Information

The exam consists of two parts, part A and part B.

- For grade 3 you need 9/16 in part A.
- For grade 4 you need grade 3 and 4/14 in part B.
- For grade 5 you need grade 3 and 10/14 in part B.

Part B is only graded if you passed part A.

Resources
From the computers in the exam hall, you have access to:

- Course web pages in English (and Swedish)
- Online programming environments: myCompiler, Judge0, GDB Online, Online Python, and Programiz.
  All of these contain an editor and a shell window where you can write Python code and test it. Use whichever environment you prefer.
- Python documentation

These resources are available from the tabs at the bottom of Inspera.

Since you have access to a programming environment, you can test your code before you submit it in Inspera. Copy your code from the online environment to Inspera. Select your code with CTRL-A, copy it with CTRL-C, and paste it into Inspera with CTRL-V. You undo with CTRL-Z.

During the exam, your answers are automatically saved by Inspera every 15 seconds. Make sure to copy code from the online IDE to Inspera frequently, to minimize the risk of losing unsaved work. After submitting your exam, you cannot access it again.

Assessment
In part A, your code must function according to the specifications. Non-functioning code will almost always be awarded 0 points.

In part B, non-functioning code may be awarded some points if it demonstrates that you have solved part of the problem.

The code in Part A does not need to be commented. In part B, use comments where necessary to increase the readability of your code.

If nothing else is mentioned, you are allowed to build on solutions to previous questions, even if those are not solved. It is permitted to introduce extra methods or functions. Expressions such as "write a function that" do not mean that you may not organise your solution into multiple functions. All questions concern the programming language Python and all code should be written in Python. The code should be readable, well structured, and indented. Names of variables, functions, methods, classes, etc, should be descriptive but still kept short.

Del A / Part A

Part A (Mandatory)

1 A1 (1 point): Multiple choice

The body of a function is given below, but the function header is missing. What should it look like?

```
new_lst = []
for el in lst:
    new_lst.append(el*scalar)
return new_lst
```

Select one alternative:

- ***def extend_list(lst, el):***
- ***def extend_list(lst, scalar):***
- ***def extend_list(lst, scalar, el):***
- ***def multiply_list_by_scalar(lst, el):***
- ***def multiply_list_by_scalar(lst, scalar):***
- ***def multiply_list_by_scalar(lst, scalar, el):***

Maximum marks: 1

2 A2 (1 point): Multiple choice

A class **Rectangle** is given below. We want to add a method **perimeter** that returns the rectangle's perimeter. What should it look like?

```
class Rectangle:

    def __init__(self, width, height):
        self.width = width
        self.height = height

    def __str__(self):
        return f'Rectangle of width {self.width} and height {self.height}.'

    def area(self):
        return self.width * self.height
```

Select one alternative:

-
```
def perimeter():
    return 2*width + 2*height
```

-
```
def perimeter(width, height):
    self.perimeter = 2*self.width + 2*self.height
```

-
```
def perimeter(self.width, self.height):
    return 2*self.width + 2*self.height
```

-
```
def perimeter(self):
    return 2*self.width + 2*self.height
```

-
```
def perimeter(self, width, height):
    perimeter = 2*self.width + 2*self.height
```

-
```
def perimeter(self):
    self.perimeter = 2*width + 2*height
```

Maximum marks: 1

3 A3 (1 point): Multiple choice

What would have been a suitable name for the function **new_function**?

```
def new_function(lst):
    new_lst = []
    for i in range(len(lst)):
        if i % 4 == 0:
            new_lst.append(lst[i])
    return new_lst
```

Select one alternative:

- ***divide_by_four***
- ***multiply_by_four***
- ***every_fourth_element***
- ***even_numbers***
- ***first_four_elements***
- ***last_four_elements***

Maximum marks: 1

4 A4 (1 point): Multiple choice

The program below contains an error that makes 2*3 appear as 33.

```
x = '3'
z = 2*x
x = int(x)
z = int(z)
print(f'2*{x} = {z}')
```

Which of the following changes can fix the program so that 2*3 = 6 is printed?

Select one alternative:

- Swap the positions of two of the lines.
- Replace **z** with **str(z)** in one place.
- Replace **int** with **float** in one place.
- Replace = with == in one place.

Maximum marks: 1

5 A5 (2 points): Average age

You have access to a dictionary with people's names (key) and their ages (value). Write a function **average_age(name_to_age, list_of_names)** that takes such a dictionary (**name_to_age**) and a list of names (list_of_names) as parameters. The function should return the average of the specified people's ages

For example the following code:

```
name_to_age = {'Adam': 17,
'Bertil': 74,
'Carina': 65,
'Denise': 18}

names = ['Adam', 'Carina', 'Bertil']
avg = average_age(name_to_age, names)

print('The average age of the following people: ')
for name in names:
    print(name, end=', ')
print(f'\nis {avg}.')
```

should give the output

```
The average age of the following people:
Adam, Carina, Bertil,
is 52.0.
```

Fill in your answer here

```
1
```

Maximum marks: 2

6 A6 (2 points): Next prime

Write a function ***next_prime(x)*** that, given an integer **x**, returns the smallest prime number that is strictly greater than **x.**

Hint: You can copy and use the function ***is_prime*** which is given below.

```
def is_prime(n):
    """Return True if n is a prime, otherwise False"""
    limit = int(n**0.5)
    for i in range(2, limit+1):
        if n % i == 0:
            return False
    return True
```

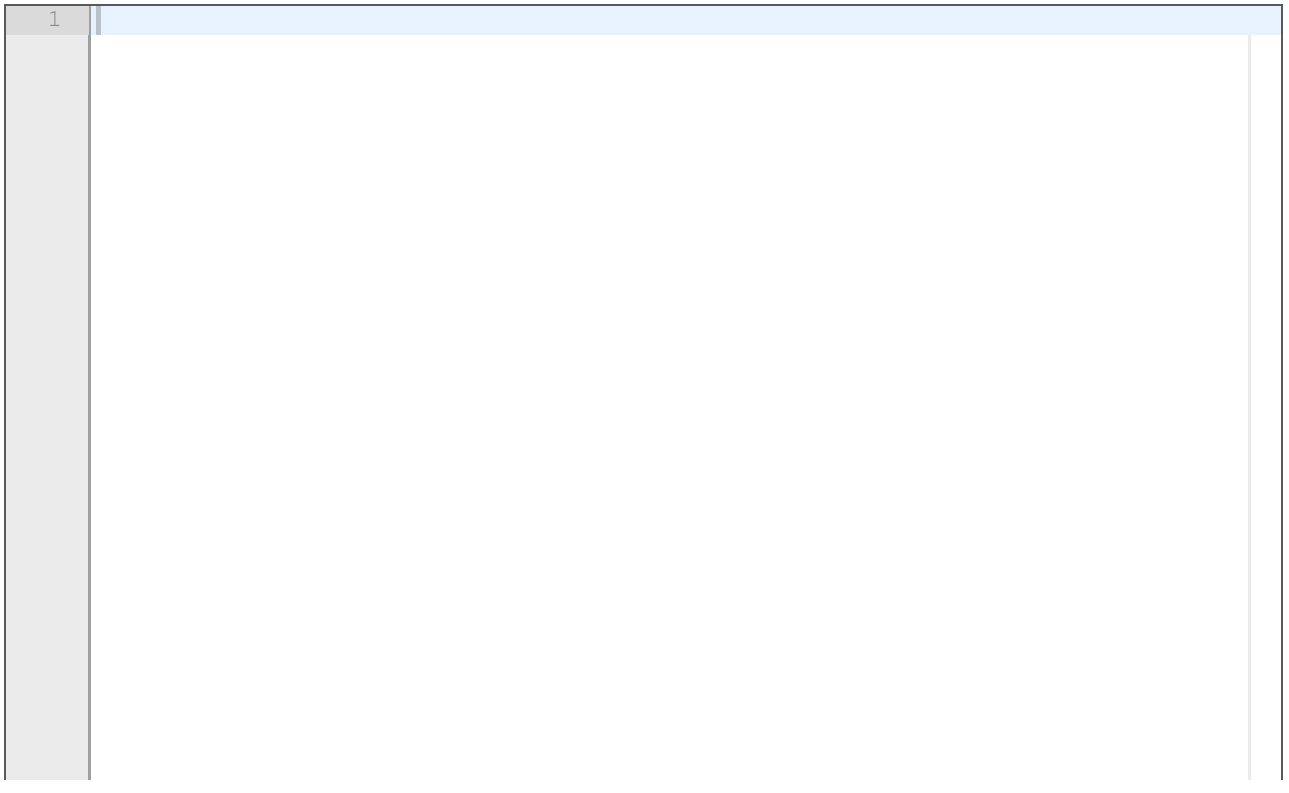
Following code:

```
x_values = [1,2,3,4,9,11,14]
for x in x_values:
    print(f'The smallest prime greater than {x} is {next_prime(x)}.')
```

should give the output

The smallest prime greater than 1 is 2.
The smallest prime greater than 2 is 3.
The smallest prime greater than 3 is 5.
The smallest prime greater than 4 is 5.
The smallest prime greater than 9 is 11.
The smallest prime greater than 11 is 13.
The smallest prime greater than 14 is 17.

Fill in your answer here



Maximum marks: 2

7 A7 (2 points): Round list of strings

Write a function ***round_list_of_strings(lst)*** that takes a list as a parameter. The list is assumed to contain decimal numbers stored as strings (see the example below). Your function should return a new list containing the same numbers, rounded to two decimal places. **The rounded numbers should also be stored as strings.**

Following code:

```
x = ['1.2145', '12.3198', '-138.4319']
x_rounded = round_list_of_strings(x)
print(f'x:          {x}')
print(f'x, rounded: {x_rounded}')
```

should give the output:

```
x:          ['1.2145', '12.3198', '-138.4319']
x, rounded: ['1.21', '12.32', '-138.43']
```

Hint: The call ***round(num, n)*** returns the number ***num*** rounded to ***n*** decimal places.

Fill in your answer here

```
1
```

Maximum marks: 2

8 A8 (2 points): Shuffle name

Write a function ***shuffle_name(name)*** that takes a first name (a string) as a parameter and returns a new version of the name, where the first letter has been removed and added to the end. No other letters have been moved. The new name should start with an uppercase letter, and the remaining letters should be lowercase.

The original name is assumed to meet the following criteria:

- It consists of at least two letters.
- The first letter is uppercase, and the others are lowercase.

The following code:

```
names = ['My', 'Jonas', 'Ibrohim']
for name in names:
    print(f'{name} -> {shuffle_name(name)}')
```

should give the output

My -> Ym
Jonas -> Onasj
Ibrohim -> Brohimi

Fill in your answer here

```
1
```

Maximum marks: 2

9 A9 (2 points): Bank account

Below is a class ***BankAccount*** defined. Unfortunately, a few minor errors have crept into the code. Your task is to correct these.

```
class BankAccount:

    def __init__(self, owner):
        self.owner = owner
        self.balance = 0

    def __str__(self):
        return f"{self.owner}'s bank account. Balance: self.balance SEK"

    def deposit(amount):
        """Add amount to the account balance"""
        self.balance += amount

    def withdraw(self, amount):
        """Remove amount from the account balance unless amount > balance"""
        if amount > balance:
            print('Error: Cannot withdraw an amount that exceeds the balance.')
        else:
            self.balance -= amount
```

The code needs to be corrected in a way that the following code

```
bills_account = BankAccount('Bill')
print(bills_account)
bills_account.deposit(100)
print(bills_account)
bills_account.withdraw(200)
print(bills_account)
bills_account.withdraw(40)
print(bills_account)
```

gives the output

```
Bill's bank account. Balance: 0 SEK
Bill's bank account. Balance: 100 SEK
Error: Cannot withdraw an amount that exceeds the balance.
Bill's bank account. Balance: 100 SEK
Bill's bank account. Balance: 60 SEK
```

You can get the class to work correctly by making minor changes to **three** of the lines, but you may make as many changes as you like as long as the final result works correctly.

Fill in your answer here

```
1
```

Maximum marks: 2

10 A10 (2 points): Extract elements

Write a function ***extract_elements(lst, indices_to_extract)*** that takes a list ***lst*** and a tuple ***indices_to_extract*** as parameters. The function should return a list of the elements in ***lst*** whose indices are specified in ***indices_to_extract***.

The following code:

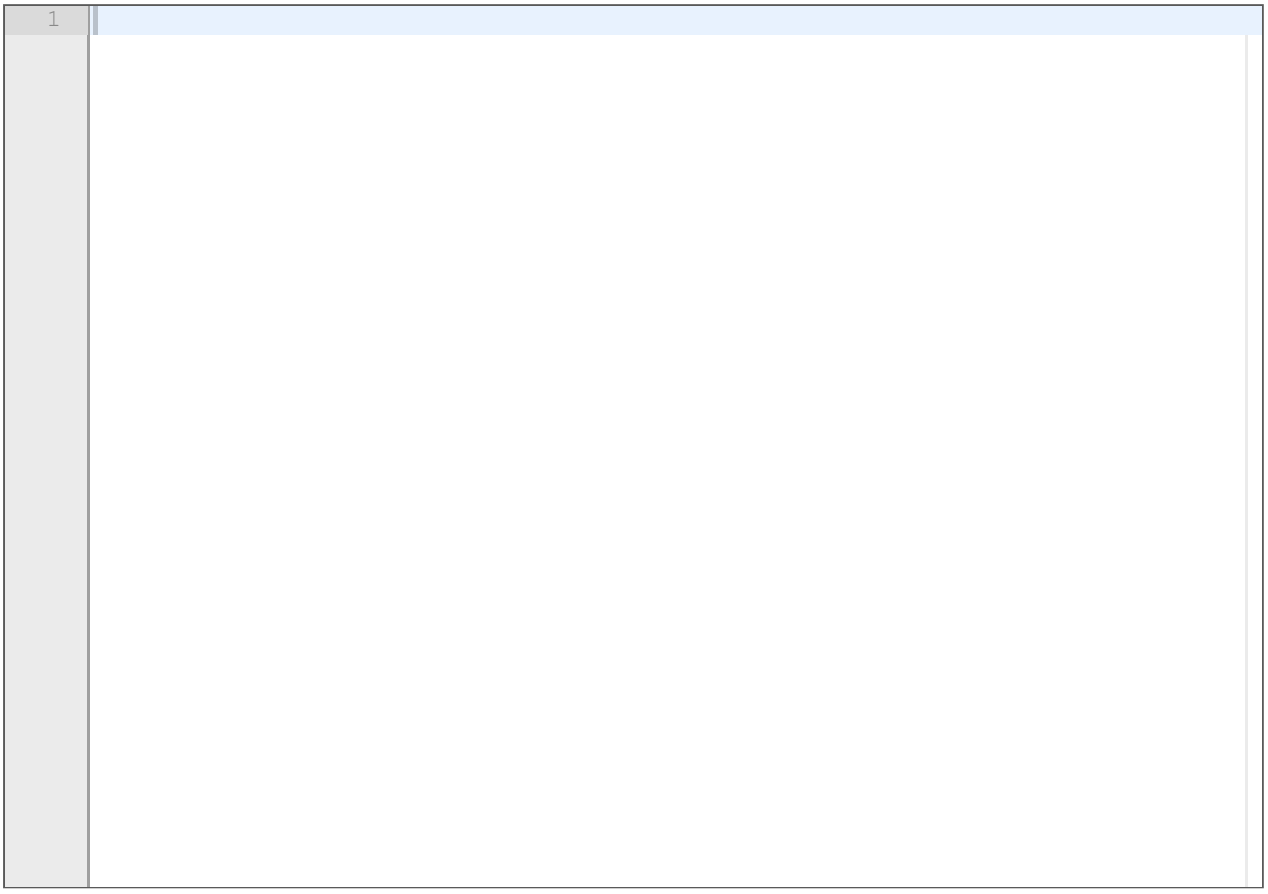
```
lst = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
ind = (0, 2, 4, 0, 1, 8, -1)
extracted = extract_elements(lst, ind)
print(f'lst: {lst}')
print(f'indices to extract: {ind}')
print(f'Extracted elements: {extracted}')
```

gives the output:

```
lst: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
indices to extract: (0, 2, 4, 0, 1, 8, -1)
Extracted elements: ['A', 'C', 'E', 'A', 'B', 'I', 'J']
```

Note that the tuple of indices is not necessarily sorted. The same index may occur multiple times; in that case, the corresponding element should appear multiple times in the extracted list (see index 0 in the example above).

Fill in your answer here



Maximum marks: 2

Del B / Part B

Part B - For higher grade (4 or 5)

For grade 4 you need to pass part A and score at least 4/14 points in part B.
For grade 5 you need to pass part A and score at least 10/14 points in part B.

11 B1 (4 points): Party planning

You are organizing a party and have asked all guests to fill out a Google form where they can answer the following questions:

- What would you like to eat? (fish/meat/vegan)
- Would you like to drink alcohol? (yes/no)

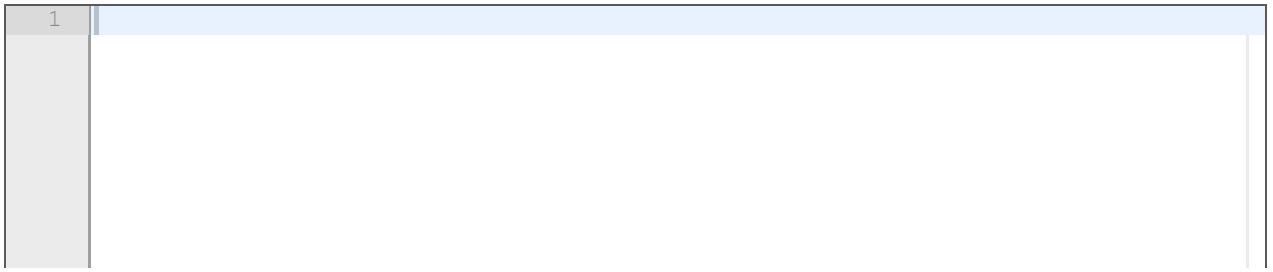
You can assume that all guests have selected one of the options, meaning no one has misspelled or forgotten to answer a question.
You can download the form responses as a dictionary, referred to as ***name_to_preferences***.
It contains all guests' names as keys and their responses as values. The responses are also stored in a dictionary. Below is an example with only two guests. Study the example to understand the structure.

You are trying to estimate how much the party will cost. You estimate that alcoholic beverages will cost 150 SEK per person who drinks alcohol (the cost of non-alcoholic drinks is disregarded). For the different food choices, you create a dictionary, ***food_to_cost***,
with the name of the food as the key and the estimated cost as the value (see the example below).

Your task is to write the function ***total_cost*** that calculates and returns the total cost for food and drink. When you are done, the code below should print the cost. Note that your function should work for any number of guests, different alcohol costs, and different food costs (i.e., different numbers in ***food_to_cost***).

```
name_to_preferences = {
   'Person 1':{
      'food':'vegan',
      'alcohol':'yes'
      },
   'Person 2':{
      'food':'fish',
      'alcohol':'no'
   }
}
food_to_cost = {
   'fish':250,
   'meat':300,
   'vegan':200
}
alcohol_cost = 150
print(total_cost(name_to_preferences, food_to_cost, alcohol_cost))
```

Fill in your answer here



Maximum marks: 4

12 B2 (10 points): Chess

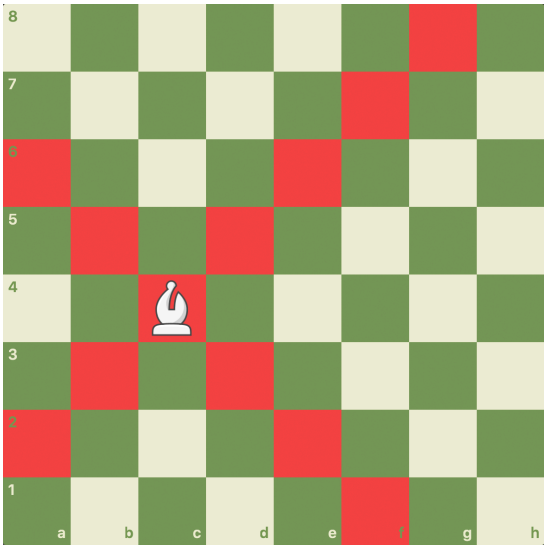
You are in the process of developing a chess computer and have started writing classes for two of the pieces: rook and bishop.

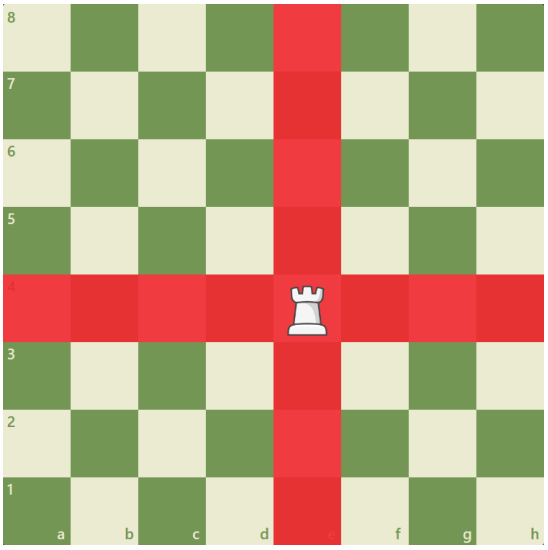
A chessboard consists of 8x8 squares. The rows of the board are numbered from 1 to 8, with 1 being the bottom row and 8 being the top row. Similarly, the board's columns are referred to as a, b, c, and so on. Column a is the far left, and column h is the far right. In chess notation, the square at the bottom left is called a1, the square at the top right is called h8, and so forth. The four central squares are d4, d5, e4, and e5. Note the lowercase letters and numbers at the bottom and to the left in the images.

In the code below, each square is represented by a tuple with two elements: a string for the column and an int for the row. For example, the square d3 is represented by the tuple **('d', 3)**.

Movement rules:

- A bishop can move any distance but only diagonally. See the figure on the left below.
- A rook can move any distance but only horizontally or vertically. See the figure on the right below.
- Moving a piece to the square it is already on is NOT a valid move.





We envision that the rook or bishop is the only piece on the board. You do not need to consider whether any other pieces are in the way.

Complete the following methods:

- ***all_legal_moves*** in the ***Bishop*** *Class*. The method should return a list of all the squares (remember that a 'square' is a tuple with two elements in the code!) that the bishop can potentially move to from the square it is currently on. **(4 points)**
- ***is_legal_move*** in the ***Rook*** *Class*. Read the documentation and the corresponding method in the ***Bishop*** class to understand how it should work. You can assume that the parameter ***new_square*** is one of the squares on the board, so you do not need to handle incorrectly specified squares like **('k', 12)**, which do not exist. **(6 points)**

Hint 1: You can assume that the other methods are correct, and you can use them to help you solve the tasks.

Hint 2: To test that ***all_legal_moves*** works correctly, feel free to use the function ***pretty_print_list_of_squares,*** which is defined at the bottom of the code below. It takes a list of squares (i.e., tuples) as a parameter and prints them in a readable format.

```
class Rook:

    def __init__(self, square=('a', 1)):
        self.square = square

    def move(self, new_square):
        """Move the rook to new_square (without checking if this is a legal move)"""
        self.square = new_square

    def is_legal_move(self, new_square):
        """Return True if the rook can move to new_square, otherwise False"""
        pass


class Bishop:

    def __init__(self, square=('a', 1)):
        self.square = square

    def move(self, new_square):
        """Move the bishop to new_square (without checking if this is a legal move)"""
        self.square = new_square

    def is_legal_move(self, new_square):
        """Return True if the bishop can move to new_square, otherwise False"""

        # Moving to the same square is not a legal move.
        if self.square == new_square:
            return False

        new_col = new_square[0]
        new_row = new_square[1]

        current_col = self.square[0]
        current_row = self.square[1]

        # Convert the character (a-h) to a column index
        new_col_index = 'abcdefgh'.index(new_col)
        current_col_index = 'abcdefgh'.index(current_col)

        # We remain on the same diagonal if we shift the same number of squares
horizontally as vertically
        return abs(new_col_index - current_col_index) == abs(new_row - current_row)

    def all_legal_moves(self):
        """Return a list of all squares that the bishop can move to"""
        pass
```

```
def pretty_print_list_of_squares(squares):
    """Print a list of squares (represented by tuples) in a readable way"""
    for square in squares:
        print(square[0]+str(square[1]), end=' ')
    print()
```

Fill in your answer here

```
1
```

Maximum marks: 10