Lab 11.1 : Sample lab exam (Deadline Monday 28 March 23:59)

Before starting

- The exam runs 1400-1550.
- Answer all questions.
- Upload all code to Einstein.
- All lab exam rules apply.

Triathlete part 1 [10 marks]

- In module `triathlete_v1_111.py` define a `Triathlete` class to model a triathlete.
- A triathlete has a name and ID number.
- When your class is correctly implemented, running the following program should produce the given output.

```
from triathlete_v1_111 import Triathlete

def main():
    t1 = Triathlete('Ian Brown', 21)
    t2 = Triathlete('John Squire', 22)

    assert(t1.name == 'Ian Brown')
    assert(t1.tid == 21)

    print(t1)
    print(t2)

if __name__ == '__main__':
    main()
```

```
Name: Ian Brown
ID: 21
Name: John Squire
ID: 22
```

Triathlete part 2 [10 Marks]

- In module `triathlete_v2_111.py` extend the `Triathlete` class to support the recording of per-discipline times for a triathlete.
- Disciplines are swimming, cycling and running and times are recorded in seconds.
- When your class is correctly implemented, running the following program should produce the given output.

```
from triathlete_v2_111 import Triathlete
```

```
def main():

    t1 = Triathlete('Ian Brown', 21)

    t1.add_time('swim', 100)
    t1.add_time('cycle', 120)
    t1.add_time('run', 150)

    print('Cycle: {}'.format(t1.get_time('cycle')))
    print(t1)

if __name__ == '__main__':
    main()
```

```
Cycle: 120
Name: Ian Brown
ID: 21
Race time: 370
```

Triathlete part 3 [15 Marks]

- In module *triathlete_v3_111.py* extend the `Triathlete` class to support the comparison of triathletes.
- Comparison is carried out in terms of a triathlete's race time.
- When your class is correctly implemented, running the following program should produce the given output.

```
from triathlete_v3_111 import Triathlete

def main():

    t1 = Triathlete('Ian Brown', 21)
    t2 = Triathlete('John Squire', 22)
    t3 = Triathlete('Alan Wren', 23)

    t1.add_time('swim', 100)
    t1.add_time('cycle', 120)
    t1.add_time('run', 150)

    t2.add_time('swim', 300)
    t2.add_time('cycle', 100)
    t2.add_time('run', 200)

    t3.add_time('swim', 150)
    t3.add_time('cycle', 120)
    t3.add_time('run', 100)

    print(t1)
    print(t2)
    print(t3)

    assert(t1 == t3)
    assert(t1 < t2)
    assert(t2 > t3)

if __name__ == '__main__':
    main()
```

```
Name: Ian Brown
ID: 21
Race time: 370
Name: John Squire
ID: 22
Race time: 600
Name: Alan Wren
ID: 23
Race time: 370
```

Triathlon part 1 [15 Marks]

- In module *triathlon_v1_111.py* define a `Triathlon` class to model a collection of triathletes.
- A triathlon is essentially a mapping from triathlete IDs to `Triathlete` objects.
- **You must include in triathlon_v1_111.py a copy of your Triathlete class definition from triathlete_v1_111.py.**
- Triathletes can be added to and removed from the triathlon via the `add()` and `remove()` methods respectively.
- A `lookup()` method returns a `Triathlete` object if a given triathlete is in the triathlon and `None` otherwise.
- When your class is correctly implemented, running the following program should produce no output.

```
from triathlon_v1_111 import Triathlete, Triathlon

def main():

    tn = Triathlon()
    t1 = Triathlete('Ian Brown', 21)
    t2 = Triathlete('John Squire', 22)

    tn.add(t1)
    tn.add(t2)

    t = tn.lookup(21)
    assert(isinstance(t, Triathlete))
    assert(t.name == 'Ian Brown')
    assert(t.tid == 21)

    tn.remove(21)
    t = tn.lookup(21)
    assert(t is None)

if __name__ == '__main__':
    main()
```

Triathlon part 2 [15 Marks]

- In module *triathlon_v2_111.py* extend the `Triathlon` class to support the printing of a triathlon.
- Printing a triathlon prints all triathlete details in alphabetical order of their names.
- **You must include in triathlon_v2_111.py a copy of your Triathlete class definition from triathlete_v1_111.py.**

- When your class is correctly implemented, running the following program should produce the given output.

```
from triathlon_v2_111 import Triathlete, Triathlon

def main():

    tn = Triathlon()
    t1 = Triathlete('Ian Brown', 21)
    t2 = Triathlete('John Squire', 22)
    t3 = Triathlete('Alan Wren', 23)

    tn.add(t1)
    tn.add(t2)
    tn.add(t3)

    print(tn)

if __name__ == '__main__':
    main()
```

```
Name: Alan Wren
ID: 23
Name: Ian Brown
ID: 21
Name: John Squire
ID: 22
```

Triathlon part 3 [15 Marks]

- In module *triathlon_v3_111.py* extend the `Triathlon` class to support retrieval of the Triathletes with the best and worst race times.
- **You must include in triathlon_v3_111.py a copy of your Triathlete class definition from Triathlete_v3_111.py.**
- When your class is correctly implemented, running the following program should produce the given output.

```
from triathlon_v3_111 import Triathlete, Triathlon

def main():

    tn = Triathlon()
    t1 = Triathlete('Ian Brown', 21)
    t2 = Triathlete('John Squire', 22)
    t3 = Triathlete('Alan Wren', 23)

    t1.add_time('swim', 100)
    t1.add_time('cycle', 120)
    t1.add_time('run', 150)

    t2.add_time('swim', 300)
    t2.add_time('cycle', 100)
    t2.add_time('run', 200)

    t3.add_time('swim', 50)
    t3.add_time('cycle', 20)
    t3.add_time('run', 10)
```

```
    tn.add(t1)
    tn.add(t2)
    tn.add(t3)

    print(tn.best())
    print(tn.worst())

if __name__ == '__main__':
    main()
```

```
Name: Alan Wren
ID: 23
Race time: 80
Name: John Squire
ID: 22
Race time: 600
```

Graph search [20 Marks]

- Below are the current contents of module *graph_111.py*.

```
class Graph(object):

    def __init__(self, V):
        self.adj = {}
        self.V = V
        for v in range(V):
            self.adj[v] = list()

    def addEdge(self, v, w):
        self.adj[v].append(w)
        self.adj[w].append(v)

class DFSPaths(object):

    def __init__(self, g, s):
        self.g = g
        self.s = s
        self.visited = [False for _ in range(g.V)]
        self.parent = [False for _ in range(g.V)]
        self.dfs(s)

    def dfs(self, v):
        self.visited[v] = True
        for w in self.g.adj[v]:
            if not self.visited[w]:
                self.parent[w] = v
                self.dfs(w)

    def hasPathTo(self, v):
        pass

    def pathTo(self, v):
        pass
```

- Complete the `hasPathTo()` and `pathTo()` methods.

- The contents of `graph01.txt` are as follows.

```
$ cat graph01.txt
7
0 1
0 2
0 5
0 6
3 4
3 5
4 5
4 6
```

- When your class is correctly implemented, running the following program should produce the given output.

```
#!/usr/bin/env python3

import sys

from graph_111 import Graph, DFSPaths

def main():

    with open('graph01.txt') as f:

        V = int(f.readline())

        g = Graph(V)

        for line in f:

            v, w = [int(t) for t in line.strip().split()]
            g.addEdge(v, w)

    paths = DFSPaths(g, 0)

    print(paths.hasPathTo(6))
    print(paths.pathTo(6))

if __name__ == '__main__':
    main()
```

```
True
[0, 5, 3, 4, 6]
```