Programming and Data Structures using Python 25 March 2021
Final Exam, Dec 2020–Mar 2021 Marks: 65, Weightage: 40%

1. In the code fragment below, `start` and `end` are integer values and `square(x)` is a function that returns `True` if `x` is a perfect square and `False` otherwise.

```
(i,j,k) = (0,0,0)
for m in range(start,end):
  if square(m):
     i = i - 1
     k = k - 1
  else:
     j = j + 1
     k = k - 1
```

Provide an expression connecting the values `i`, `j` and `k` that is invariant after each iteration of the `for` loop. (For example, such an expression could be `i+k == 2*j`.) Explain why your expression is a loop invariant. *(5 marks)*

2. Consider the following Python code.

```
def f():
  global l3,x2
  l1 = l3 + [74]
  l2.append(80)
  l3 = l3 + [90]
  x1 = x2 + 31
  x2 = x2 + 42

(l1,l2,l3) = ([73],[82],[91])
(x1,x2) = (20,21)
f()
```

What are the values of `l1`, `l2`, `l3`, `x1`, and `x2` at the end of the execution? Explain your answer. *(5 marks)*

3. We have an incomplete Python function definition, along with its behaviour in the Python intepreter, as described below.

```
def f(.........):
  print("a",a,"b",b,"c",c,"d",d)

>>> f(d=4,a=3)
a 3 b 20 c 35 d 4

>>> f(3,5,7)
a 5 b 7 c 35 d 3

>>> f(3,b=7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() missing 1 required positional argument: 'a'
```

What was the original definition of the function `f`? Explain your answer. *(5 marks)*

4. Consider the min-heap $[17, 21, 38, 39, 76, 89, 54, 43, 42]$ built by repeatedly inserting values into an empty heap. Identify all the values that could have been the last element inserted into this heap. Explain your answer. *(5 marks)*

5. Suppose we add this function `foo()` to a class `Tree` that implements binary trees. For an object `t` of type `Tree`, the attributes `t.value`, `t.left` and `t.right`, and the functions `t.isempty()` and `t.isleaf()` have the usual interpretation. Given an object `mytree` of type `Tree`, what would `mytree.foo()` compute? Explain your answer.

```
def foo(self):
  if self.isempty():
    return(0)
  elif self.isleaf():
    return(1)
  else:
    return(1 + max(self.left.foo(), self.right.foo()))
```

(5 marks)

6. Consider a plate stacked with several disks, each of a different diameter (they could all be, for instance, *dosas* or *chapatis* of different sizes). We want to sort these disks in decreasing order according to their diameter so that the widest disk is at the bottom of the pile. The only operation available for manipulating the disks is to pick up a stack of them from the top of the pile and invert that stack. (This corresponds to lifting up a stack *dosas* or *chapatis* between two big spoons and flipping the stack.)

   Assume the stack of disks is represented as a list `A` that stores the diameters of the disks, where `A[0]` is the bottom disk and `A[len(A)-1]` is the top disk. The function `flip(A,i)` flips the stack from `A[i]` to `A[len(A)-1]`.

```
def flip(A,i):
   A[i:] = reversed(A[i:])
   return
```

   Write a Python function `dosasort` that takes as input a list `A` of integers and sorts it in descending order using only the function `flip` to rearrange elements in the list.

(10 marks)

7. Consider user-defined lists in Python built up from the basic class `Node` defined on the right. As discussed in class, an empty list is represented by a `Node` with both `value` and `next` set to `None`.

   Add a function `mymap` to the class that takes a function `f` and transforms each value `v` in the list to `f(v)`.

```
class Node:
  def __init__(self,initval=None):
    self.value = initval
    self.next = None
    return

  def mymap(self,f):
    # To be written by you
```

(10 marks)

8. Write a Python class `DSet` that implements a set of values. (These can be any immutable values, but to be concrete you can assume the values are integers.) Internally, use a dictionary to represent the set through its keys.

   (a) The constructor should take a list of values and convert it into a set. Remember that a set has no duplicate values. If no list is provided, the constructor should create an empty set.

   (b) The class should have functions `size`, `member`, `union` and `intersection` with the expected interpretation.

   - Given a set `S`, `S.size()` returns the number of elements in `S`.
   - Given a set `S` and a value `x`, `S.member(x)` returns `True` if `x` belongs to `S` and `False` otherwise.
   - Given sets `S` and `T`, `S.union(T)` and `S.intersection(T)` should return new sets corresponding to $S \cup T$ and $S \cap T$, respectively.

   (10 marks)

9. An *alignment* between two strings w_1 and w_2 (over the alphabet $\{a, b, c, \ldots, z\}$) is obtained by inserting hyphens in the two strings such that the modified strings *align* (i.e., the modified strings are of equal length, and at each position, either both strings have the same letter or one of the strings has a hyphen).

   Here are three examples of alignments. The first is between `ocurrance` and `occurrence` and the second and third are between `ctatg` and `ttaagc`.

   (1)
   ```
   oc-urr-ance
   occurre-nce
   ```
   (2)
   ```
   ct-at-g-
   -tta-agc
   ```
   (3)
   ```
   ctat---g-
   ---ttaagc
   ```

   A *mismatch* in an alignment is a position where one of modified strings has a hyphen and the other does not. There are three mismatches in the first alignment given above, five mismatches in the second, and seven mismatches in the third. The best alignment is the one with the fewest mismatches.

   Let `s` and `t` be the two strings to be matched, of length m and n, respectively. Let `M[i][j]` be the number of mismatches in the best alignment of `s[:i]` and `t[:j]`

   (a) Write a recursive formulation of `M[i][j]`.

   (b) Write Python code to compute `M[m][n]` using dynamic programming.

   (10 marks)