**IF100 – Fall 2019-2020**

**Take-Home Exam #5**

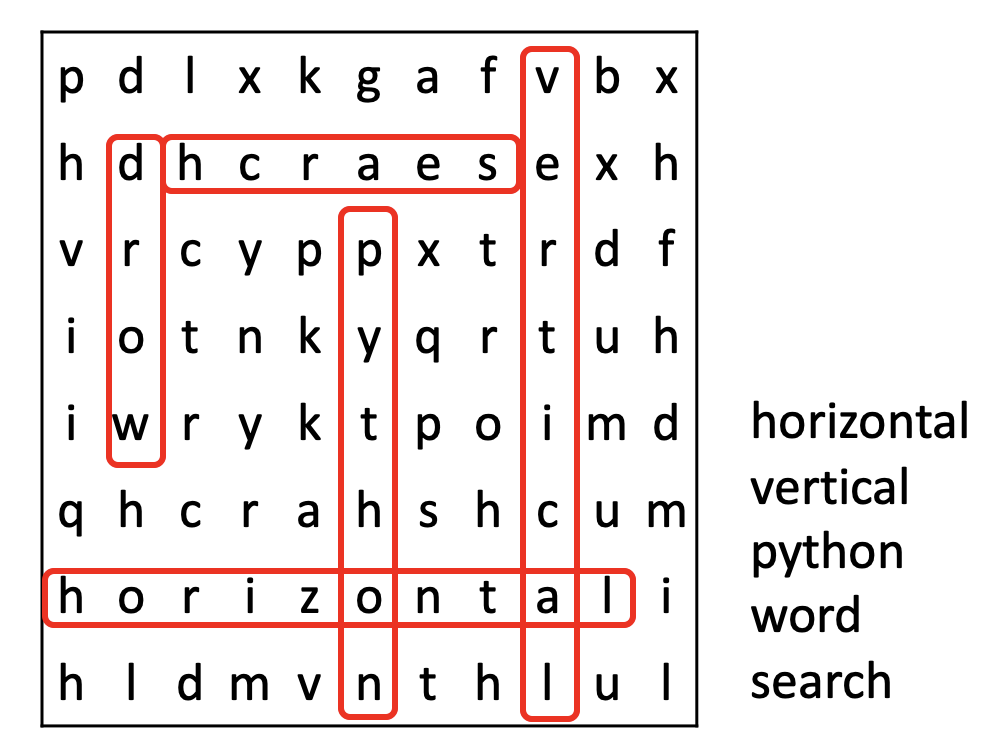
**Due December 13rd, Friday, 23:55 (Sharp Deadline)**

**Introduction**

The aim of this take-home exam is to practice on functions. You are given a main program that uses some functions and for the program to work correctly, you should implement those functions. Therefore, you cannot finish this take-home exam without using functions.   
  
Please note that a different main program will be used for grading purposes. If you modify the one given with the take-home exam documentation, then your grade will be affected with respect to the severity of the modification carried out. Therefore, do **not** modify the given main program. However, you still need to upload the entire solution while submitting your take-home exam. That is, your submission should include the function implementations, and also the main program that is provided together with the take-home exam documentation.

**Description**

A word search puzzle is a word puzzle that consists of letters placed as a grid, as shown below in Figure 1. The aim is to find some given words in this puzzle. The words can be found in different orientations. For this take-home exam, the words can be placed both horizontally and vertically. In both of these, there are two possible directions. Horizontally placed words can be found *left to right* or *right to left* and vertically placed words can be found *top to bottom* or *bottom to top*.



*Figure 1. An example word search puzzle and its solution*

In this take-home exam, you are given a main program, which aims to solve a word search puzzle. This main program uses some functions, which are not provided. Your aim is to implement these functions, so that the program can work correctly.

Here is the flow of the program to be implemented via functions: The program first gets two inputs, which are the puzzle and the words it will search for. Then, it displays the puzzle as a grid. Finally, it searches for the words in the puzzle and displays their starting and ending positions. Please see the next section for a more detailed explanation.

*You should not change the cell that contains the main program in the provided file. We will use another main program for grading purposes. Therefore, any change on the given template main program may (and possibly will) result in grade reduction.*

**Inputs, Outputs and Functions**

The inputs of the program and their order are explained below. It is extremely important to follow this order with the same format since we automatically process your programs. Also, prompts of the input statements to be used has to be exactly the same as the prompts of the "Sample Runs". ***Thus, your work will be graded as 0 unless the order is entirely correct***.

You must implement four functions to make the given main program work. These functions are explained in this section together with the inputs and outputs. You must have those functions in your program but if you want, you are also welcome to use some other additional functions.

The program will have two inputs: the puzzle and the words.

You will get these inputs from the user using the *getInput()* function. This function has exactly one parameter of type string. When the value of this parameter is "puzzle", your program should ask for the puzzle; and when it is "words", your program should ask for the words to be searched. This function should return the value of the input obtained from the user.

The first input to the program, the puzzle, will have the following format:

***row1-row2-row3-row4-...-rowN***

As you can notice, each row on the puzzle is separated by a dash ("-") character. Once your program gets this input from the user, it needs to check the validity of this input. For this purpose, the *checkInput()* function will be used. This function takes a single parameter, the puzzle, which is the first input obtained from the user, and returns True if it is in the correct format, and False otherwise. Note that True and False are boolean values.

This function should check the followings:

* The input should contain at least one dash ("-") character.
* Each row should consist only of alphabetical characters.
* All rows should have the same length.

The program will ask for the first input until it is entered correctly. It means that the user may need to enter the first input more than once.

When the first input is valid, your program will get the second input, the words to be searched in the puzzle, again using the *getInput()* function. This input will have the following format:

***word1,word2,word3,word4,...,wordM***

As you can see, each word is separated by a comma. Your program does not need to perform any checks on this second input. You can assume that the user will always enter this input correctly and the words will be at least 3 letters long.

After the program gets these two inputs from the user, it will print the puzzle as a matrix. For this purpose, the *printPuzzle()* function is used. This function gets the puzzle string as its only parameter, and prints each row in a separate line. For example, if the puzzle is ***row1-row2-row3-row4-...-rowN,*** then this function should print the following:

*row1*

*row2*

*row3*

*row4*

*...*

*rowN*

Finally, the program will search for the words in the puzzle, in the given order, using the function *findWords()*. This function takes two parameters (puzzle and words), and it prints the location of each word that can be found in the given puzzle. You can assume that each word appearing in the second input entered by the user will occur at most once in the puzzle given as the first input by the user. If your program can find a word, it should print its starting position and ending position. For example, if ***word1*** is found in the given puzzle, then your program should print the following for that word:

*Found* ***word1*** *at (****starting\_row****,****starting\_column****)-(****ending\_row****,****ending\_column****)*

If a ***word2*** does not exist in the given puzzle, then your program should display the following prompt:

***word2*** does not exist.

The row and column indices should start from 0. The top row and the leftmost column will have the index 0.

You cannot make any assumptions on the capitalization of the inputs, but you must print them all in lowercase.

Please see "Sample Runs" section for some examples.

**Sample Runs**

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are inputs taken from the user. You have to display the required information in the same order and with the same words and characters as below.

**Sample Run 1 (**Example from the figure, correct input format**)**

Please enter the puzzle: ***pdlxkgafvbx-hdhcraesexh-vrcyppxtrdf-iotnkyqrtuh-iwryktpoimd-qhcrahshcum-horizontali-hldmvnthlul***

Please enter the words: ***python,vertical,horizontal,word,search***

pdlxkgafvbx

hdhcraesexh

vrcyppxtrdf

iotnkyqrtuh

iwryktpoimd

qhcrahshcum

horizontali

hldmvnthlul

Found python at (2,5)-(7,5)

Found vertical at (0,8)-(7,8)

Found horizontal at (6,0)-(6,9)

Found word at (4,1)-(1,1)

Found search at (1,7)-(1,2)

**Sample Run 2 (**Only horizontal, uppercase-lowercase mixed input**)**

Please enter the puzzle: ***MPOOLMPTCGY-ABSTRACTION-vnoiTCNUFPU-ALGOritHMSD-ZPSeuDOCODE-RGNIRTSLIST-hooxchacler-svariablewg***

Please enter the words: ***function,pseudocode,LOOP,abstraction,stRIng,algorithm,VARiable,list***

mpoolmptcgy

abstraction

vnoitcnufpu

algorithmsd

zpseudocode

rgnirtslist

hooxchacler

svariablewg

Found function at (2,8)-(2,1)

Found pseudocode at (4,1)-(4,10)

Found loop at (0,4)-(0,1)

Found abstraction at (1,0)-(1,10)

Found string at (5,6)-(5,1)

Found algorithm at (3,0)-(3,8)

Found variable at (7,1)-(7,8)

Found list at (5,7)-(5,10)

**Sample Run 3 (**Only vertical, uppercase input**)**

Please enter the puzzle: ***LVNMEFOPEA-IOKLBXRCJR-MTEOIBKAJA-ATIHCESNVK-BABKAINBIN-OWOCIJIEEA-YARORIMRNR-KLITONLRNO-OYASEGIAAM-TVNAERYLCE***

Please enter the words: ***OTTAWA,VIENNA,ANKARA,BEIJING,NAIROBI,ROME,MINSK,CANBERRA,LIMA,TOKYO,CAIRO,STOCKHOLM***

lvnmefopea

ioklbxrcjr

mteoibkaja

atihcesnvk

babkainbin

owocijieea

yarorimrnr

klitonlrno

oyasegiaam

tvnaerylce

Found ottawa at (1,1)-(6,1)

Found vienna at (3,8)-(8,8)

Found ankara at (5,9)-(0,9)

Found beijing at (2,5)-(8,5)

Found nairobi at (9,2)-(3,2)

Found rome at (6,9)-(9,9)

Found minsk at (6,6)-(2,6)

Found canberra at (1,7)-(8,7)

Found lima at (0,0)-(3,0)

Found tokyo at (9,0)-(5,0)

Found cairo at (3,4)-(7,4)

Found stockholm at (8,3)-(0,3)

**Sample Run 4 (**Horizontal and vertical with wrong inputs**)**

Please enter the puzzle: ***yngejzfocox-jflapbenorangehksy-ydnuarqfuiishcaepbhqkhq-shdbryjlicotknhbmmaz***

Wrong input format.

Please enter the puzzle: ***yngejzfocoxxmwhqpeue***

Wrong input format.

Please enter the puzzle: ***yngejzfoco45mwhqpe1e-jflap34nora5gehkklce***

Wrong input format.

Please enter the puzzle: ***yngejzfocoxxmwhqpeue-jflapbenorangehkklce-nifiiugdcsbfmscasqex-sucynzsgoynnesynblbf-ggfqefrnnyrrebwartsy-ydnuarqfuiimspknwanz-gfwypdbmtykhyiyazzyw-qgcfpearirxhrczbsies-okjvlphnkrrnmorlowcj-grapehhvsebfzvzfjief-jzipwvnhshcaepbhqkhq-shdbryjlicotknhbmmaz***

Please enter the words: ***pear,banana,orange,pineapple,grape,cherry,peach,strawberry,coconut,kiwi***

yngejzfocoxxmwhqpeue

jflapbenorangehkklce

nifiiugdcsbfmscasqex

sucynzsgoynnesynblbf

ggfqefrnnyrrebwartsy

ydnuarqfuiimspknwanz

gfwypdbmtykhyiyazzyw

qgcfpearirxhrczbsies

okjvlphnkrrnmorlowcj

grapehhvsebfzvzfjief

jzipwvnhshcaepbhqkhq

shdbryjlicotknhbmmaz

Found pear at (7,4)-(7,7)

Found banana at (7,15)-(2,15)

Found orange at (1,8)-(1,13)

Found pineapple at (1,4)-(9,4)

Found grape at (9,0)-(9,4)

Found cherry at (11,9)-(6,9)

Found peach at (10,13)-(10,9)

Found strawberry at (4,18)-(4,9)

Found coconut at (0,8)-(6,8)

Found kiwi at (10,17)-(7,17)

**Sample Run 5 (**Horizontal and vertical with some words not in puzzle**)**

Please enter the puzzle: ***pandaxngeqrymtso-ezmlaesowxaqbujl-noilktxreecytrql-gskaboofsfoxdtei-utsmakotufodhlrd-iroachimpanzeesa-nintrwflyrkhcdum-jcecahkktiklsvhr-mhvsbaykagodwgca-koalatcwlkfmrwbb-jsrrfdolphinuytc***

Please enter the words: ***platypus,dolphin,penguin,unicorn,armadillo,seal,dog,giraffe,panda,racoon,dragon,frog,kraken,ostrich,llama,lion,chimpanzee,phoenix,turtle,raven,fox,koala,crab***

pandaxngeqrymtso

ezmlaesowxaqbujl

noilktxreecytrql

gskaboofsfoxdtei

utsmakotufodhlrd

iroachimpanzeesa

nintrwflyrkhcdum

jcecahkktiklsvhr

mhvsbaykagodwgca

koalatcwlkfmrwbb

jsrrfdolphinuytc

Found platypus at (10,8)-(3,8)

Found dolphin at (10,5)-(10,11)

Found penguin at (0,0)-(6,0)

unicorn does not exist.

Found armadillo at (8,15)-(0,15)

Found seal at (1,6)-(1,3)

Found dog at (8,11)-(8,9)

Found giraffe at (8,9)-(2,9)

Found panda at (0,0)-(0,4)

Found racoon at (0,10)-(5,10)

dragon does not exist.

Found frog at (3,7)-(0,7)

kraken does not exist.

Found ostrich at (2,1)-(8,1)

Found llama at (1,3)-(5,3)

Found lion at (2,3)-(2,0)

Found chimpanzee at (5,4)-(5,13)

phoenix does not exist.

Found turtle at (0,13)-(5,13)

Found raven at (10,2)-(6,2)

Found fox at (3,9)-(3,11)

Found koala at (9,0)-(9,4)

Found crab at (5,4)-(8,4)

**How to get help?**

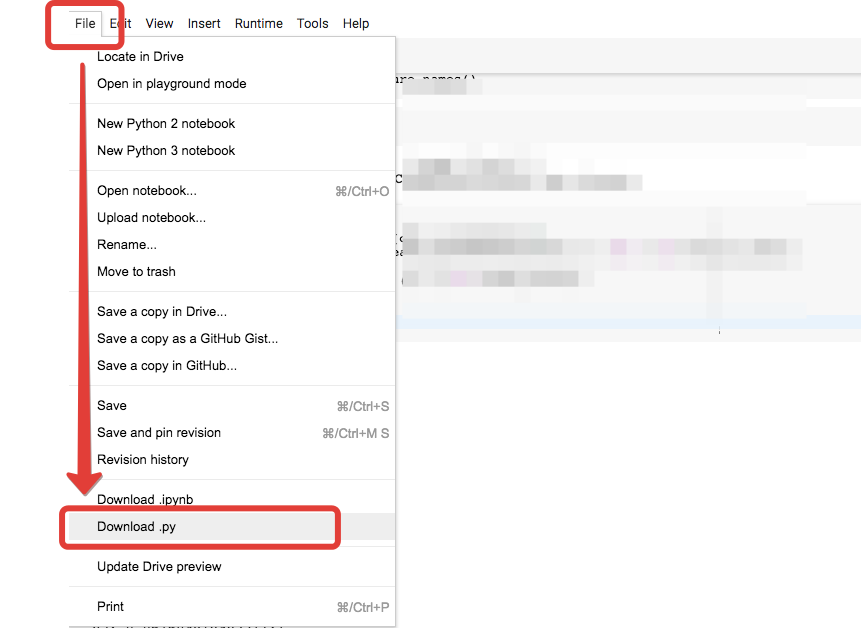
You can use GradeChecker (<http://sky.sabanciuniv.edu:8080/GradeChecker/>) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

**What and where to submit?**

You should prepare (or at least test) your program using Python 3.x.x. We will use Python 3.x.x while testing your homework.

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your name and last name is "İnanç Arın", and if you want to write it as comment; then you must type it as follows:  
 *# Inanc Arin*

Submission guidelines are below. Since the grading process will be automatic, students are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be 0.

* Download your code as *py* file with "File" -> "*Download .py*" as below:  
    
  
* Name your *py* file that contains your program as follows:  
     
   "**username\_hw5.py**"   
    
  For example: if your SuCourse username is "**duygukaltop**", then the name of the *py* file should be: **duygukaltop\_hw5*.py*** (please only use lowercase letters).
* Please make sure that this file is the latest version of your homework program.
* Submit your work **through SUCourse only**! You can use the GradeChecker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse.

**General Homework Rules**

* Successful submission is one of the requirements of the homework. If, for some reason, you cannot successfully submit your homework and we cannot grade it, your grade will be 0.
* There is NO late submission. You need to submit your homework before the deadline. Please be careful that SUCourse time and your computer time may have a 1-2 minutes differences. You need to take this time difference into consideration.
* Do NOT submit your homework via email or in hardcopy! SUCourse is the only way that you can submit your homework.
* If your code does not work because of a syntax error, then we cannot grade it; and thus, your grade will be 0.
* Please do submit your **own** work only. It is really easy to find out "similar" programs!
* Plagiarism will not be tolerated. Please check our plagiarism policy given in the syllabus of the course.

Good luck!