Python Workshop Series 2020 Fall

Session 03: Problem Solving Strategies & Python Review

Raphael Yee

26th Nov 2020

01 **Problem Solving Strategies**

02 **Strings**

03 **Arrays, slicing and indexing**

04 **Dictionaries**

05 **Advanced Problem Sets**

Problem Solving Strategies

Breaking down the problem

When you first encounter a problem, **stop.**
Do not start coding.

Breaking down the problem

Ask yourself:

1. What variables am I provided?
2. What are my inputs?
3. What should be my output?

Breaking down the problem

When you encounter difficult problems, ask yourself:

1. What variables am I provided?
2. What are my inputs?
3. What should be my output?
4. Does order matter?

```
Sort a given list of integers [2,3,4,1,5] from small to
large.
```

Breaking down the problem

```
You are given two lists:
- List1 contains a set of scrambled letters.
- List2 contains the correct index position of each letter to form a word.
Create a function that takes in List1 and List2 and return the word
```

```
def word_builder(letters, positions):
    # 0. define empty string
    # 1. loop through positions
    # 2. get the letter at n index
    # 3. append the letter to the string
    # 4. return string
```

```
def word_builder(letters, positions):
    result = ""
    for n in positions:
        letter = letters[n]
        result += letter
    return result
```

Iterative Problem Strategies

If you see a for/while loop,
track your values!

Iterative Problem Strategies

```
sum = 0
for i in range(2, 10, 3):
    sum += i
What is the final value of sum?
```

Coding is about having the humility to accept that sometimes brain power just isn’t enough.

Iterative Problem Strategies

When you encounter indexing…

1. look for your min/max index
2. What is your step size?
3. Do you see a pattern?

Iterative Problem Strategies

The function **swap_elements** takes in a list **ls**.

- It swaps the list elements at indices index1 and index2.
- If index1 or index2 are integers that are outside of the valid indices in the list, return None.
- Remember that list indices can be negative.

```
ls = [3, 6, 8, 7]
new_ls = swap_elements(ls, 2, 3)
print(new_ls)

Expected Output: [3, 6, 7, 8]
```

```
ls = [3, 6, 8, 7]
new_ls = swap_elements(ls, 1, -1)
print(new_ls)

Expected Output: [3, 8, 7, 1]
```

Iterative Problem Strategies

```
def swap_elements(ls, index1, index2):

    # check if index1 is valid
    if index1 > len(ls)-1 or index1 <= -len(ls):
        return None

    # check if index2 is valid
    if index2 > len(ls)-1 or index2 <= -len(ls):
        return None

    #swap
    ls[index1], ls[index2] = ls[index2], ls[index1]

    return ls
```

Note: You can use indexing for strings as well!

Immutability

Encountering mutability questions…

1. What is this data type?
2. Am I changing the original variable?

Mutability = can change or not?

Strings…

```
test_string = “abc”
test_string[0] = “1”
print(test_string)
```

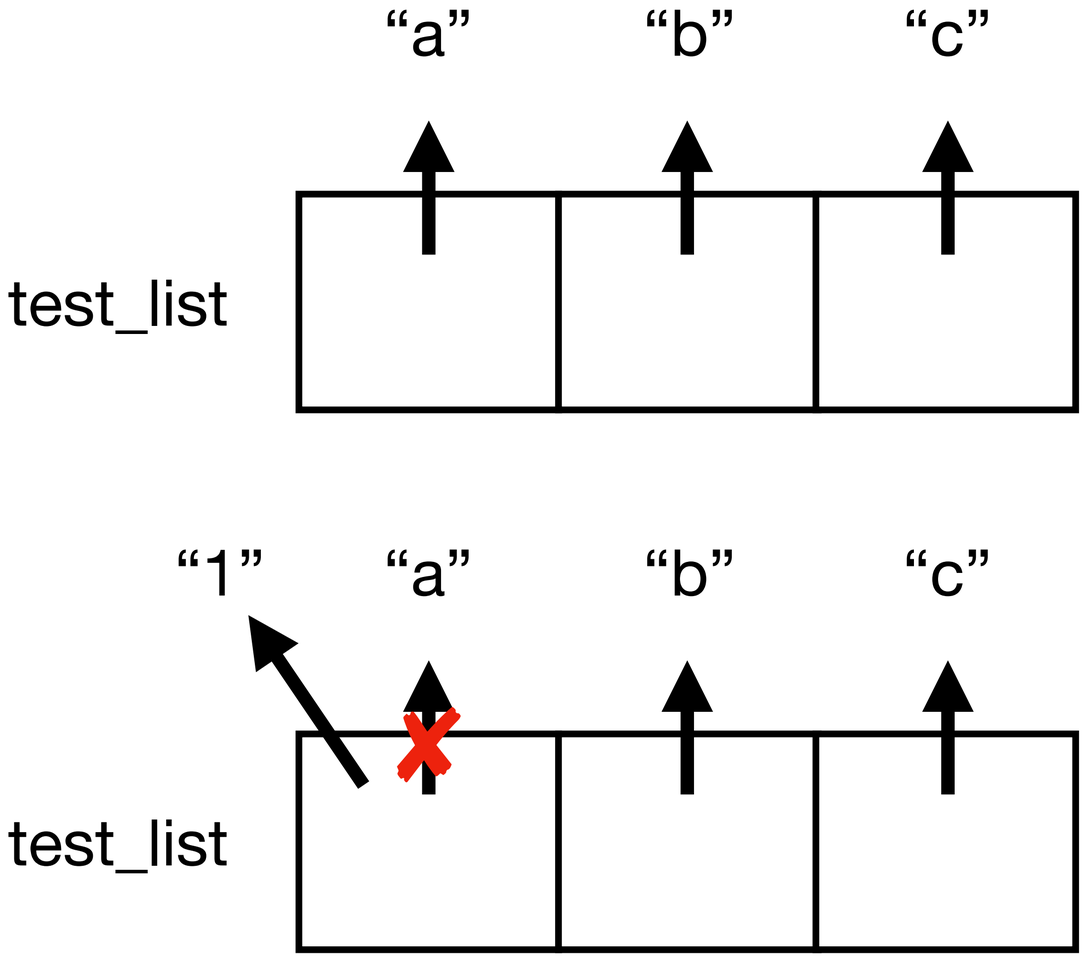
Lists…

```
test_list = [“a”, “b”, “c” ]
test_list[0] = “1”
print(test_list)
```

What will happen?
In either scenario, will the print statement get executed?
If yes, why?

Box-and-Pointer Diagrams

```
test_list = [“a”, “b”, “c” ]
test_list[0] = “1”
print(test_list)
```



Activity #1

```
test_list = [“a”, “b”, “c” ]
new_list =[ “a”, “b”, “c” ]
new_list[0] = test_list
print(new_list)
```

What do you think will be printed?

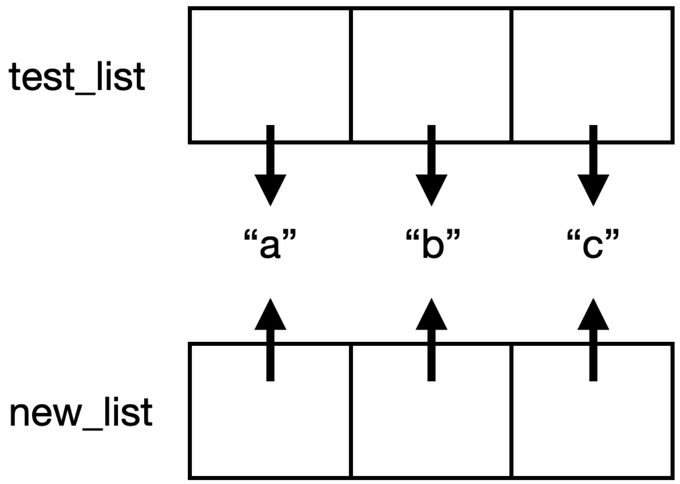
Box-and-Pointer Diagrams

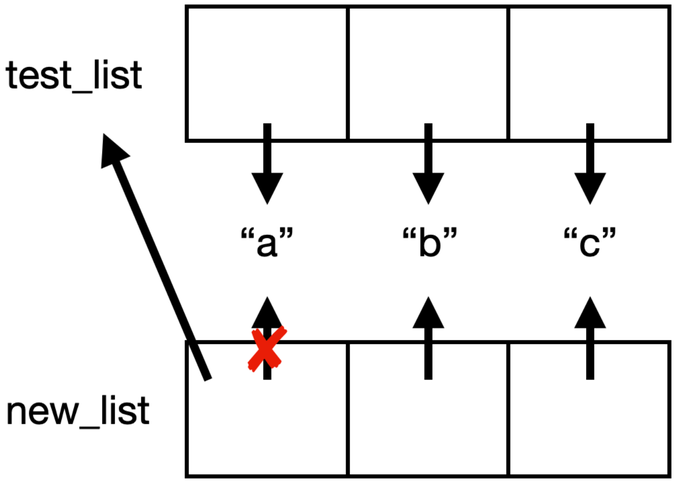
test_list = [“a”, “b”, “c”]
new_list =[“a”, “b”, “c”]
new_list[0] = test_list
print(new_list)

=> [test_list, “b”, “c”]

Therefore, output will be:

=> [[“a”, “b”, “c”], “b”, “c”]





Challenge Yourself: What will happen if test_list = new_list?

Control-Flow & Scope of Variables

When working inside functions or loops…

1. Has this variable been defined before?
2. Am I modifying what I want to modify?

Control-Flow & Scope of Variables

Global Scope

```
a = “I am the global scope”
b = [1,2,3,4]

function1:
    a = 1
    print(a)
    b.append(5)

function2:
    print(a)
    print(b)

function1()
function2()
```

What will be printed?

Can we break down the problem?

Control-Flow & Scope of Variables

Global Scope

```
a = “I am the global scope”
b = [1,2,3,4]

function1:
    a = 1
    print(a)
    b.append(5)

function2:
    print(a)
    print(b)

function1()
function2()
```

Expected Output:

=> 1

=> I am the global scope

=> [1, 2, 3, 4, 5]

Coding Practices

Try approaching these exercises without writing code.

Use pen and paper!

Given a string, check if it is a palindrome.

Sample Output

s2 = “racecar”

```
result = is_palindromic(s2)

print(result) # True
```

Hint: Strings and Lists are similar in some ways…

What is the output?

```
def sorry(x):
    return 2*x if x else “cool”

print(sorry(1))
print(sorry(“False”))
print(sorry([]))
print(sorry(1 != “1”))
print(sorry(3))
```

Hint: Strings and Lists are similar in some ways…

Given a string, create a dictionary whose keys are a letter and a values are the no. of times the letter occurs in the word.

Sample Output

```
word = “tomato”

characters = count_letters(word)

print(characters) # {“t”: 2, “o”: 2, “m”: 1, “a”: 1}
```

Given a word and a dictionary whose key, value pairs are letters and their occurrences, return True if they both describe the same word.

Sample Output

s1 = “alabama”
char = {“a”: 4, “l”: 1, “b”: 1, “m”: 1}

```
result = is_same(s1, char)

print(result) # True
```

Hint: Break the problem down, how do they relate?

Given two strings, string1 and string2, find out if string1 is a rotation of string2.

Sample Output

```
s1 = “erbottlewat”
s2 = “waterbottle”

result = is_rotation(s1, s2)

print(result) # True
```

If both strings are rotations of each other, how are they related?

Hint: You only need to use **is_substring** once.

Alex works at a clothing store. There is a large pile of socks that must be paired by colour for sale. Given a list of integers representing the colour of each sock, determine how many pairs of socks with matching colours there are.

Sample Output

```
socks = [1, 2, 1, 2, 1, 3, 2]

result = count_pairs(socks)

print(result) # 2
```

Hint: Can we keep track of one element while looking through the rest of the list?

An avid hiker keeps meticulous records of their hikes. For every step it was noted if it was an *uphill*, or a *downhill*, step. Hikes always start and end at sea level, and each step up or down represents a unit change in altitude.

- A *mountain* is a sequence of consecutive steps *above* sea level, starting with a step *up* from sea level and ending with a step *down* to sea level.

- A *valley* is a sequence of consecutive steps *below* sea level, starting with a step *down* from sea level and ending with a step *up* to sea level.

Given a string of up / down steps, how many valleys did he walk through?

Hint: Each step up is +1 and step down is -1.

Sample Output

```
path = “DDUUUUDD”

result = count_valleys(path)

print(result) # 1
```

What will this print?

```
a= [1, 2, 3]

b = a

a[0] = b

print(a is b)
```

Draw the box and pointer diagram!

Hint: “==“ checks for equality. The “is” operator checks if two objects are the same.
If two objects are the same, they must also be equal. But not the other way around.

Thank you