THE PROCESS OF WRITING PROGRAMS II

REMINDERS

- HW03 Presentations this week

TODAY'S PRIMARY QUESTION

> **WHAT DO I DO WHEN I RUN INTO A PROBLEM I DON'T IMMEDIATELY KNOW HOW TO SOLVE?**

Note: Today's topic is based on the first parts of the framework presented in *How to Design Programs* (a textbook by Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi) and the PEDAC process designed by the LaunchSchool.
If you want more information or practice I suggest you check them out.

WRITING A PROGRAM SYSTEMATICALLY

Overall, solving a programming problem take ~9 steps -->

1. Understand the Problem
2. Write your Signature(s) and Purpose
3. Identify Examples/Test Cases
4. Solve your Examples by Hand
5. Identify Your Patterns
6. Write your Outline
7. Fill in the Gaps
8. Code!
9. Test

WRITING A PROGRAM SYSTEMATICALLY

Overall, solving a programming problem take ~9 steps -->

Today we're focusing on the next final 6

1. Understand the Problem
2. Write your Signature and Purpose
3. Identify Examples/Test Cases
4. **Solve your Examples by Hand**
5. **Identify Your Patterns**
6. **Write your Outline**
7. **Fill in the Gaps**
8. **Code!**
9. **Test**

THE PROBLEM FROM LAST TIME

You are building a digital version of a domino-style word game called "Word Train." To make a valid train, each boxcar (word) must connect to the next one by matching its last letter to the first letter of the next boxcar.

A train is represented by a list of strings, where each string is a word (boxcar).

Your task is to write a Python program that checks if a given train is valid.

STEP 1: UNDERSTAND THE PROBLEM

Questions to ask yourself:

- What is the overall goal of this program?
- What variables do we need to track? How are they represented/what data types make sense for them?
- What are our expected inputs and outputs?
- What other information (if any) do we need to do this task?

STEP 1: UNDERSTAND THE PROBLEM

Questions to ask yourself:

- What is the overall goal of this program?
- What variables do we need to track? How are they represented/ what data types make sense for them?
- What are our expected inputs and outputs?
- What other information (if any) do we need to do this task?

Your answers:

- To check a given list of strings to see if each string ends in the same letter the next starts with
- The train (list[str]), last letter in prev word (str), first letter in current word (str)
- in --> list[str], out --> boolean
- What to do about case? (ignore case) What about the empty list? (True) Single letter strings? (treat as first and last)

STEP 2: WRITE YOUR SIGNATURE(S) AND PURPOSE

1. Write a comment or a function signature (or both!) to help yourself understand the moving in and outputs
2. If you identified that you'll need additional information (more functions), you can write those signatures too!

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule -->
        the last letter of index i must be the first of index i+1
    Input: a list of strings that is to be checked
    Output: a boolean, true if valid, false otherwise
    """
```

STEP 3: IDENTIFY EXAMPLES/TEST CASES

1. What are some examples that you can use to check your program?
2. Find some obvious edge cases that you can use to check yourself as well

Add some additional examples to the table below to create some test cases

INPUT	EXPECTED OUTPUT
["hi", "is", "so"]	True
["maya", "a", "and"]	True
[]	True
["hello", "so", "otter"]	False

STEP 3.5: WRITE YOUR TEST CASES AS UNIT TESTS

Examples:

```
def test_two_letters_list():
    input = ["hi", "is" "so"]
    output = is_valid_train(input)
    assert output
```

```
def test_failure_on_first_pair():
    input = ["hello", "so" "otter"]
    output = is_valid_train(input)
    assert not output
```

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Grab your examples and solve them how you would do it *as a human*

Remember to pay attention to every step you take -- Don't skip steps!

Make a note of what you actually do ✍️

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Let's try doing this case:
["hi", "is", "so] --> remember we expect it to be True!

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

STEP 4: SOLVE YOUR EXAMPLES BY HAND

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (third) word in the list

STEP 4: SOLVE YOUR EXAMPLES BY HAND

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

STEP 4: SOLVE YOUR EXAMPLES BY HAND

Your turn! **[C1]**

Write down every step you take when solving this example:
["east", "trauma", "among", "along", "gone"]

STEP 5: IDENTIFY YOUR PATTERNS

Look at what you wrote and consider:

- Any time I "take" or "look at" or "grab" or "save" something --> probably setting or updating a variable
- Any time I "check" something or "if" true --> probably a conditional
- Any time I "decide" or "conclude" --> probably a return
- Any time I "display" something --> probably a print
- Any time I "note" or "remember" --> probably storing in a list, set, dict, etc.

IMPORTANTLY:

- Check if you are doing the same sequence of actions on different items over and over and over... --> Probably a loop (or occasionally a function)

Step 5: Identify Your Patterns (what do you notice?)

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

Step 5: Identify Your Patterns (what do you notice?)

First, I **look** at the first word in the list
Then, I **look** at the last letter in that word (let's call it LAST)
I **look** at the second word in the list
Then, I **look** at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I **look** at the next (second) word in the list
Then, I **look** at the last letter in that word (let's call it LAST)
I **look** at the third word in the list
Then, I **look** at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I **look** at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

Step 5: Identify Your Patterns (what do you notice?)

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I **check if** LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I **check if** LAST is equal to FIRST

I look at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

Step 5: Identify Your Patterns (what do you notice?)

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (second) word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the third word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST

I look at the next (third) word in the list
There is no fourth word to compare to so I **stop here and decide it's valid**

Step 5: Identify Your Patterns (what do you notice?)

1>First, I look at the first word in the list
1>Then, I look at the last letter in that word (let's call it LAST)
1>I look at the second word in the list
1>Then, I look at the first letter in that word (let's call it FIRST)
1>Then I check if LAST is equal to FIRST

2>I look at the next (second) word in the list
2>Then, I look at the last letter in that word (let's call it LAST)
2>I look at the third word in the list
2>Then, I look at the first letter in that word (let's call it FIRST)
2>Then I check if LAST is equal to FIRST

3>I look at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

STEP 5: IDENTIFY YOUR PATTERNS [C1]

Look at the steps you wrote in C1 and leave some notes about patterns your steps might represent.

Underline parts that you think might be variable assignment
Circle parts that you think might be a conditional
Draw a ~wavy~ underline under parts that you think might be a return or print
Draw a square around parts that you think might be the body of a loop

STEP 6: WRITE YOUR OUTLINE

Using the patterns you identified, try to rewrite your steps to mimic steps you might take in code

STEP 6: WRITE YOUR OUTLINE

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST
👇

save the first word in a variable called CURRENT
save the last letter of CURRENT in a variable called LAST
save the second word in a variable called NEXT
save the first letter of NEXT in a variable called FIRST
if LAST == FIRST then continue as normal

STEP 6: WRITE YOUR OUTLINE

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST
👇

save the first word in a variable called CURRENT
save the last letter of CURRENT in a variable called LAST
save the second word in a variable called NEXT
save the first letter of NEXT in a variable called FIRST
if LAST == FIRST then continue as normal

🌀 BUT! I realized that this was part of a loop remember?🌀

Step 5: Identify Your Patterns (what do you notice?)

1>First, I look at the first word in the list
1>Then, I look at the last letter in that word (let's call it LAST)
1>I look at the second word in the list
1>Then, I look at the first letter in that word (let's call it FIRST)
1>Then I check if LAST is equal to FIRST

2>I look at the next (second) word in the list
2>Then, I look at the last letter in that word (let's call it LAST)
2>I look at the third word in the list
2>Then, I look at the first letter in that word (let's call it FIRST)
2>Then I check if LAST is equal to FIRST

3>I look at the next (third) word in the list
There is no fourth word to compare to so I stop here and decide it's valid

STEP 6: WRITE YOUR OUTLINE

First, I look at the first word in the list
Then, I look at the last letter in that word (let's call it LAST)
I look at the second word in the list
Then, I look at the first letter in that word (let's call it FIRST)
Then I check if LAST is equal to FIRST
👇

For each word in the list:
. save that word in a variable called CURRENT
. save the last letter of CURRENT in a variable called LAST
. save the second word in a variable called NEXT
. save the first letter of NEXT in a variable called FIRST
. if LAST == FIRST then continue as normal

STEP 6: WRITE YOUR OUTLINE

For each word in the list:
. save that word in a variable called CURRENT
. save the last letter of CURRENT in a variable called LAST
. save the second word in a variable called NEXT
. save the first letter of NEXT in a variable called FIRST
. if LAST == FIRST then continue as normal

🤔🤔 Is this enough?
Take a look at the steps you wrote on your paper and the patterns you identified.
Are there any changes you would make to this outline?

STEP 6: WRITE YOUR OUTLINE

For each word in the list:
. save that word in a variable called CURRENT
. save the last letter of CURRENT in a variable called LAST
. save the second word in a variable called NEXT
. save the first letter of NEXT in a variable called FIRST
. if LAST == FIRST then continue as normal

after we finish the whole list, if we haven't stopped then return TRUE

STEP 7: FILL IN THE GAPS

This step may or may not be relevant to you.

It's a moment to ask: "are my steps simple enough that each is about a line of code? or am I doing something complicated all at once?"

For example:
"Find all the odd numbers in the list" --> Maybe too complicated (unless we have a function available that does this for us)

"Find every word that starts with K" --> Maybe too complicated (but one day when this feels trivial, maybe not!)

STEP 8: CODE! (FINALLY!)

Take your outline and put it in your signature from step 2 as a comment.

Replace your comment with code, line by line.

If you find that a line of your outline is too complicated for one line of code, you might want to go back and repeat steps

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    # For each word in the list: <--
    # save that word in a variable called CURRENT
    # save the last letter of CURRENT in a variable called LAST
    # save the second word in a variable called NEXT
    # save the first letter of NEXT in a variable called FIRST
    # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for word in train:
        # save that word in a variable called CURRENT <--
        # save the last letter of CURRENT in a variable called LAST
        # save the second word in a variable called NEXT
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for current in train:
        # save the last letter of CURRENT in a variable called LAST <--
        # save the second word in a variable called NEXT
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for current in train:
        last = current[-1]
        # save the second word in a variable called NEXT <--
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for current in train:
        last = current[-1]
        next = train... # UHOH!
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        # save that word in a variable called CURRENT <--
        # save the last letter of CURRENT in a variable called LAST
        # save the second word in a variable called NEXT
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        # save the last letter of CURRENT in a variable called LAST <--
        # save the second word in a variable called NEXT
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        # save the second word in a variable called NEXT <--
        # save the first letter of NEXT in a variable called FIRST
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        # save the first letter of NEXT in a variable called FIRST <--
        # if LAST is not equal to FIRST then stop and return FALSE


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        first = next_word[0]
        # if LAST is not equal to FIRST then stop and return FALSE <--


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        first = next_word[0]
        if first != last:
            return False


    # after we finish the whole list, if we havent stopped then return T
```

STEP 8: CODE [C2] TRY TO WRITE THE FUNCTION

```
def is_valid_train(train : list[str]) -> bool:
    """
    Description: checks if all strings in list train follow the rule ->
        the last letter of index i must be the first of index i+1
    """
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        first = next_word[0]
        if first != last:
            return False
    return True
```

STEP 9: TEST

Run your unit tests! See if you need to tweak anything (you probably do) but be confident that you have a good start

Any guesses on where the bug is in our code? **[L1]**

```
def is_valid_train(train : list[str]) -> bool:
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        first = next_word[0]
        if first != last:
            return False
    return True
```

STEP 9: TEST

Run your unit tests! See if you need to tweak anything (you probably do) but be confident that you have a good start

Any guesses on where the bug is in our code? **[L1]**

```
def is_valid_train(train : list[str]) -> bool:
    for i in range(len(train)):
        current = train[i]
        last = current[-1]
        next_word = train[i+1] #<-- We'll try to call outside the list!
        first = next_word[0]
        if first != last:
            return False
    return True
```

STEP 9: TEST

Run your unit tests! See if you need to tweak anything (you probably do) but be confident that you have a good start

Any guesses on where the bug is in our code? **[L1]**

```
def is_valid_train(train : list[str]) -> bool:
    for i in range(len(train)-1): #<-- Fix
        current = train[i]
        last = current[-1]
        next_word = train[i+1]
        first = next_word[0]
        if first != last:
            return False
    return True
```

IF TIME: TRY TO WRITE THE OUTLINE [C3]

The question:

You are working the gate at a sold-out concert. People are streaming in, and their ticket numbers are being logged into a list (as strings). Because of a glitch in the online system, you suspect a counterfeit ticket was scanned twice. You need to find the very first ticket number that appears a second time.

Write a function that returns the first element that shows up more than once. If every ticket is unique, return `None`.

Step 4: Solve your Examples by Hand
Step 5: Identify Your Patterns
Step 6: Write your Outline

Some examples from last time:

INPUT	EXPECTED OUTPUT
["1110", "1111"]	None
["1", "2", "3", "1"]	"1"
["12", "42", "12", "42"]	"12"

Example steps I might take:

I take the first value in the list ("1")
I note down that I've seen "1"

I take the second value in the list ("2")
I check if I've already noted down "2"
I note down that I've seen "2"

I take the next value in the list ("3")
I check if I've already noted down "3"
I note down that I've seen "3"

I take the next value in the list ("1")
I check if I've already noted down "1"
Since I have, I stop and decide that "1" has been repeated

STEP 5: IDENTIFY YOUR PATTERNS

Look at what you wrote and consider:

- Any time I "take" or "look at" or "grab" or "save" something --> probably setting or updating a variable
- Any time I "check" something or "if" true --> probably a conditional
- Any time I "decide" or "conclude" --> probably a return
- Any time I "display" something --> probably a print
- Any time I "note" or "remember" --> probably storing in a list, set, dict, etc.

IMPORTANTLY:

- Check if you are doing the same sequence of actions on different items over and over and over... --> Probably a loop (or occasionally a function)

Example steps I might take:

1>I *take* the first value in the list ("1")
1>I `note down` that I've seen "1"

2>I *take* the second value in the list ("2")
I **check if** I've already noted down "2"
2>I `note down` that I've seen "2"

3>I *take* the next value in the list ("3")
I **check if** I've already noted down "3"
3>I `note down` that I've seen "3"

4>I *take* the next value in the list ("1")
I **check if** I've already noted down "1"
Since I have, I stop and decide that "1" has been repeated

🤔 CAN I MAKE MY LOOPS MORE LOOPY?

There's some repetition but it's not quite matching up. Are there steps I've forgotten?

1>I *take* the first value in the list ("1")
1>I `note down` that I've seen "1"

2>I *take* the second value in the list ("2")
I **check if** I've already noted down "2"
2>I `note down` that I've seen "2"

3>I *take* the next value in the list ("3")
I **check if** I've already noted down "3"
3>I `note down` that I've seen "3"

🤔 CAN I MAKE MY LOOPS MORE LOOPY?

There's some repetition but it's not quite matching up. Are there steps I've forgotten?

1>I *take* the first value in the list ("1")
1>I **check if** I've already noted down "1"
1>I `note down` that I've seen "1"

2>I *take* the second value in the list ("2")
2>I **check if** I've already noted down "2"
2>I `note down` that I've seen "2"

3>I *take* the next value in the list ("3")
3>I **check if** I've already noted down "3"
3>I `note down` that I've seen "3"

Example early outline from my steps:

for each ticket
. save the string into variable TICKET
. check if TICKET has already been seen
. if TICKET hasn't been seen, then note that I saw it
. if TICKET has been seen, then return TICKET

if we get through the whole thing without stopping then return None

Example early outline from my steps:

for each ticket
. save the string into variable TICKET
. **check if TICKET has already been seen** <-- how do we do this??
. if TICKET hasn't been seen, then note that I saw it
. if TICKET has been seen, then return TICKET

if we get through the whole thing without stopping then return None

Example outline from my steps:

create a list of SEEN tickets

for each ticket
. save the string into variable TICKET
. check if TICKET is in SEEN
. if TICKET isnt in SEEN, then add it to SEEN
. if TICKET is in SEEN, then return TICKET

if we get through the whole thing without stopping then return None

Example outline from my steps:

create a list of SEEN tickets <-- I realized that actually I only need to know if it exists, order and duplicates don't matter! What does that sound like? 🤔🤔🤔

for each ticket
. save the string into variable TICKET
. check if TICKET is in SEEN
. if TICKET hasn't been seen, then note that I saw it
. if TICKET has been seen, then return TICKET

if we get through the whole thing without stopping then return None

Example outline from my steps:

create a **set** of SEEN tickets

for each ticket
. save the string into variable TICKET
. check if TICKET is in SEEN
. if TICKET hasn't been seen, then note that I saw it
. if TICKET has been seen, then return TICKET

if we get through the whole thing without stopping then return None