CA117 FINAL EXAM (11 April 2022 : 0930-1230)

Before starting

- The exam runs 0930-1230.
- Answer all questions.
- Upload all code to Einstein.
- All lab exam rules apply.
- **The IP address of every login event is logged for analysis.**
- **The IP address of every submission you make to Einstein is logged for analysis.**

Question 1 [15 marks]

- Write a program called *`fizzbuzz_132.py`* that reads a single line of text from `stdin`.
- The line consists of three integers, X, Y, N (in that order) where ($1 \leq X \leq Y \leq N \leq 1000$).
- Your program should print the numbers 1 to N where numbers divisible by X are replaced by *fizz*, numbers divisible by Y are replaced by *buzz* and numbers divisible by both X and Y are replaced by *fizzbuzz*.
- For example:

```
$ cat fizzbuzz_stdin_00_132.txt
3 4 13
```

```
$ python3 fizzbuzz_132.py < fizzbuzz_stdin_00_132.txt
1
2
fizz
buzz
5
fizz
7
buzz
fizz
10
11
fizzbuzz
13
```

Question 2 [10 marks]

- In module *`vehicle_v1_132.py`* define a `Vehicle` class to model a vehicle.
- A vehicle has an ID number, a category, a mileage and a list of authorised drivers.
- When your class is correctly implemented, running the following program should produce the given output (drivers are printed comma-separated and in the order they are associated with a vehicle).

```
from vehicle_v1_132 import Vehicle

def main():
    v1 = Vehicle(21, 'van', 9100, ['joe'])
    v2 = Vehicle(22, 'car', 33000, ['mary'])
    v3 = Vehicle(33, 'truck', 16000, ['max', 'joe'])

    assert(v1.vin == 21)
    assert(v1.cat == 'van')
    assert(v1.mileage == 9100)
    assert(v1.drivers == ['joe'])

    print(v1)
    print(v2)
    print(v3)

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

```
ID: 21
Category: van
Mileage: 9100
Drivers: joe
ID: 22
Category: car
Mileage: 33000
Drivers: mary
ID: 33
Category: truck
Mileage: 16000
Drivers: max, joe
```

Question 3 [10 marks]

- In module *vehicle_v2_132.py* extend the `Vehicle` class such that authorised drivers can be associated with a vehicle either at the time of `Vehicle` object creation or subsequently by invoking the `add_driver()` method.
- When your class is correctly implemented, running the following program should produce the given output.

```
from vehicle_v2_132 import Vehicle

def main():
    v1 = Vehicle(21, 'van', 9100, ['joe'])
    v2 = Vehicle(22, 'car', 33000)

    print(v1)
    print(v2)

    v2.add_driver('mary')
    print(v2)

    v2.add_driver('fred')
    print(v2)

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

```
ID: 21
Category: van
Mileage: 9100
Drivers: joe
ID: 22
Category: car
Mileage: 33000
Drivers:
ID: 22
Category: car
Mileage: 33000
Drivers: mary
ID: 22
Category: car
Mileage: 33000
Drivers: mary, fred
```

Question 4 [10 marks]

- A vehicle must be serviced every 10000 miles.
- In module *vehicle_v3_132.py* extend the `Vehicle` class to report how many miles remain before a service is due.
- When your class is correctly implemented, running the following program should produce the given output.

```
from vehicle_v3_132 import Vehicle

def main():
    v1 = Vehicle(21, 'van', 9100, ['joe'])

    print(v1)

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

```
ID: 21
Category: van
Mileage: 9100
Drivers: joe
Service due in 900 miles
```

Question 5 [10 marks]

- In module *fleet_v1_132.py* define a `Fleet` class to model a collection of vehicles.
- You may assume that the vehicle ID number of each vehicle in the fleet is unique.
- **You must include in fleet_v1_132.py a copy of your Vehicle class definition from vehicle_v1_132.py.**
- Vehicles can be added to and removed from the fleet using the `add()` and `remove()` methods respectively.
- Removing a vehicle that is not in the collection has no effect.
- A `lookup()` method returns a `Vehicle` object if a given vehicle is in the fleet and `None` otherwise.

- When your class is correctly implemented, running the following program should produce no output.

```
from fleet_v1_132 import Vehicle, Fleet

def main():
    v1 = Vehicle(21, 'van', 9100, ['joe'])
    v2 = Vehicle(22, 'car', 33000, ['mary'])
    v3 = Vehicle(33, 'truck', 16000, ['max', 'joe'])

    f = Fleet()

    f.add(v1)
    f.add(v2)
    f.add(v3)

    v = f.lookup(21)
    assert(isinstance(v, Vehicle))
    f.remove(21)
    v = f.lookup(21)
    assert(v is None)

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

Question 6 [10 marks]

- In module *fleet_v2_132.py* extend the `Fleet` class with a method `get_drivers_by_category()` which returns the number of unique drivers authorised to drive a vehicle of the given category.
- **You must include in fleet_v2_132.py a copy of your Vehicle class definition from vehicle_v1_132.py.**
- When your class is correctly implemented, running the following program should produce the given output.

```
from fleet_v2_132 import Fleet, Vehicle

def main():
    v1 = Vehicle(21, 'van', 9100, ['joe'])
    v2 = Vehicle(22, 'car', 33000, ['mary'])
    v3 = Vehicle(33, 'truck', 16000, ['max', 'joe'])
    v4 = Vehicle(38, 'van', 18212, ['martha', 'joe'])

    f = Fleet()

    f.add(v1)
    f.add(v2)
    f.add(v3)
    f.add(v4)

    print(f.get_drivers_by_category('van'))

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

```
2
```

Question 7 [20 marks]

- Write a program called *`schedule_132.py`* that reads a list of one or more meeting times from `stdin`.
- Each meeting time has the format H:M Z where:
    - H is an integer in the range [1, 12] representing the hour,
    - M is a zero-padded two-digit integer in the range [00, 59] representing the minute,
    - Z is either *a.m.* or *p.m.* representing times before or after midday, respectively.
- Note the following:
    - A day starts at 12:00 a.m.
    - A day ends at 11:59 p.m.
    - 12:00 p.m. is midday.
- Your program should output a list containing the same times ordered from earliest to latest.
- For example:

```
$ cat schedule_stdin_00_132.txt
4:12 p.m.
10:02 p.m.
3:00 a.m.
```

```
$ python3 schedule_132.py < schedule_stdin_00_132.txt
3:00 a.m.
4:12 p.m.
10:02 p.m.
```

- For example:

```
$ cat schedule_stdin_01_132.txt
1:15 a.m.
12:30 a.m.
1:19 p.m.
1:19 a.m.
```

```
$ python3 schedule_132.py < schedule_stdin_01_132.txt
12:30 a.m.
1:15 a.m.
1:19 a.m.
1:19 p.m.
```

Question 8 [15 marks]

- Sheila is in a lift in a skyscraper.
- The lift is currently on floor *s*.
- The lift has two buttons, UP and DOWN.
- Pressing the UP button takes the lift up *u* floors (if there are not enough floors then pressing the button has no effect).

- Pressing the DOWN button takes the lift down *d* floors (if there are not enough floors then pressing the button has no effect).
- Sheila is trying to get to floor *g*.
- The skyscraper has *f* floors.
- Write a program called `lift_132.py` that reads in a single line of text from `stdin` consisting of five integers *f*, *s*, *g*, *u*, *d* (in that order).
- You know that $1 \leq s, g \leq f \leq 1000000$ and $0 \leq u, d \leq 1000000$.
- Your program should output the minimum number of button pushes required by Sheila in order to get from floor *s* to floor *g*.
- If it is impossible to reach floor *g* from floor *s* your program should output *Sorry Sheila!*
- For example:

```
$ cat lift_stdin_00_132.txt
10 1 10 2 1
```

```
$ python3 lift_132.py < lift_stdin_00_132.txt
6
```

- For example:

```
$ cat lift_stdin_01_132.txt
100 2 1 1 0
```

```
$ python3 lift_132.py < lift_stdin_01_132.txt
Sorry Sheila!
```