

COMP10001 - Sem 2 2024 - Week 6

Foundations of Computing



Daksh Agrawal



Checkpoint 1



Project 1

Out

GET STARTED NOW

Grok Tips

Assume that `seq` is a list. Write one line of code to:

Sort `seq` in-place

Assign the sorted version of `seq` to a new variable `new_seq`. Do not change the original list.

Add the string `"hi"` to the end of `seq`.

What is the output of the following code? Classify the variables by which namespace they belong in.

```
def foo(x, y):  
    a = 42  
    x, y = y, x  
    print(a, b, x, y)
```

```
a, b, x, y = 1, 2, 3, 4  
foo(17, 4)  
print(a, b, x, y)
```

What is the output of this code? Why?

```
def mystery(x):  
    x.append(5)  
    x[0] += 1  
    print("mid-mystery:", x)
```

```
my_list = [1, 2]  
print(my_list)  
mystery(my_list)  
print(my_list)  
mystery(my_list.copy())  
print(my_list)
```

Compare the two functions below. Are they equivalent? Why would we prefer one over the other?

```
1  def noletter_1(words, letter='z'):
2      for word in words:
3          if letter in word:
4              return False
5      return True
6
7
8  1 usage  new *
9  def noletter_2(words, letter='z'):
10     no_z = True
11     for word in words:
12         if letter in word:
13             no_z = False
14     return no_z
15
16 wordlist = ['zizzer'] + ['aardvark'] * 10_000_000
17 print(noletter_1(wordlist))
18 print(noletter_2(wordlist))
```


Write a Python function `find_ints(text)` that takes a (possibly empty) Python string `text` and returns a (possibly empty) list of the word locations at which integers occur. A “word” is defined as a consecutive sequence of non-whitespace characters, and an “integer” is defined as a word that is either completely made up of digits, or is a single `+` or `-` sign, and then nothing but digits. Word positions within text are counted from one.

For example:

- `find_ints("Ints -34 there and here +551 but not here88")` should return `[2, 6]`
- `find_ints("No integers here99 88-77 or there")` should return `[]`
- `find_ints("+18 and -777 and 666 are all 3 integers")` should return `[1, 3, 5, 8]`

Hint: if you get stuck, try separating the first character from the rest for each word.

```
1 def find_ints(text):
2     words = text.split()
3     results = []
4     for i, word in enumerate(words):
5         word_cut_off = word.lstrip("+").lstrip("-")
6         if word_cut_off.isdigit():
7             results.append(i + 1)
8     return results
```

Write a function which takes two lists as input and returns a list containing the numbers which they both have in common.

in_common([1, 2, 4], [3, 4, 5]) should return **[4]**.

```
def is_common(lst_1, lst_2):
```

```
    set_1 = set(lst_1)
```

```
    set_2 = set(lst_2)
```

```
    set_inter = set_1.intersection(set_2)    set_1 & set_2
```

```
    return list(set_inter)
```

```
def is_common(lst_1, lst_2):
```

```
    return list(set(lst_1) & set(lst_2))
```

Write a function which takes a dictionary and returns a sorted list containing the unique values in that dictionary.

unique_values({**'a'**: 1, **'b'**: 0, **'c'**: 0}) should return **[0, 1]**.

```
def unique_values(d):  
    vals = d.values()  
    s = set(vals)  
    l = list(s)  
    return sorted(l)
```


Write a function which takes a list of words and checks whether any of those words are palindromes (spelled the same way backwards as forwards, like "kayak"). It should return True if there are any palindromes and False if there are none. Use a timely return to save some time!

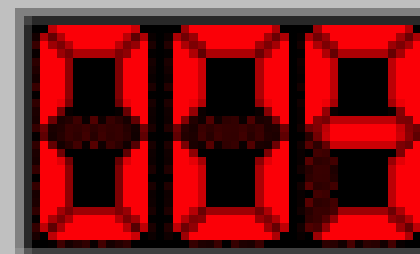
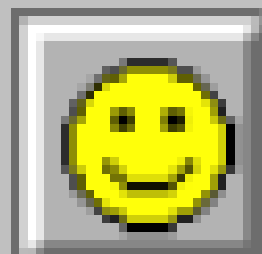
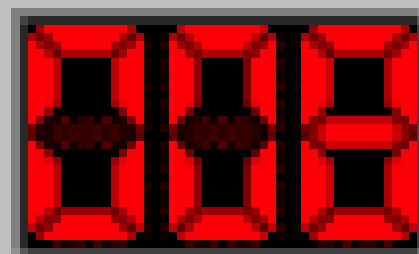
any_palindrome(['kayak', 'motorbike', 'scooter']) should return **True**.

```
41 def any_palindrome(words):
42     for word in words:
43         if word == word[::-1]:
44             return True
45     return False
```

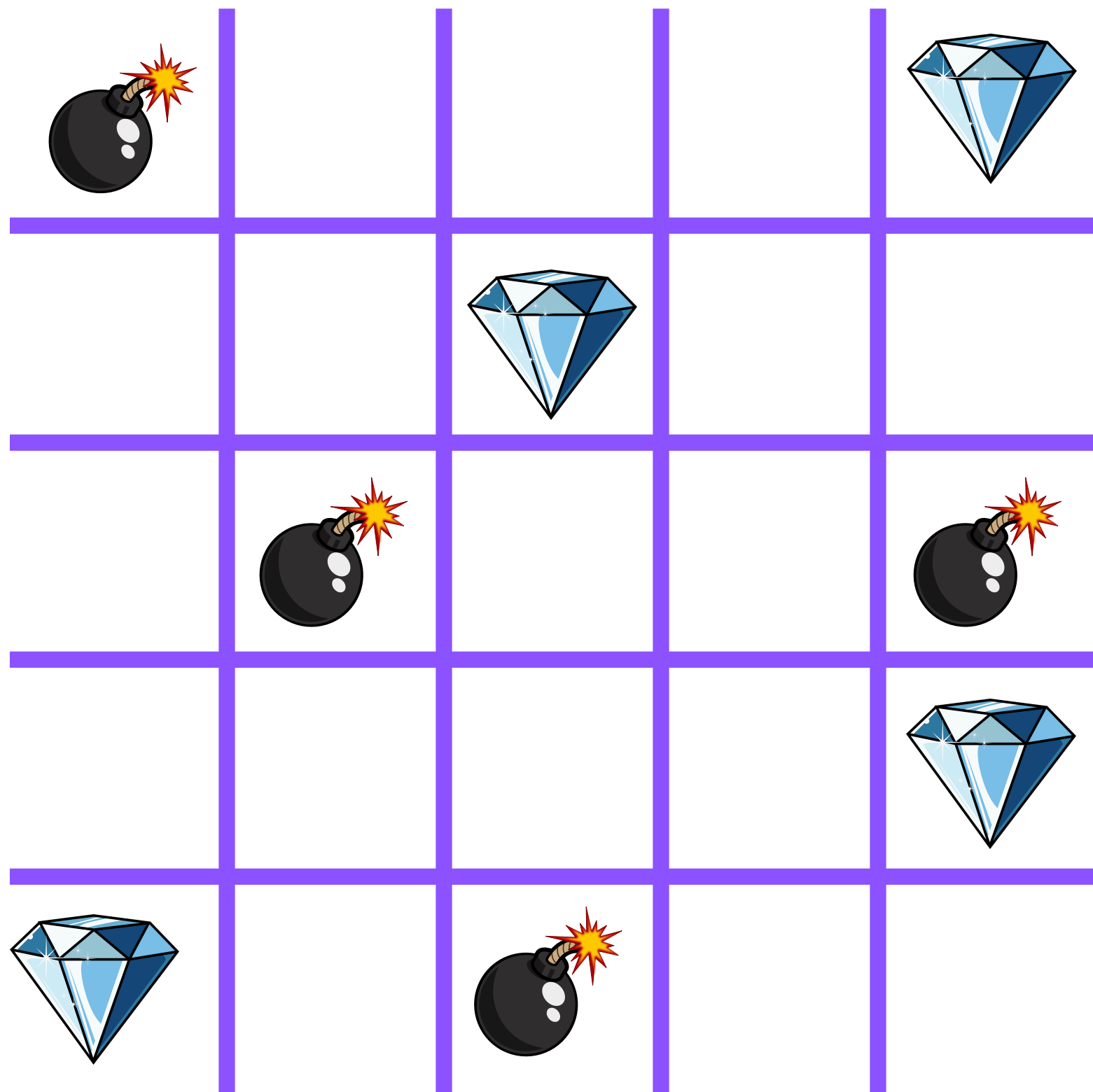
word[::-1]

MAKE YOU MINE







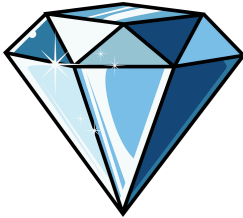





| | | | | | | | | |
|---|---|--|---|---|---|---|--|--|
| | | | | 1 | | | | |
| | | | | 1 | 2 | | | |
| | 1 | 1 | 1 | | 1 | 1 | | |
| | 1 |  | 1 | | | 1 | | |
| 1 | 2 | 2 | 1 | | | 1 | | |
| 1 |  | 1 | 1 | 1 | 1 | 1 | | |
| 1 | 1 | 1 | 1 | | | | | |
| | | 1 | 3 | | | | | |
| | | 1 | | | | | | |



[

["b", "e", "e", "e", "d"],
["e", "e", "d", "e", "e"],
["e", "b", "e", "e", "b"],
["e", "e", "e", "e", "d"],
["d", "e", "b", "e", "e"]

]

| | 0 | 1 | 2 | 3 | 4 |
|---|---|--|---|---|---|
| 0 |  | | | |  |
| 1 | | |  | | |
| 2 | |  | | |  |
| 3 | | | | |  |
| 4 |  | |  | | |

| | |
|---|---|
| 0 | 2 |
| 3 | 2 |
| 4 | 0 |
| 3 | 1 |
| 2 | 1 |

```
11 def contains_diamonds(board):
12     for row in board:
13         if "d" in row:
14             return True
15     return False
16
17
18 new *
19 def make_you_mine(board):
20     diamonds = 0
21     while True:
22         user_input = input("where mine?")
23         x, y = user_input.split()
24         x = int(x)
25         y = int(y)
26
27         board_item = board[x][y]
28         if board_item == "e":
29             print("Empty!")
30         elif board_item == "b":
31             print("BOMB!! GAME OVER")
32             break
33         else:
34             diamonds += 1
35             print("Found Diamond")
36             board[x][y] = "e"
37             if not contains_diamonds(board):
38                 print("You WON!! Congratulations")
39                 break
```

[illegible]

Anonymus Feedback

