Jerry Cain
CS 106AX

Handout #16
October 29th, 2025

Section Handout #5

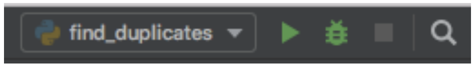
Problem 1: Finding duplicates in arrays

Download the starter code from the course website and open it in PyCharm. Open **`find_duplicates.py`**, and write a function

```
def findDuplicate(array)
```

that returns the first element in **`array`** that appears more than once. If there is no duplicated value in the array, **`findDuplicate`** should return the value **`None`**.

You can run the program by selecting **`find_duplicates`** to the left of the green "play" button in the upper right corner, then clicking "play."



The following examples illustrate the values that **`findDuplicate`** should return:

```
findDuplicate([1, 2, 3, 4, 3, 2]) → 2
findDuplicate(['a', 'b', 'c', 'd', 'c']) → 'c'
findDuplicate([1, 2, 3, 4, 5, 6, 7]) → None
findDuplicate([]) → None
```

Problem 2: Playfair Ciphers

The ***Playfair cipher*** relies on a 5x5 table of letters—all letters of the alphabet minus the Q, which is omitted because it's so rare—to guide the encryption of a cleartext message. The table is constructed from a ***passphrase*** by filling in the 25 slots from left to right, top to bottom, with the letters of the passphrase in the order they first appear and then filling any remaining spots with the rest of the letters of the alphabet.

For instance, if the passphrase is **`'OBSERVEOBLONGCHEWYINERT'`**, the table would look like this:

O	B	S	E	R
V	L	N	G	C
H	W	Y	I	T
A	D	F	J	K
M	P	U	X	Z

For this problem, you'll implement just the `construct_playfair_table` function, which accepts the passphrase and constructs the relevant table as described above. The table is really list of length 5, where each entry itself is a list of 5, one-character strings. You may assume the passphrase is composed of uppercase letters minus the `'Q'`. So, a call to `construct_playfair_table('OBSERVEOBLONGCHEWYINERT')` might go like this:

```
>>> print(construct_playfair_table('OBSERVEOBLONGCHEWYINERT'))
  [['O', 'B', 'S', 'E', 'R'],
   ['V', 'L', 'N', 'G', 'C'],
   ['H', 'W', 'Y', 'I', 'T'],
   ['A', 'D', 'F', 'J', 'K'],
   ['M', 'P', 'U', 'X', 'Z']]
>>>
```

Problem 3: Reading data structures from files

In World War II, the German Enigma operators used a codebook to set up the machine for each day's transmission. The complete settings consisted of the following pieces of information (of which you implemented only the second in Assignment #4):

- *The rotor order.* The wartime Enigma machines came equipped with a box of five rotors, which could be inserted into the machine in any order. The codebook told the operator which rotors to choose for the three rotors. For example, the rotor order **`513`** asks the operator to use stock rotor #5 as the slow rotor, stock rotor #1 as the medium rotor, and stock rotor #3 as the fast rotor.
- *The rotor setting.* The rotor setting was a three-letter code showing what letters should be set on the three rotors. For example, several examples in the Adventure handout used the setting **`JLY`**.
- *The Stecker pairings.* During the war, the German military also added a plug board (*Steckerbrett* in German) to the front of the apparatus, which would swap pairs of letters at the beginning and end of the encryption process. Although the actual Enigma machine typically used ten pairs of letters in its Stecker pairings, this problem assumes that there are always four Stecker pairs, mostly so the examples fit on the page.

Your job in this problem is to read a data file containing these codebook values for every day in a year. Each line of the file has the form

date order setting stecker

where the individual components of the line have the following values:

- The *date* component is the date, written as a string (without the quotes).
- The *order* component is the rotor order, written as a three-digit integer.
- The *setting* component is the rotor setting, written as a string of three letters.
- The *stecker* component consists of four letter pairs separated by spaces.

For example, the line

```
1-Feb-44 241 YVM AC LS BQ WN
```

tells the operator that on February 1st, 1944, the rotors should be inserted in the order 241, that the rotors should be rotated to show the letters **`Y`**, **`V`**, and **`M`**, and that four wires should be added to the plug board connecting the pairs of letters **`A`**⌦**`C`**, **`L`**⌦**`S`**, **`B`**⌦**`Q`**, and **`W`**⌦**`N`**.

You can run this program by selecting **`enigma_settings`** next to the "play" icon in the upper right of PyCharm. For this problem, your job is to open **`enigma_settings.py`** and write the function

```
def readEnigmaCodebook(filename)
```

which reads a file consisting of lines such as the example shown on the preceding page and returns a dictionary linking dates to dictionaries describing the setting. The components of each setting dictionary are:

- A **`rotorOrder`** field, which is an integer
- A **`rotorSetting`** field, which is a three-letter string
- A **`steckerPairings`** field, which is an array of four two-letter strings

Suppose, for example, that the file **`Feb44.txt`** has the following contents:

`EnigmaCodebook.txt`

```
1-Feb-44 241 YVM AC LS BQ WN
2-Feb-44 135 FQW DS VJ FS QH
3-Feb-44 354 NKT DY CF WN OV
4-Feb-44 421 PHP KD FQ CO VJ
5-Feb-44 243 RZW MK GO RQ XT
```

Calling **`readEnigmaCodebook("EnigmaCodebook.txt")`** should produce the following data structure:

```
{
    '1-Feb-44': {'rotorOrder': '241',
                 'rotorSetting': 'YVM',
                 'steckerPairing': ['AC', 'LS', 'BQ', 'WN']},
    '2-Feb-44': {'rotorOrder': '135',
                 'rotorSetting': 'FQW',
                 'steckerPairing': ['DS', 'VJ', 'FS', 'QH']},
    '3-Feb-44': {'rotorOrder': '354',
                 'rotorSetting': 'NKT',
                 'steckerPairing': ['DY', 'CF', 'WN', 'OV']},
    '4-Feb-44': {'rotorOrder': '421',
                 'rotorSetting': 'PHP',
                 'steckerPairing': ['KD', 'FQ', 'CO', 'VJ']},
    '5-Feb-44': {'rotorOrder': '243',
                 'rotorSetting': 'RZW',
                 'steckerPairing': ['MK', 'GO', 'RQ', 'XT']}
}
```

Note: Typically, we would rather store the settings as class-based objects instead of dictionaries. However, we are using dictionaries in this problem to give you some extra practice with using them, and because we haven't covered classes in lecture yet.

In answering this question, you should keep the following points in mind:

- All you have to do is read the data into the internal structure. Any code that uses the codebook data structure is the responsibility of your clients.
- You do not need to do any error-checking. Thus, you may assume that the file exists and that every line in the file is properly formatted with exactly one space between each field.