CS 1358 Introduction to Programming in Python

Fall Semester 2019

Prof. Pai H. Chou

Self-Check 6

Due Date: Sunday, October 20, 2019, 11:59pm

Answer the following questions to check your understanding of your material. Expect the same kind of questions to show up on your tests.

## 1. Definitions and Short Answers

1. Given the following if-statement  
   x = int(input('enter one number: '))  
   y = int(input('another number: '))  
   if y == 0:  
    print('sorry, cannot divide by 0!')  
   else:  
    print(f'{x} / {y} = {x/y}')  
   * What is the purpose of checking if y == 0?

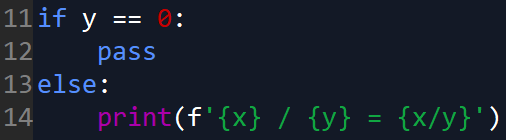
since it cannot divided by 0

* + What is the purpose of calling the int() function on the first two lines? Will the code still work if you replace the first two lines with  
    x = input('enter one number: ')  
    y = input('another number: ')  
    instead? Why or why not?

No, since the input is a string, in order to to the division, we need to call the int() function

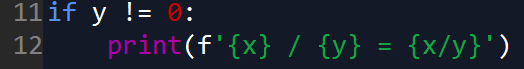
1. The following if-statements are not correct in Python syntax. What is wrong with them and how can they be fixed? Do not change the functionality.
   * if y == 0:  
      # do nothing  
     else:  
      print(f'{x} / {y} = {x/y}')

since Python does not allow skipping suite



* + if y != 0:  
     print(f'{x} / {y} = {x/y}')  
    else:  
     # do nothing

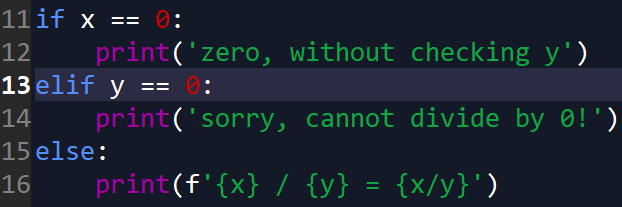
since Python does not allow skipping suite



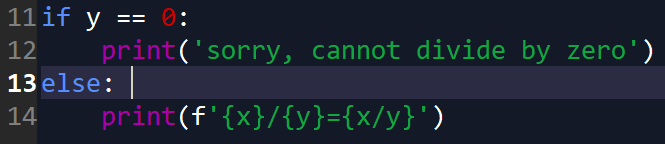
* + if x == 0:  
     print('zero, without checking y')  
    else if y == 0:  
     print('sorry, cannot divide by 0!')  
    else:  
     print(f'{x} / {y} = {x/y}')

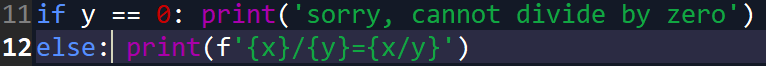
else if is an invalid syntax

must use elif → Python language rule



* + if y == 0: print('sorry, cannot divide by zero') else:   
     print(f'{x}/{y}={x/y}')  
    if, else can’t be at the same line





1. Given the dictionary  
   eng = {'one': 1, 'two': 2, 'three': 3, 'four': 4}  
   what is the **value** of the following expressions, and which ones cause **errors**?
   * 4 in eng

False

* + 'three' in eng

True

* + ('two', 2) in eng

False

* + {'one': 1} in eng

TypeError

* + eng['three']

3

* + eng[0]

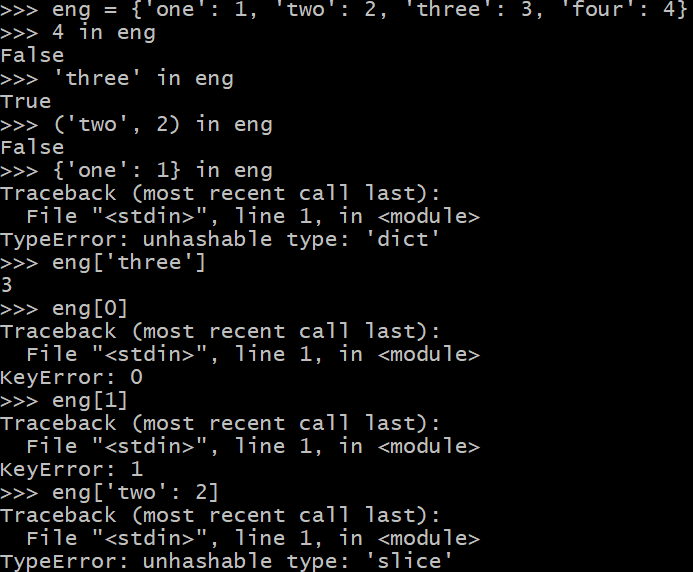
KeyError

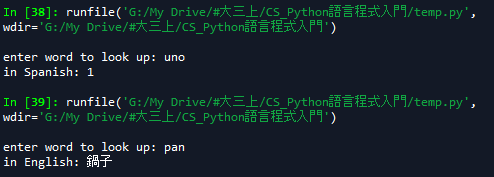
* + eng[1]

KeyError

* + eng['two':2]

TypeError



1. Consider the code:  
    1 english={'one':1,'pan':'鍋子''cabbage':'高麗菜','pie':'派餅'}   
    2 spanish = {'uno':1, 'pan':'麵包','col':'高麗菜', 'pie':'腳'}  
    3  french = {'une':1, 'toi':'你', 'col':'衣領', 'pie':'鵲'}  
    4 word = input('enter word to look up: ')  
    5  if word in english:  
     6 print(f'in English: {english[word]}')   
    7 elif word in spanish:   
    8 print(f'in Spanish: {spanish[word]}')  
    9  elif word in french:   
   10 print(f'in French: {french[word]}')   
   11 else:   
   12 print(f'{word} not found in dictionary')  
   When running the program, what happens when you type the following text when prompted?
   * uno

in Spanish: 1

* + pan

in English: 鍋子

* + toi

in French: 你

* + pie

in English: 派餅

* + col

in Spanish: 高麗菜

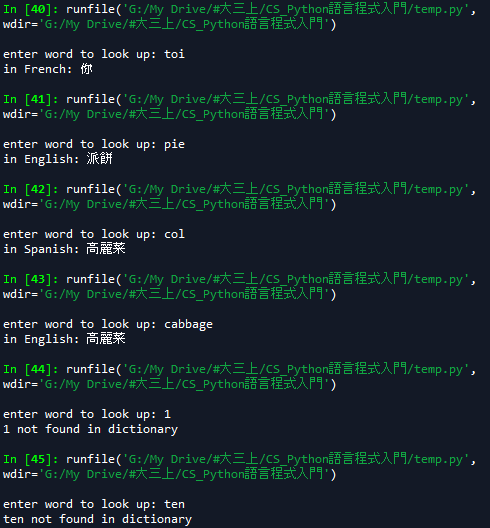
* + cabbage

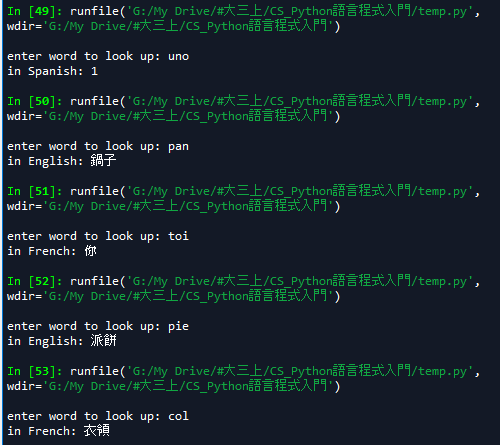
in English: 高麗菜

* + 1

1 not found in dictionary

* + ten

ten not found in dictionary  


1. In Question #4, if you swap lines 7-8 with lines 9-10, will the code output exactly the same results for all inputs or different for some inputs? Which?
   * uno

in Spanish: 1

* + pan

in English: 鍋子

* + toi

in French: 你

* + pie

in English: 派餅

* + col → different

in French: 衣領

* + cabbage

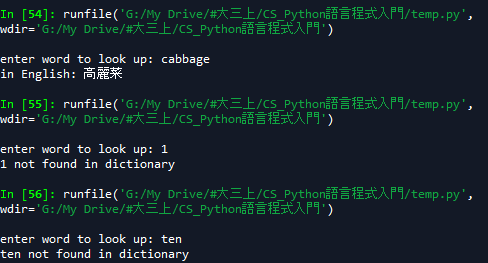
in English: 高麗菜

* + 1

1 not found in dictionary

* + ten

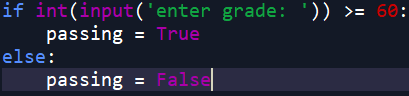
ten not found in dictionary



1. If the code in Question#4 is modified so all elif clauses get replaced by if, as shown in the following code:  
    5  if word in english:  
    6   print(f'in English: {english[word]}')   
    7 if word in spanish:   
    8 print(f'in Spanish: {spanish[word]}')  
    9  if word in french:   
   10 print(f'in French: {french[word]}')   
   11 else:   
   12 print(f'{word} not found in dictionary')  
     
   Suppose the user enters a word that is in english but not in french, does the print statement on the last line still get executed? Why or why not?  
   Yes, since the last line is matching to the if statement for french



1. Given the following code  
   passing = False  
   if int(input('enter grade: ')) >= 60:  
    passing = True
   * what is the equivalent statement that uses both if and else?



* + what is the equivalent (assignment) statement that does not use if at all?  
    

1. Consider the following version of the code  
    1 word = input('enter word to look up: ')  
    2 outList = []  
    3  if word in english:   
    4 outList.append(f'in English: {english[word]}')  
    5  if word in spanish:  
    6   outList.append(f'in Spanish: {spanish[word]}')  
    7  if word in french:   
    8 outList.append(f'in French: {french[word]}')  
    9 if not outList:  
   10   print(f'{word} not found in dictionary')   
   11 else:  
   12   print('\n'.join(outList))  
   * If line 9 is changed to if outList:, then how should the rest of the code be updated so it behaves exactly the same?

9 if outList:  
 10   print('\n'.join(outList))  
 11 else:  
 12   print(f'{word} not found in dictionary')

* + What does line 12 do?

output the result by separating it by 換行

* + Is line 2 really necessary?

Yes, since we have to initial the outList

* + What if line 2 is removed and line 4 is replaced by  
     outList = [f'in English: {english[word]}']  
    will the code still work the same way as before? if not, how can it be fixed by adding more code?

Yes

1. Convert the if-else statement in the previous question (lines 9-12) into a single print() with a **conditional expression** as its argument.

print(f'{word} not found in dictionary') if not outList \  
 else print('\n'.join(outList))

1. Convert the if-elif-elif-else statement with four suites in Question#4 (lines 5-12) into a single print() statement whose argument is a **conditional expression**.

print(f'in English: {english[word]}') if word in english \

else print(f'in Spanish: {spanish[word]}') if word in spanish \

else print(f'in French: {french[word]}') if word in French \

else print(f'{word} not found in dictionary')

1. Is there a form of conditional expression that uses the elif keyword? If so, provide an example.

No

1. Given the following nested if-statement  
   if x < 0:  
    if y > 3:  
    print('correct')  
   rewrite it as a single if statement by combining the two conditions.

if x < 0 and y > 3:

print('correct')

1. Given the following function that returns a boolean value:  
   def test(y):  
    if y % 10:  
    return False  
    else:  
    return True  
   rewrite it as a single return statement by eliminating the if-else construct completely.

def test(y):  
 return (y % 10 == 0)

1. Given the following function:  
   def fn(z):  
    if y % 10:  
    return True  
    if y > 400:  
    return False  
    return True  
   rewrite it as a single return statement by eliminating all if-constructs completely.

def fn(z):  
 return (y % 10 != 0) or not y > 400

1. Write a Python function to concatenate a list of strings **using a while loop**.  
   def concat\_str(L): # L is a list of strings  
    # initialize local variables:  
    # - an index into L to select one item at a time  
    # - a result variable initialized to empty string  
    while \_\_\_\_: # fill in your condition  
    # concatenate result with the next string in L  
    # increment the index variable  
    # return the result string

def concat\_str(L):

i = 0

output = ''

while i < len(L):

output = output + L[i]

i = i + 1

return output

1. Write a Python function to concatenate a list of strings **using a for loop**.  
   def concat\_str(L): # L is a list of strings  
    # initialize local variables:  
    # - the result variable initialized to empty string  
    for \_\_\_\_\_\_ : # get one string at a time from L  
    # concatenate result with string from for loop  
    # return the result string

def concat\_str(L):

i = 0

output = ''

for i in range(len(L):

output = output + L[i]

return output

1. Given the source code  
   1 def index(L, val):  
   2 i = 0  
   3 while i < len(L):  
   4 if L[i] == val:  
   5 break  
   6 i += 1  
   7 return i if i < len(L) else -1  
   Rewrite the code so that it does not use a break to exit the loop but still uses the same code as line 7 to return the i value. Hint: you need a separate "flag" variable that can take a True or False value; it should be initialized before the loop and is tested as part of the while-condition. The flag should be set to cause a break out of the loop. Be sure the i value is correct after exiting the loop.

def index(L, val):

i = 0

flag = 0

while i < len(L) and flag == 0:

if L[i] == val:

flag = 1

if flag == 0:

i += 1

return i if i < len(L) else -1

1. Convert the code from Problem #17 into a while-else construct (where the else clause is associated with the while construct rather than with an if clause)

def index(L, val):

i = 0

flag = 0

while i < len(L):

if L[i] == val:

break

i += 1

else:

return -1

return i

1. Convert the code from Problem #17 into a while True: on line 3 and convert the while condition into an if condition inside the loop.

def index(L, val):

i = 0

while True:

if i >= len(L):

break

if L[i] == val:

break

i += 1

return i if i < len(L) else -1

1. Referring to the same code as in Problem #17, what happens if you replace break on line 5 with continue? For example,
   * >>> L = "abaca"  
     >>> index(L, 'a')  
     Does the function return a value or go into an infinite loop? Why?  
     infinite loop

since when i == 0, the if statement was satisfied, then it will continue to the while loop, where i is still 0, thus it will do the same thing over and over, cause the infinite loop

* + >>> L = "abcde"  
    >>> index(L, 'z')  
    Does the function return a value or go into an infinite loop and why?  
    return -1  
    Try to predict the behavior of the code before you try running it to verify your answer.

1. In the code   
    1 def index(L, val):    
    2 i = 0   
    3 while i < len(L):   
    4 if L[i] == val:   
    5 break   
    6 i = i + 1   
    7 else:  
    8 return -1   
    9 return i  
   What causes the else suite (lines 7-8) to be executed?

when i > len(L)

If line 5 is executed, will line 8 be executed?

No, it won’t. When line 5 is executed, it will break the while-else loop and execute line 9

1. Given the code  
    1 ESdict = { 'one': 'uno', 'dos': 'two', 'three': 'tres'}  
    2 word = input('enter an English word:')  
    3 whileword != '':  
    4 if word in ESdict**:** 5 print(f'{word} in Spanish is {ESdict[word]}')  
    6else: 7 print(f'{word} not in dictionary')  
    8 word = input('enter an English word:')  
   However, there is redundancy because line 2 and line 8 are exactly the same (except for indentation). How can this code be rewritten to eliminate this redundancy?

ESdict = { 'one': 'uno', 'dos': 'two', 'three': 'tres'}

word = input('enter an English word:')

while True:

ifword != '':

break

if word in ESdict**:**

print(f'{word} in Spanish is {ESdict[word]}')

else:

print(f'{word} not in dictionary')

1. Given the following code  
    1 defStackInterpreter():  
    2 L = []  
    3 while True:  
    4 line = input('command? ')  
    5 words = line.split()  
    6 if len(words) == 0:  
    7 pass  
    8 elif words[0] == 'show':  
    9 print(L)  
   10 elif words[0] == 'push':  
   11 L.extend(words[1:])  
   12 elif words[0] == 'pop':  
   13 print(L.pop())  
   14 elif words[0] == 'quit':  
   15 break  
   16 else:  
   17 print('unknown command')

If the elif keywords on lines 8, 10, 12, 14 are replaced by if, will the code still behave the same way as far as the user is concerned? Why?

No,

1. Using the same code from the previous problem, line 7 can be replaced by a single-keyword statement and the code still behaves the same way to the user. What is the keyword? Explain.

continue jumps to top of loop to test while condition

1. Using the same code from the previous problem, line 15 can be replaced by a single-keyword statement and the code still behave the same way. What is it?

return it will leave the function, just like break the while loop and end the function

1. Given the code  
    1 defStackInterpreter():  
    2 L = []  
    3 while True:  
    4 line = input('command? ')  
    5 words = line.split()  
    6 if len(words) == 0:  
    7 continue  
    8 if words[0] == 'show':  
    9 print(L)  
   10 continue  
   11 if words[0] == 'push':  
   12 L.extend(words[1:])  
   13 continue  
   14 if words[0] == 'pop':  
   15 print(L.pop())  
   16 continue  
   17 if words[0] == 'quit':  
   18 break  
   19 print('unknown command')  
   Why doesn't line 19 need to be inside an else-clause but can be an unconditional statement at the same level as all preceding if-statements?

If it meets each if statement, it will jump to top of loop to test while condition, and when it doesn’t meet each if statement, that’s when it will print(‘unknown’)

1. Given the Python code  
    1 def product(L):  
    2 p = 1  
    3 for i in range(len(L)):  
    4 p = p \* L[i]  
    5 return p
   * Rewrite it as a while-loop

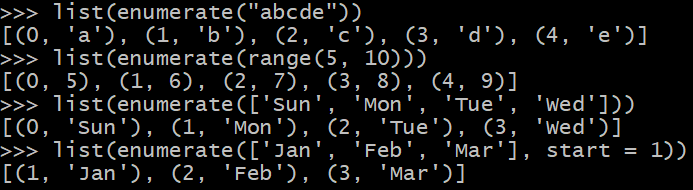
def product(L):  
 p = 1  
 while i < len(L):  
 p = p \* L[i]

i = i + 1  
 return p

* + Rewrite it as a Pythonic for-loop

def product(L):  
 p = 1  
 for n in L:  
 p = p \* n  
 return p

1. What is the value of the expression
   * list(enumerate("abcde"))
   * list(enumerate(range(5, 10)))
   * list(enumerate(['Sun', 'Mon', 'Tue', 'Wed']))
   * list(enumerate(['Jan', 'Feb', 'Mar'], start=1))



1. Rewrite the following code using a Pythonic for-loop:  
    1 def find(L, val):  
    2 i = 0  
    3 while i < len(L):  
    4 if L[i] == val:  
    5 return i  
    6 i = i + 1  
    7 return -1

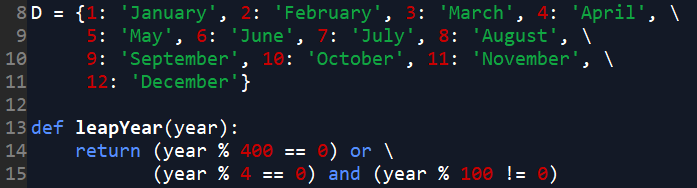
def find(L, val):  
 i = 0  
 while i < len(L):  
 if L[i] == val:  
 return i  
 i = i + 1  
 return -1

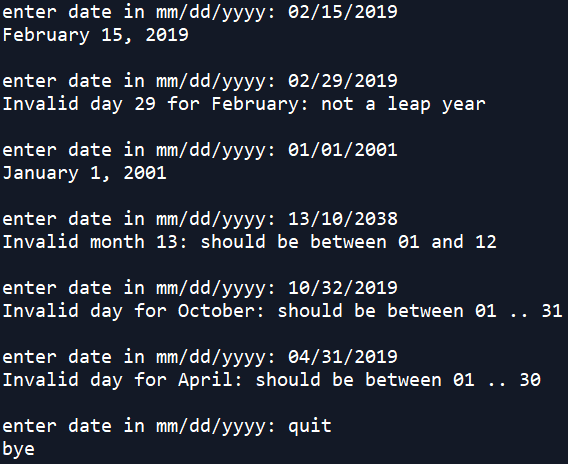
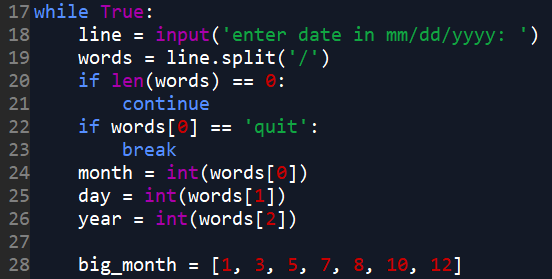
## 2. Programming Exercises

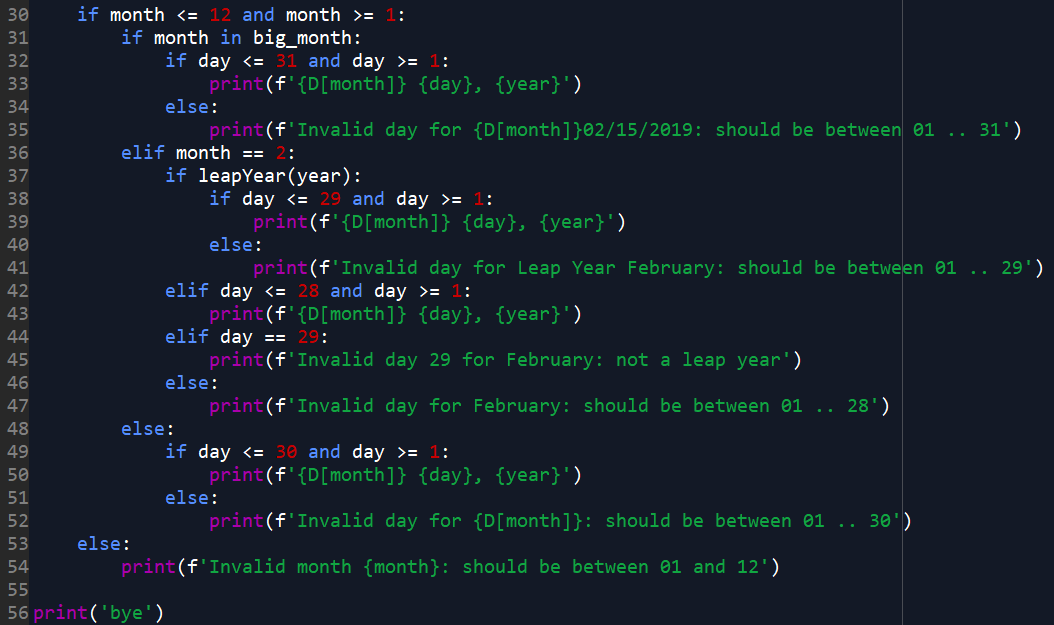
1. Write a Python program named **dateconv.py** to prompt the user for date in US format of mm/dd/yyyy (e.g., '02/15/2019' and outputs it in the full format of 'February 15, 2019'. You may assume the dates are given as decimal literals separated by '/', but you should also check if the dates are in range. If the dates are out of range, then you should inform the user what is wrong. For instance  
   $ python3 dateconv.py  
   enter date in mm/dd/yyyy: 02/15/2019  
   February 15, 2019  
   enter date in mm/dd/yyyy: 02/29/2019  
   Invalidate day 29 for February: not a leap year  
   enter date in mm/dd/yyyy: 01/01/2001  
   January 1, 2001  
   enter date in mm/dd/yyyy: 13/10/2038  
   Invalid month 13: should be between 01 and 12  
   enter date in mm/dd/yyyy: 10/32/2019  
   Invalid day for October: should be between 01..31  
   enter date in mm/dd/yyyy: 04/31/2019  
   Invalid day for April: should be between 01..30  
   enter date in mm/dd/yyyy: quit  
   bye  
   $ \_

Hin: Build up your program one piece at a time. First, simply use a loop to prompt for input and check for 'quit'. If not quit then first use a print statement to check if the input is as you expect. Once confirmed, you can use str's split() method to split a string by its separator ('/' in this case). Then, you can check the array of strings, assuming you can convert them to int, and check if each one is within range. Month can be checked easily. Day depends on the month, and for the month of February, day also depends on whether the year is a leap year (28 or 29 days). If any field is invalid, then report the error and loop again. If all fields are correct then display the full month string.

You may store the names of the months all in a list indexed by the month number or in a dictionary keyed by the month. You may use the leap year function from a previous lecture.







## 3. Small Programming Project

1. [somewhat challenging] Write a Python program named long\_multiply(a, b) to return a string that shows the steps in a long multiplication. (note: a is called the *multiplier*, and b is the *multiplicand*) For example  
   >>> print(long\_multiply(12, 34))  
    12  
   x)34  
   ----  
    48  
    36   
   ----  
    408

Hint: You should construct the return value of the function by concatenating different strings together. It is probably easier if you just append the strings into a list and join them by '\n'.join(*listOfStrings*).

1. Before you join the strings, you need to determine the **width** to format each line. This is the maximum number of characters needed to represent the multiplier, multiplicand + 2 (because 'x)' takes two positions), and the product.
2. Create the following strings and append each one into a list:
   1. string for the multiplier with the width (right aligned),
   2. 'x)' concatenated the multiplicand formatted with width-2 (also right-aligned),
   3. string of '-' repeated for width times
   4. loop over the partial products of multiplier \* each digit of the multiplicand, but each time shifted one position to the left.
   5. anotherr string of '-' repeated for width times
   6. finally, the product as a string, formatted to the same width, right aligned.
3. finally, return '\n'.join( *list of strings created in above steps* )

Note: you may assume a and b are both nonnegative integers. Your code should work with integers of any number of digits.