DAT516 Exam Instructions

Write your answers in Inspera. You can of course use separate sheets of paper to draft and experiment, but only the answers in Inspera will be graded. The answers can and should be short.

Python interpreters such as Self Practice are not available. Hence you have to reason about the answers on your own. Using pencil and paper is a good aid for this.

You can also have with you an A4 paper, hand-written on both sides, with whatever information you want to have available

You will need 24 points out of 60 in this exam to get accepted with grade 3, 36 points for grade 4, and 48 points for grade 5. The exam grade will also be the final grade for the course.

If used as a re-exam for DAT515 or DIT515 from earlier years, the final grade of the course will be the grade from your labs, provided that you get at least 24 points in this exam.

Python Syntax

For your reference: the syntax of (the relevant parts of) Python

```
<stm> ::= <decorator>* class <name> (<name>,*)?: <block>
    | <decorator>* def <name> (<arg>,*): <block>
    | import <name> <asname>?
    | from <name> import <imports>
    | <exp>,* = <exp>,*
    | <exp> <assignop> <exp>
    | for <name> in <exp>: <block>
    | <exp>
    | return <exp>,*
    | yield <exp>,*
    | if <exp>: <block> <elses>?
    | while <exp>: <block>
    | pass
    | break
    | continue
    | try: <block> <except>* <elses> <finally>?
    | assert <exp> ,<exp>?
    | raise <name>
    | with <exp> as <name>: <block>
<decorator> ::= @ <exp>
<asname>    ::= as <name>
imports     ::= * | <name>,*
<elses>     ::= <elif>* else: <block>
<elif>      ::= elif exp: <block>
<except>    ::= except <name>: <block>
<finally>   ::= finally: <block>
<block>     ::= <stm> <stm>*
<exp> ::= <exp> <op> <exp>
    | <name>.?<name>(<arg>,*)
    | <literal>
    | <name>
    | ( <exp>,* )
    | [ <exp>,* ]
    | { <exp>,* }
    | <exp>[exp]
    | <exp>[<slice>,*]
    | lambda <name>*: <exp>
    | { <keyvalue>,* }
    | ( <exp> for <name> in <exp> <cond>? )
    | [ <exp> for <name> in <exp> <cond>? ]
    | { <exp> for <name> in <exp> <cond>? }
    | { <exp>: <exp> for <name> in <exp> <cond>? }
    | - <exp>
```

```
    |  ~ <exp>
    |  not <exp>
<keyvalue> ::= <exp>: <exp>
<arg>      ::= <name>
        |  <name> = <exp>
 |  *<name>
 |  **<name>
<cond> ::= if <exp>
<op>   ::= + | - | * | ** | / | // | % | @ | == | > | >= | < | <= | != | in | not in | and | or
     | & | | | ^ | << | >> | :=
<assignop> ::= += | -= | *= | /= | %= | &= | |= | ^=
<slice> ::= <exp>? :<exp>? <step>?
<step>  ::= :<exp>?
```

1 Question 1

Question 1 (20 p). What is the value and its type of the following expressions? Remember that None is also a value! It can also happen that the expression contains an error. In that case, indicate the kind of error, one of SyntaxError, TypeError, AttributeError, ZeroDivisionError, KeyError, NameError, IndexError, and explain (in your own words) the reason of the error. (1+1 p for each part)

[1, 2, 3].append(3)

(Value and Type) or (Error and Explanation)

len({1, 2, 3}.add(3))

(Value and Type) or (Error and Explanation)

set('python') == set('typhoon')

(Value and Type) or (Error and Explanation)

[n*'2' for n in range(4)]

(Value and Type) or (Error and Explanation)

{x % 3 for x in range(20)}

(Value and Type) or (Error and Explanation)

[] is []

(Value and Type) or (Error and Explanation)

print((lambda x, y: x + y)(5, 6))

(Value and Type) or (Error and Explanation)

(lambda x, y: x + y)(5, 6)

(Value and Type) or (Error and Explanation)

len({x%2: x for x in range(100)})

(Value and Type) or (Error and Explanation)

{x: x ** x for x in range(4)}.get(4, 63)

(Value and Type) or (Error and Explanation)

Totalpoäng: 20

2 Question 2

Question 2 (12 p). In Lab 1 of this course, we built dictionaries of tram stops, tram lines, and transition times. They were collected into a single dictionary of the following shape:

```
tramnetwork = {
   "stops": {
      "Östra Sjukhuset": {
         "lat": 57.7224618,
         "lon": 12.0478166
      },
   # the positions of every stop
   },
   "lines": {
      "1": [
         "Östra Sjukhuset",
         "Tingvallsvägen",
         # and the rest of the stops along line 1
         ],
   # the sequence of stops on every line
   },
   "times": {
      "Östra Sjukhuset": {},
      "Tingvallsvägen": {
         "Östra Sjukhuset": 1
      },
   # the times from each stop to its alphabetically later neighbours
   }
 }
```

Using this definition, write a Python expression whose value is the set of those line numbers where the line passes through "Chalmers" but not through "Korsvägen". All line and stop names are not shown in the fragment above, but you can assume that appear in the complete data. Notice: the answer must be an expression, not a statement or a sequence of statements. (5p)

Using the same definition, write a Python expression whose value is the pair of neighbouring stops that have the longest transition time between them. If there are many such pairs, it can return any of them. Notice: the answer must be an expression, not a statement or a sequence of statements. (7p)

Totalpoäng: 12

3 Question 3

Question 3 (14 p). The following class defines undirected graphs in a way that often used in the mathematical theory of graphs: the internal representation consits of a set of vertices and a set of edges. All vertices that appear in the edge set are also included in the vertex set, but the vertex set can also contain isolated vertices, ones that have no edges to other vertices. The adjacency dictionary is obtained with a getter method from these internal representations. There is also a method of merging graphs, to be defined as the last point of this question.

```
class Graph:

    def __init__(self):
        self.vertices = set()
        self.edges = set()

    def add_vertex(self, a):
        self.vertices.add(a)

    def add_edge(self, a, b):
        a, b = sorted((a, b))
        self.edges.add((a, b))

    def adjacency(self):
        return {a: {b for c, b in self.edges if c == a} for a in self.vertices}

    def merge_graph(self, other):
        # add the vertices and edges of the other graph to self
```

Using the above definition, we build a graph as follows:

```
G = Graph()
G.add_edge(2, 1)
G.add_edge(1, 2)
G.add_vertex(3)
print(G.adjacency())
```

What is printed when we ask for the adjacency list? Watch out: the result is not what we want, because there is a bug in the code. So start by following the class definition strictly and print exactly what it gives you. (3p)
print(G.adjacency())

Now, write what you would expect the adjacency dictionary to be. (2p)
corrected adjacency dict:

Now, write a corrected version of the merhod or methods that the error results from. (6p)
corrected methods:

As the last item in this question, implement the merge_graphs method. It should add to self those vertices and edges of other that are not yet in self. The definition should be as simple as possible; too much code may lead to point reductions. (3p)
Write the body of the merge_graph() method.

Totalpoäng: 14

4 Question 4

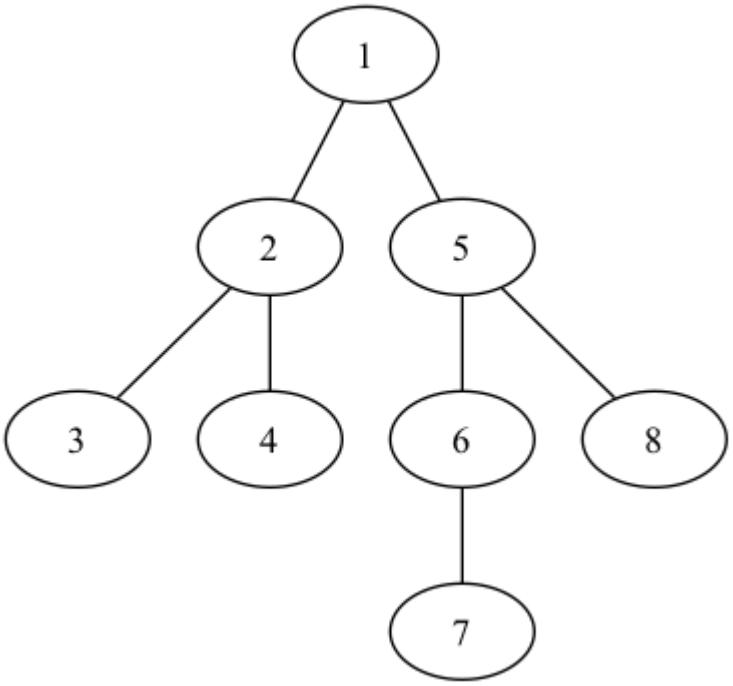
Question 4 (14 p). Trees are a special case of graph, but they can also be defined independently as the following class:

```
class Tree:
    def __init__(self, node, trees):
        self.root = node
        self.subtrees = trees
```

The idea is that Tree is a recursive data structure: a tree consists of a root node and a list of subtrees. A tree that has no subtrees but only the root node can be concisely constructed with the following function:

```
def atom(a):
    return Tree(a, [])
```

Your first task is to write an expression that constructs the following tree. You can use both Tree() and atom(). (4p)



Write your answer here:

mytree =

Your next task is to define the function that converts a Tree in the sense of this question to a Graph in the sense of Question 3. Hint: the simplest way is to use a recursive function that also uses merge_graph() from Question 3. But other solutions can also be accepted. (7p)

```
def tree2graph(tree):
    # convert a Tree as in Question 4 to a Graph as in Question 3
```

write the body of the function here

Finally, show the adjacency dictionary of mytree above obtained via tree2graph() and Graph.adjacency(). You can assume that the bug in Question 3 has been fixed. You can get 3 points for this even if you have not fixed the bug yourself and even if your tree2graph() function is missing or wrong. (3p)

tree2graph(mytree).adjacency()

Totalpoäng: 14