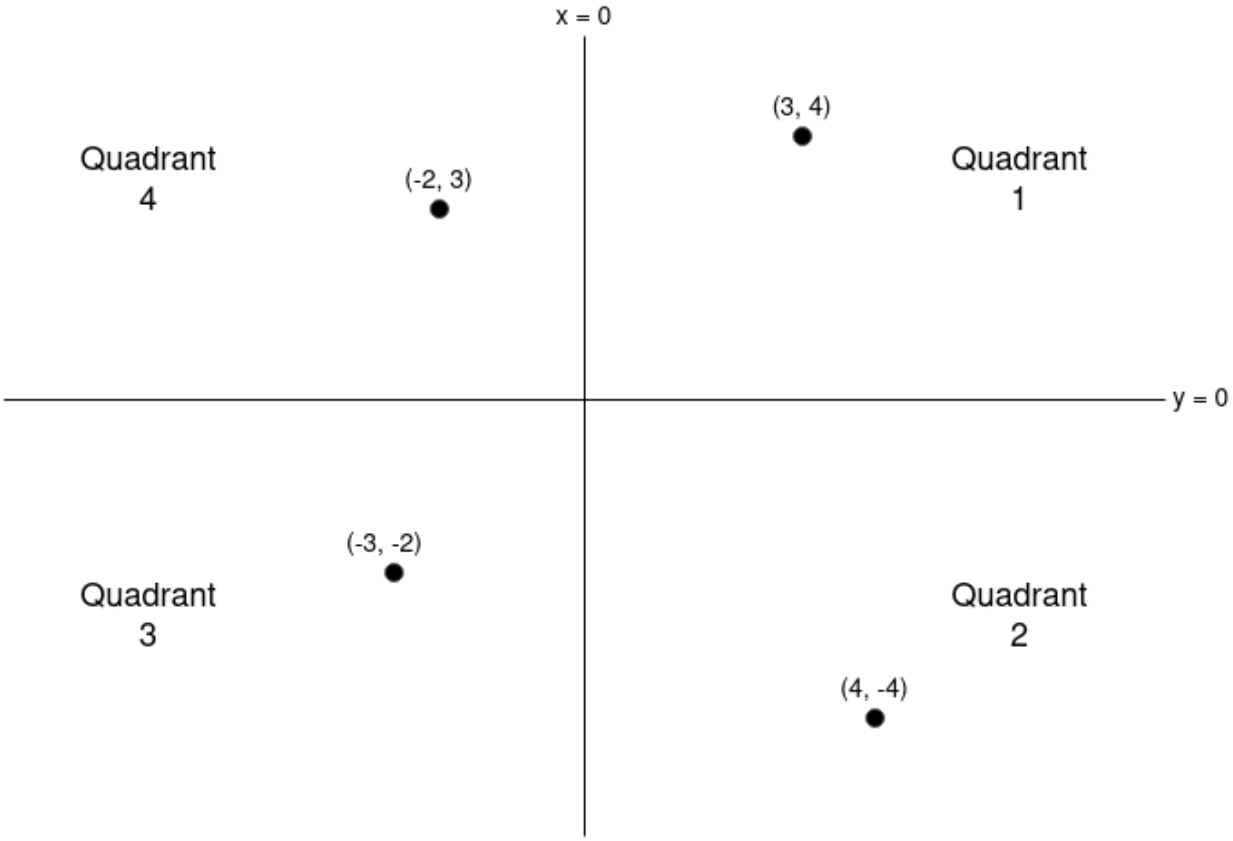
CA117 SAMPLE FINAL EXAM

Before starting

- The exam runs 0930-1230.
- Answer all questions.
- Upload all code to Einstein.
- All lab exam rules apply.

Quadrants [10 marks]

- Two-dimensional space can be divided into four quadrants as illustrated below.



- Write a program called *quadrants_131.py* that reads lines of text from `stdin` (the number of lines is arbitrary).
- Each line consists of two integers, x and y, representing the x and y coordinates of a point in two dimensions.
- The co-ordinates of four sample points are provided in the diagram above.
- You can assume that neither x nor y are ever equal to zero.
- For each point read the program should print out the number of the quadrant it belongs to.
- For example:

```
$ cat quadrants_stdin_00_131.txt
3 4
-3 -2
```

```
$ python3 quadrants_131.py < quadrants_stdin_00_131.txt
1
3
```

MP3Track part 1 [8 marks]

- In module *mp3track_v1_131.py* define an `MP3Track` class to model an MP3 audio track.
- An MP3 track has a title and a duration (supplied in seconds).
- When your class is correctly implemented, running the following program should produce the given output.

```
from mp3track_v1_131 import MP3Track

def main():
    t1 = MP3Track('Fools Gold', 604)
    t2 = MP3Track('Shallow', 197)
    t3 = MP3Track('Telephone', 220)

    assert(t1.title == 'Fools Gold')
    assert(t1.duration == 604)

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

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

```
Title: Fools Gold
Duration: 604
Title: Shallow
Duration: 197
Title: Telephone
Duration: 220
```

MP3Track part 2 [8 marks]

- In module *mp3track_v2_131.py* extend the `MP3Track` class such that one or more artists can be associated with a track.
- When your class is correctly implemented, running the following program should produce the given output (artists are printed in the order they are added to a track).

```
from mp3track_v2_131 import MP3Track

def main():
    t1 = MP3Track('Fools Gold', 604, ['The Stone Roses'])
    t2 = MP3Track('Shallow', 197, ['Lady Gaga', 'Bradley Cooper'])
    t3 = MP3Track('Telephone', 220, ['Beyonce', 'Lady Gaga'])

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

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

```
Title: Fools Gold
By: The Stone Roses
Duration: 604
Title: Shallow
By: Lady Gaga, Bradley Cooper
Duration: 197
Title: Telephone
By: Beyonce, Lady Gaga
Duration: 220
```

MP3Track part 3 [8 marks]

- In module *mp3track_v3_131.py* extend the `MP3Track` class such that artists can be associated with a track either at the time of `MP3Track` object creation or subsequently by invoking the `add_artist()` method.
- When your class is correctly implemented, running the following program should produce the given output.

```
from mp3track_v3_131 import MP3Track

def main():
    t1 = MP3Track('Fools Gold', 604, ['The Stone Roses'])
    t2 = MP3Track('Shallow', 197)

    print(t1)
    print(t2)

    t2.add_artist('Lady Gaga')
    print(t2)

    t2.add_artist('Bradley Cooper')
    print(t2)

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

```
Title: Fools Gold
By: The Stone Roses
Duration: 604
Title: Shallow
By:
Duration: 197
Title: Shallow
By: Lady Gaga
Duration: 197
Title: Shallow
By: Lady Gaga, Bradley Cooper
Duration: 197
```

MP3Track part 4 [8 marks]

- In module *mp3track_v4_131.py* extend the `MP3Track` class to support printing its duration in *minutes:seconds* format.
- When your class is correctly implemented, running the following program should produce the given output.

```
from mp3track_v4_131 import MP3Track

def main():
    t1 = MP3Track('Fools Gold', 604, ['The Stone Roses'])
    t2 = MP3Track('Shallow', 197, ['Lady Gaga', 'Bradley Cooper'])
    t3 = MP3Track('Telephone', 220, ['Beyonce', 'Lady Gaga'])
    t4 = MP3Track('Her Majesty', 34, ['The Beatles'])
    t5 = MP3Track('Seven Seconds', 7, ['Neneh Cherry'])

    print(t1)
    print(t2)
    print(t3)
    print(t4)
    print(t5)

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

```
Title: Fools Gold
By: The Stone Roses
Duration: 10:04
Title: Shallow
By: Lady Gaga, Bradley Cooper
Duration: 3:17
Title: Telephone
By: Beyonce, Lady Gaga
Duration: 3:40
Title: Her Majesty
By: The Beatles
Duration: 0:34
Title: Seven Seconds
By: Neneh Cherry
Duration: 0:07
```

MP3Collection part 1 [8 marks]

- In module *mp3collection_v1_131.py* define an `MP3Collection` class to model a collection of MP3 tracks.
- You may assume that the title of each track in the MP3 collection is unique.
- **You must include in mp3collection_v1_131.py a copy of your MP3Track class definition from mp3track_v1_131.py.**
- MP3 tracks can be added to and removed from the collection using the `add()` and `remove()` methods respectively.
- Removing a track that is not in the collection has no effect.
- A `lookup()` method returns an `MP3Track` object if a given title is in the collection and `None` otherwise.
- When your class is correctly implemented, running the following program should produce no output.

```
from mp3collection_v1_131 import MP3Track, MP3Collection

def main():
    t1 = MP3Track('Fools Gold', 604)
    t2 = MP3Track('Shallow', 197)
    t3 = MP3Track('Telephone', 220)

    c = MP3Collection()

    c.add(t1)
    c.add(t2)
    c.add(t3)

    t = c.lookup('Fools Gold')
    assert(isinstance(t, MP3Track))
    assert(t.title == 'Fools Gold')
    assert(t.duration == 604)

    c.remove('Fools Gold')
    t = c.lookup('Fools Gold')
    assert(t is None)

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

MP3Collection part 2 [8 marks]

- In module *mp3collection_v2_131.py* extend the `MP3Collection` class to support track lookup by artist i.e. add a method `get_mp3s_by_artist()` that returns a list of all tracks by the given artist.
- **You must include in mp3collection_v2_131.py a copy of your MP3Track class definition from mp3track_v2_131.py.**
- When your class is correctly implemented, running the following program should produce no output.

```
from mp3collection_v2_131 import MP3Track, MP3Collection

def main():
    t1 = MP3Track('Fools Gold', 604, ['The Stone Roses'])
    t2 = MP3Track('Shallow', 197, ['Lady Gaga', 'Bradley Cooper'])
    t3 = MP3Track('Telephone', 220, ['Beyonce', 'Lady Gaga'])

    c = MP3Collection()

    c.add(t1)
    c.add(t2)
    c.add(t3)

    # Retrieve all MP3Tracks from the collection by Lady Gaga
    tracklist = c.get_mp3s_by_artist('Lady Gaga')
    assert(len(tracklist) == 2)
    assert(t2 in tracklist)
    assert(t3 in tracklist)

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

MP3Collection part 3 [8 marks]

- In module *mp3collection_v3_131.py* extend the `MP3Collection` class to support the combining (through addition) of MP3 collections.
- The new MP3 collection should contain the tracks in the original collections and should not contain duplicate tracks.
- **You must include in mp3collection_v3_131.py a copy of your MP3Track class definition from mp3track_v2_131.py.**
- When your class is correctly implemented, running the following program should produce no output.

```
from mp3collection_v3_131 import MP3Track, MP3Collection

def main():
    t1 = MP3Track('Fools Gold', 604, ['The Stone Roses'])
    t2 = MP3Track('Shallow', 197, ['Lady Gaga', 'Bradley Cooper'])
    t3 = MP3Track('Telephone', 220, ['Beyonce', 'Lady Gaga'])

    c1 = MP3Collection()
    c1.add(t1)
    c1.add(t2)

    c2 = MP3Collection()
    c2.add(t3)

    # Make a new collection from two existing ones
    c3 = c1 + c2
    assert(isinstance(c3, MP3Collection))
    assert(c3 is not c1)
    assert(c3 is not c2)

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

Cakes [16 marks]

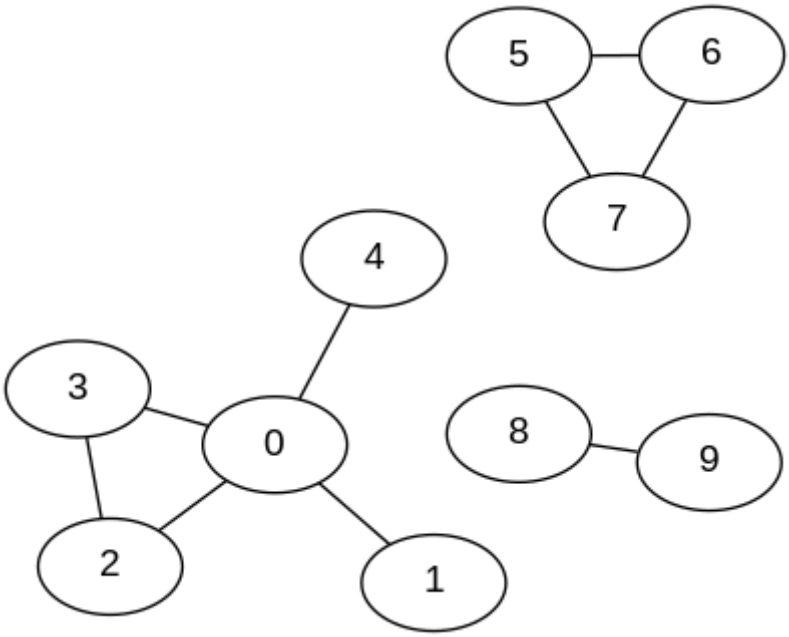
- A bakery is offering a special deal.
- For any three cakes bought the cheapest of the three comes free.
- Write a program called *cakes_131.py* that reads lines of text from `stdin`.
- Each line consists of an arbitrarily long list of cake prices (each price is a positive integer).
- For each line the program should output the minimum amount required to purchase those cakes (i.e. assume each customer groups their cakes in order to optimise their savings).
- For example:

```
$ cat cakes_stdin_00_131.txt
7 4 6
5
7 9 4 6
7 4
10 10 10 10 10 10 10
```

```
$ python3 cakes_131.py < cakes_stdin_00_131.txt
13
5
20
11
50
```

Islands [18 marks]

- Consider the graph below.



- A set of vertices forms an *island* in a graph if any vertex from the set of vertices can reach any other vertex by traversing links.
- Thus the graph above contains three islands: [0, 1, 2, 3, 4], [5, 6, 7], [8, 9].
- The graph is described by the text file below.
- The first line gives V, the number of vertices in the graph, with vertices numbered 0 to V-1.
- Each subsequent line defines a link between two vertices.

```
$ cat islands_stdin_00_131.txt
10
0 1
0 2
0 3
0 4
2 3
5 6
5 7
6 7
8 9
```

- Write a program called `islands_131.py` that reads in any graph described in the above format.
- The program should output the number of islands in the graph.
- For example:

```
$ python3 islands_131.py < islands_stdin_00_131.txt
3
```