



### 1. Digit Set (40%).

A set is a collection of objects with no duplicate elements. In this program we consider sets whose elements are decimal digits: 0,1,2,3,4,5,6,7,8, or 9.

The cardinality of a set is the number of elements in it. For example,  $\{5, 2, 0\}$  and  $\{1, 2, 5, 9\}$  are sets of cardinality 3 and 4 respectively. The union of two sets ( $\cup$ ) is a set containing elements that appear in either of the two sets. The intersection ( $\cap$ ) of two sets is a set containing elements that appear in both sets. If we refer to the sets above as  $A$  and  $B$ , then  $A \cup B = \{5, 2, 0, 1, 9\}$  and  $A \cap B = \{2, 5\}$ .

Write a class `Dset` that implements the interface below:

- `__init__()`: constructor that takes no arguments or a single string argument to build the set. With no argument, an empty set should be constructed. A string argument should contain only digits. For example, `'76114'` constructs the set  $\{7, 6, 1, 4\}$ . The constructor should raise an appropriate exception if given an invalid input.
- `__len__()`: returns the cardinality of a set
- `in_set()`: takes a digit  $d$  as argument, returns a boolean indicating whether or not  $d$  is in the set
- `add_element()`: takes an integer digit  $d$  as argument and adds it to the set (modifying the set, if needed). The method should raise an exception unless  $0 \leq d \leq 9$ .
- `__eq__()`: takes a set  $s$  as argument, returns a boolean indicating whether this set and  $s$  have the same elements
- `union()`: takes a set  $s$  as argument, returns the union of this set and  $s$
- `intersection()`: takes a set  $s$  as argument, returns the intersection of this set and  $s$
- `__str__()`: returns a string representation of the set (of the form  $\{1, 4, 6, 7\}$ )

Your code should be in a file called `dset.py`, and should include a function that tests the methods of the `Dset` class. You have to think about and write an appropriate set of tests. You will be graded on the tests you generate and use.

### 2. Super Set (20%).

Write a program `superset.py` that prints the union of multiple digit sets. Your program should have:

- a function that reads a list of strings from the command line and returns a list of `Dset` objects
- a function that takes a list of `Dset` objects as argument, and generates the union of the list of sets

For example, an invocation of the program would produce something like:

```
C:> python superset.py 841 44 5015 1541
Union of all sets: {0 1 4 5 8}
```




### 3. Puzzle Search (40%).

Write a program to check if a word may be found in a two dimensional grid of letters. For example, given the following grid:

```
0 1 2 3 4 5 6 7
m v j l i x a p e
j h b c e e n p p
h u n o h b s w y
r w a m n u y z h
p p f p r d z k q
t p n u q o y j y
a n h t p f g b g
h x m e h w y l y
u a r r a y s o a
```

len word = 6  
3 9



your program will find if a word can be spelled out in the grid by starting at any character, then moving in a straight line down or right. For example, the grid above contains the word **computer** because it can be spelled out by starting at the character **c** in the second row, fourth column and moving down.

Write a program `puzzle.py` that reads a grid of characters from a file and checks whether or not a word may be found in it. If a word appears multiple times, only the first appearance found needs to be displayed.

As an example, the file `grid.txt` (download from Moodle) contains the following input for the grid shown above:

```
mvjlixape
jhbceenpp
hunohbswy
rwamnuyzh
ppfprdzkq
tpnuqoyjy
anhtpfgbg
hxmehwlyl
uarraysoa
```



Invocations of the program would produce something like:

```
> python puzzle.py grid.txt computer
computer found: row 2, column 4, going down
> java puzzle.py grid.txt party
party not found
```



You should pay careful attention to the organization of your program. You may want to first write functions or methods that solve individual pieces of the problem:

- read an input file and build the puzzle as a 2D list (i.e., list of lists) of characters
- check if a word can be found by starting at a given location and moving to the right
- check if a word can be found by starting at a given location and moving down

Make sure you test these helper functions/methods individually first. You can then use them to construct the complete puzzle search program.

### 4. Token3.

When you are ready to submit, get from your proctor your individualized 4-character token string and write a one line program `token3.py` that contains a single statement of the form `token = 'AB12'` which assigns to the variable `token` the value you get from the proctor.

**Submission.** Zip the four `.py` files above (`dset.py`, `superset.py`, `puzzle.py`, and `token3.py`) in a single archive file `exam3_netid` where `netid` is your AUBnet user name, and submit to Moodle.