz bob
woof@gmail.com dog
>>> b.meet(a)
>>> a.contacts()
bob@blahblah.biz bob
woof@gmail.com dog
```

Note that `s1.meet(s2)` makes both `s1` and `s2` have each other as a contact.

7. Below is the code for a `Animal` class that could be used in a zoo simulation. (You can imagine that the code for the larger simulation calls the `update` method at every time step.)

```
class Animal:

    def __init__(self, id, species):
        self.id = id
        self.species = species
        self.hunger = 0
        self.energy = 100
        self.asleep = False

    def update(self):
        if self.asleep:
            self.energy += 1
            if self.energy >= 100:
                self.asleep = False
        else:
            self.energy -= 1
            if self.energy <= 0:
                self.asleep = True

    def eat(self, amount):
        if self.asleep:
            self.asleep = False
        self.hunger = max(0,self.hinger - amount)
```

Define a new class `Elephant` to represent elephants. Its species is always `'elephant'` so its constructor doesn't take that information. Unlike many animals, elephant don't wake up to eat, so its `eat` method should do nothing if an elephant is asleep. Otherwise, elephants behave like any other animals.

8. Give the asymptotic running time for each of the following functions that take n as a parameter. Write this using Theta notation, fully simplified, e.g., $\Theta(n^2)$, $\Theta(n \log(n))$, etc. Note that some of the later functions call the earlier ones.

(a)
```
def sum_odds(n):
    if n == 0:
        return 0
    elif n % 2 == 1:
        return n + sum_odds(n-1)
    else:
        return sum_odds(n-1)
```

(b)
```
def sum_sums(n):
    total = 0
    while n >= 1:
        total += sum_odds(n)
        n -= 2
    return total
```

(c)
```
def do_the_sum_sums(n):
    i = 0
    while i < n:
        sum_sums(n)
        i += 1
```

(d)
```
def halves_a_lot(n):
    i = n
    while i > 0:
        j = n
        while j > 0:
            j = j // 2
        i -= 1
```

9. Write the function `firstMissing` which takes a sorted list of positive integers with no repeated values. It should return the smallest positive integer missing from the list. **Your solution must run in $\Theta(\log_2(n))$ time, where n is the length of the list.**

   For example:

```
>>> firstMissing([1, 2, 3, 5, 6, 7, 8])
4
>>> firstMissing([2, 3, 4, 5, 6, 7, 8])
1
>>> firstMissing([1, 2, 3, 4, 5, 6, 7])
8
```

10. Below is the start of the code that defines a binary search tree like from lecture.

```
class BSTNode:
    def __init__(self,key):
        self.key = key
        self.left = None
        self.right = None
class BSTree:
    def __init__(self):
        self.root = None
    def insert(self, k):
        p = None
        n = self.root
        while n is not None:
            p = n
            if n.key == k:
                return
            elif n.key > k:
                n = n.left
            else:
                n = n.right
        e = BSTNode(k)
        if p is None:
            self.root = e
        elif p.key > k:
            p.left = e
        else:
            p.right = e
```

Write a method for `BSTree` named `maxResult`. It should take a function `f` that can be applied to each of the nodes' keys. It should return the largest value of `f(x.key)` over all the nodes `x` of the key. If the tree is empty, it should return `None`.

BONUS Write the code for a function `firstError`. The function takes two arguments: a function `f` and an integer `n`. It should attempt to run the function `f` on the input 0, then 1, then 2, and so on up to and including `n`. If any of those lead to an error, it should return the smallest such input. If none lead to an error, it should return `None`. For example:

```
>>> firstError(lambda x: 99 / x, 6)
0
>>> firstError(lambda x: 99 / (x - 2), 6)
2
>>> print(firstError(lambda x: 99 / (x + 1), 6))
None
>>> a = [100,77,86]
>>> firstError(lambda i: a[i], 6)
3
```