Division of Computing Science
Department of Information Technology
Uppsala University

Re-Exam in Programming in Python (1TD327), 5 ECTS

- *Note*: Answers should be written in English. All problems must be solved using Python code. Use short, appropriate and descriptive names for all your variables and functions. Note that your grade will be affected negatively if:
  - your code is unclear and/or hard to read
  - identical snippets of code are repeated several times (copy/paste)
  - common data structures (lists, dictionaries, strings) are not used correctly
- *Tools*: Any electronic devices or any other formula books **are not allowed**!
- *Date*: June 10, 2022, 8:00 – 13:00.
- *Place*: Fyrislundsgatan 80, sal 1
- *The grading system:*
  - The exam has two parts: (A1.-A10.) Basic and (B1.-B16.) Advanced.
  - In order to **Pass**, you need to get at least **75%** of the points from Part (A).
  - If Part (A) is failed, part (B) will not be graded.
  - To get **Grade 4**, you need to pass part (A) and get at least **50%** of the points from part (B).
  - To get **Grade 5**, you need to get at least **85%** of the points **both** from part (A) and (B).

Part A

Estimated time required: 1 hour

Each correct answer is worth 1 point.

1. Given the definitions in the code below, choose the expression that returns the following string: 'Balance on account for Smith, John at the end of January is 31.45'
   Code:

```
client_name = "John"
client_surname = "Smith"
month = "January"
balance = 31.45
```

   Select one alternative:

   A.
```
f'Balance on account for {client_surname}, {client_name} at the
    end of {month} is {balance}'
```

   B.
```
f'Balance on account for' + {client_surname} + ',' + {
   client_name} + 'at the end of' + {month} + 'is' + {balance}
```

   C.
```
'Balance on account for', client_surname,',', client_name, 'at
   the end of', month, 'is', balance
```

   D.
```
f'Balance on account for {client_surname}, {client_name[0]} at
   the end of {month[0:3]} is {balance}'
```

2. Choose the answer that correctly classifies the entities used in the following code snippet:
   Code:

```
import math

def triangle_area(a, b, c):
    s = (a + b + c)/2
    t = s*(s-a)*(s-b)*(s-c)
    r = math.sqrt(t)
    return r
```

   Select one alternative:

   A. variables: math, triangle_area, r
      functions: s, t, r
      modules: math.sqrt

   B. variables: a, b, c, s, t, r
      functions: triangle_area, math.sqrt
      modules: math

C. variables: math.sqrt, triangle_area
functions: =, +, /, *, -
modules: import, def, return

D. variables: import, def, return
functions: s, t, r
modules: triangle_area, math.sqrt

3. Choose the correct output of the following code snippet:
Code:

```
[z*z for z in range(0, 7) if z%2==0]
```

Select one alternative:

A. `['0*0', '2*2', '4*4', '6*6']`

B. `[]`

C. `[0, 4, 16, 36]`

D. `[[], [2, 2], [4, 4, 4, 4], [6, 6, 6, 6, 6, 6]]`

4. Given the following definition of the class Dice, choose the answer that only contains valid Python statements
Code:

```
import random

class Dice:
    def __init__(self, sides):
        self.sides = sides
        self.value = random.randint(1, self.sides)

    def __str__(self):
        return f'Sides: {self.sides:2d}, value: {self.value:2d}'

    def roll(self):
        self.value = random.randint(1, self.sides)
```

Select one alternative:

A.
```
d1=Dice(7)
d2=Dice('six')
d1.roll()
d2.roll()
```

B.
```
d=Dice(6)
d.roll()
print(d)
```

C.
```
d=Dice(6)
d.roll()
print(d.value())
```

D.
```
d1=Dice(2000)
d2=Dice(6)
d1.roll()
d2.roll()
is_d1_higher = d1 > d2
```

5. What is a variable in Python and what are its attributes?
Select one alternative:

   A. A variable is a Python object of class Variable.

   B. A variable in Python is any entity passed to a function and used in the function body.

   C. A variable in Python has a name and a value. The value (as well as the type) can be changed using the assignment operator =.

   D. A variable in Python is any entity returned by a function through the use of return statement.

6. What is the output of the following code? Code:

```
student = {1: {'name': 'Emma', 'age': 27, 'points': [7.0, 0.5, 3.5,
   2.0]},
           2: {'name': 'Mike', 'age': 22, 'points': [4.0, 2.5, 3.0,
   3.5]}}

student.update({3: {'name': 'Bob', 'age': 26, 'points': [5.0, 0.5, 4.5,
    3.0]}})
print(student.keys())
print(sum(student[2]['points']))
bob = student.pop('Bob')
print(bob)
```

Select one alternative:

A.
```
dict_keys([1, 2, 3])
13.0
{'name': 'Bob', 'age': 26, 'points': [5.0, 0.5, 4.5, 3.0]}
```

B.
```
dict_keys([1, 2, 3])
13.0
KeyError: 'Bob'
```

C.
```
dict_keys(['name', 'age', 'points'])
13
{'name': 'Bob', 'age': 26, 'points': [5.0, 0.5, 4.5, 3.0]}
```

D.

```
dict_keys(['name', 'age', 'points'])
13.0
{'name': 'Bob', 'age': 26, 'points': [5.0, 0.5, 4.5, 3.0]}
```

7. What is stored in the variable `counts`? Code:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 100, 110, 21, 33, 32, 2, 4]

count = 0

for n in my_list:
    if n % 2 == 0:
        count +=1
```

Select one alternative:

A. The cumulative sum of every other number in the list, that is 161.

B. The cumulative sum of all even numbers in the list, that is 260.

C. The number of even numbers in the list, that is 8.

D. Nothing because the code exits with `IndexError: list index out of range`

8. The function below should determine if the number given as input argument is an integer power of three. That is for example,

`print(is_power_of_three(3))` should print `True`,

`print(is_power_of_three(5))` should print `False`,

`print(is_power_of_three(9))` should print `True`.

Which line(s) should you uncomment in the function such that the code can finish successfully for any number n?

```
def is_power_of_three(n):
    """
    Returns True if n is a power of 3, else False.
    """
    if n == 1:
        return True
    # a)
    # else:
    #     break

    while n != 0 and n % 3 == 0:
        n = n / 3
        if n == 1:
            return True
        # b)
        # n += 1
```

```
        # c)
        # else:
        #   return False

    # d)
    # return False
```

Select one alternative:

A. The lines

```
  # else:
  #     break
```

B. The line

```
  # n +=1
```

C. The lines

```
  # else:
  #     return False
```

D. The line

```
  # return False
```

9. What is the output of the following code?
   Code:

```
a, b, c, d = 12, 3, 5, True
if a >= 0 and d:
    if b < 2:
        print("Hi")
    elif c > 3:
        print("Hello")
else:
    print("Good bye!")
```

Select one alternative:

A. `Good bye!`

B. `Hello`

C. `Hi`
   `Hello`

D. `Hi`

10. Which is the output of the code snippet below?

```
a = 10

def dummy_function(a):
    a = 20
    print(a)

a = 'ten'

dummy_function(a)
```

A. `ten`

B. `20`

C. `10`

D. `NameError: name 'a' is not defined`

Part B

Simulation of a Simple War card game

Estimated time required: 2 hours 30 minutes

For this part, you are expected to model a simplified version of the "Simple War" card game. The tasks for you to complete are to implement Python classes for representing various kinds of objects that are needed in the game, as well as a script for simulating playing turns. Make sure to remove the `pass` and the `...` instructions in the code when you implement the different tasks.

Do not forget to have a look at the appendix where you are provided with documentation for some Python methods, you might want to use them for completing the assignment.

Description of the rules for playing Simple War

The Simple War game uses a set of playing cards that consists of 52 cards, each card being characterized by a suit and a rank:

- There are 4 suits: Hearts, Clubs, Spades, and Diamonds,
- There are 14 ranks (valued from 2 to 14 points): 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.

An illustration of the game is shown on Figure 1.

At the beginning of the game, the main deck of 52 shuffled cards is dealt to the two players, each of them gets the same number of cards, that is 26 cards. The players keep their cards hidden. In order to win the game, each player wants to have the most cards in its deck, ideally all cards.

A turn is played as follows:

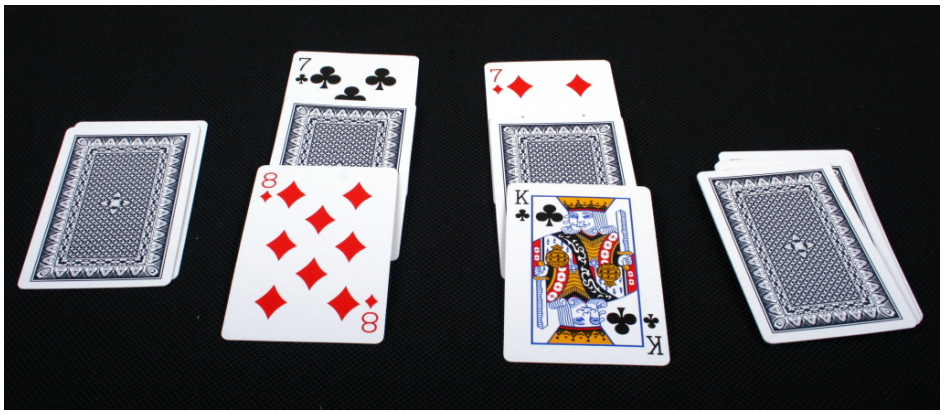

Figure 1: Example of card decks of two players in a Simple War game

1. The players take the top cards of their deck, flip them and put the cards face-up in front of each other.
2. The player with the highest card wins and takes both cards then adds them to the bottom of its deck. Aces are the highest card and 2s the lowest. Suits are ignored. For instance, "Queen of Heart" is higher than "Seven of Spades".
3. If the two cards flipped are equal, for example "Jack of Diamonds" and "Jack of Clubs", they are left in the center and no player takes any card.

Turns are played until the players agree on finishing the game, or if one of them has no more cards.

Description of the Python classes and objects used for modelling the game

Three classes are used in the game modelling:

- The `Card` class (code listing 2) that is used for modelling each of the 52 cards used in the game. The constructor of the class takes 2 input parameters that are:
  - `suit`: (string), suit of the card,
  - `rank`: (string), rank of the card.
- The `Deck` class (code listing 4) that models a deck of cards. The constructor does not take any input parameter. An instance of the `Deck` class can be used for modelling both the deck shared by the two players as well as the cards in the hand of a player.
- The `SimpleWar` class (code listing 6) that implements a few methods for playing the game. The constructor does not take any input parameters.

The two players are modeled with one dictionary each. The dictionary representing a player contains two items:

- `name`: (string), name of the player,
- `deck`: (`Deck` object), cards in the hand of the player.

Tasks to solve

Provided with a short script for initializing the game in the code listing 1, the tasks for you to solve are described below (B.1. to B.14.). Note that the task B.14. is a bonus task, that means the points you can get with it are not taken into account for defining the grading thresholds.

B.1. [3 points]

Implement the constructor of the `Card` class:

- Define an attribute `suit` from the input parameter *suit*.
- Define an attribute `rank` from the input parameter *rank*.
- Define an attribute `pts`: value of the card, that is the number of points of the card.

B.2. [2 points]

Implement `wins(self, adversary_card)` method of the `Card` class. The variable `adversary_card` is another instance of the `Card` class. The method should return `-1` if the adversary card is worth more points than the local class instance, `1` if the adversary card is worth fewer points than the local class instance, or `0` if both cards are equally valued.

B.3. [2 points]

Implement the `__str__(self)` method of the `Card` class, based on the output example in the code listing 3.

B.4. [1 point]

Implement the constructor of the `Deck` class: define an attribute `set_of_cards` that models the cards in the deck and that is initialized to an empty list of cards.

B.5. [1 point]

Implement the method `nb_of_cards(self)` of the `Deck` class, based on the output example in the code listing 5. This method returns the number of cards in the deck.

B.6. [1 point]

Implement the `add_cards(self, new_cards)` method of the `Deck` class, based on the output example in the code listing 5. This method adds all cards from the list `new_cards` to the bottom of the deck. Note that the first card in the attribute `set_of_cards` is the card at the bottom of the deck, the last card in the attribute `set_of_cards` is the card on the top.

B.7. [1 point]

Implement the method `shuffle_cards(self)` of the `Deck` class, based on the output example in the code listing 5. This method shuffles the cards in the deck, that means it randomly changes the order of the cards.

B.8. [2 points]

Implement the method `pick_top_cards(self)` of the `Deck` class, based on the output example in the code listing 5. This method returns the top card from the deck if it is not empty. If the deck is empty, the methods returns None.

B.9. [1 point]

Implement the method `empty_deck(self)` of the `Deck` class, which removes all cards from the deck.

B.10. [4 points]

Implement the constructor of the `SimpleWar` class:

- Create two players `player1` and `player2` named `name1` and `name2`. Each player is initialized with an empty set of cards in its hand.
- Create a main deck of 52 cards, that is all cards in a classical set of playing cards. Shuffle the cards in this deck.

Look carefully at the code output in the code listing. 7.

B.11. [3 points]

Implement the `deal_cards(self)` method of the `SimpleWar` class, based on the output example in the code listing 7. This method deals half of the cards of the main deck to each player, that implies that the main deck is empty after this operation.

B.12. [5 points]

Implement the `play_turn(self)` method of the `SimpleWar` class, based on the output example in the code listing 7. This method plays a turn of the simple war game as follows:

1. Each player picks the top card of its deck and adds them to the main deck.
2. The player with the highest card (most points on the card) wins and put the two cards in its deck. The main deck is left empty. If ever both cards are equal, there is war. The cards are left in the main deck.

This method should return the number of cards in the deck of each player after the turn is played.

B.13. [2 points]

Implement the `winner(self)` method of the `SimpleWar` class. This method returns the player which has the most cards.

B.14. Bonus task! [4 points]

Simulate a game! You can assume that all classes and functions are written and saved together in the same file. Complete the code listing 7:

1. Instantiate a game with the variable `game` and assume the two players are named "Piff" and "Puff".
2. Deal the cards to the players.
3. Play turns until either one of the players has no more cards, or 100 turns have been played.

Listing 1: Initialization

```
import random

suits = ['Hearts','Clubs','Spades','Diamonds']

points = {'Two': 2, 'Three': 3, 'Four': 4, 'Five': 5, 'Six' : 6,
          'Seven' :7, 'Eight': 8,'Nine':9, 'Ten':10,
          'Jack':11, 'Queen': 12,'King': 13,'Ace':14}
```

Listing 2: Card class

```
class Card():
    """
    Models a playing card.
    """
    def __init__(self, suit, rank):
        # Task B.1.:
        pass

    def wins(self, adversary_card):
        # Task B.2:
        pass

    def __str__(self):
        # Task B.3.:
        pass
```

Listing 3: Card instance example

```
mycard = Card('Clubs', 'Ace')
print(f'My card is {mycard} and is worth {mycard.pts} points.')

# Output:
My card is Ace of Clubs and is worth 14 points.
```

Listing 4: Deck class

```
class Deck():
  """
  Models a deck of playing cards.
  """
  def __init__(self):
    # Task B.4:
    pass

  def nb_of_cards(self):
    # Task B.5:
    pass

  def add_cards(self, new_cards):
    # Task B.6:
    pass

  def shuffle_cards(self):
    # Task B.7:
    pass

  def pick_top_card(self):
    # Task B.8:
    pass

  def empty_deck(self):
    # Task B.9:
    pass

  def __str__(self):
    return '(bottom) ' + ', '.join([card.__str__() for card in self.
   set_of_cards]) + ' (top)'
```

Listing 5: Deck instance example

```
a_deck = Deck()
a_deck.add_cards([mycard, Card('Spades', 'King'), Card('Clubs', 'Queen'),
   Card('Hearts', 'Three')])
```

```
print(f'There are {a_deck.nb_of_cards()} cards in the deck.')
print('\nThe cards are: ', a_deck)

print('\nThe top card is: ', a_deck.set_of_cards[-1])

print('\nShuffling the cards!')
a_deck.shuffle_cards()
print('\nPicking the top card: ', a_deck.pick_top_card())

print(f'\nNow, there are {a_deck.nb_of_cards()} cards left in the deck.')

# Output:
There are 4 cards in the deck.

The cards are:  (bottom) Three of Hearts, Queen of Clubs, King of Spades,
    Six of Clubs (top)

The top card is:  Six of Clubs

Shuffling the cards!

Picking the top card:  Three of Hearts

Now, there are 3 cards left in the deck.
```

Listing 6: SimpleWar class

```
class SimpleWar():
  """
  Methods for simulating a simple war card game.
  """

  def __init__(self, name1, name2):
    # Task B.10:
    pass

  def deal_cards(self):
    # Task B.11:
    pass

  def play_turn(self):
    # Task B.12:
    ...
    print()
    print('Card 1: ', card1)
    print('Card 2: ', card2)
    print(self.main_deck)
```

```
    if card1 is None or card2 is None:
      print('\nGame over!')
    ...
            print(f"{self.player1['name']} wins the turn")
        ...
            print(f"{self.player2['name']} wins the turn")
        ...
            print('War!')

    ...

  def winner(self):
    # Task B.13:
    pass
```

Listing 7: SimpleWar game example

```
game = ...

print(game.player1)
print(game.player2)

...
print(game.player1['deck'])
print(game.player2['deck'])

n1, n2 = game.player1['deck'].nb_of_cards(), game.player2['deck'].
    nb_of_cards()

max_n_turns = 100
...

...

print(f"\nAnd the winner is... {game.winner()['name']}!")

# Output
{'name': 'Piff', 'deck': <__main__.Deck object at 0x7f6c11642e50>}
{'name': 'Puff', 'deck': <__main__.Deck object at 0x7f6c11642850>}

(bottom) Two of Clubs, Two of Diamonds, Four of Diamonds, Four of Clubs,
    Two of Spades, Two of Hearts (top)
(bottom) Three of Diamonds, Four of Hearts, Three of Clubs, Four of Spades
    , Three of Spades, Three of Hearts (top)

...
Card 1:  Three of Hearts
Card 2:  Three of Diamonds
(bottom) Three of Diamonds, Three of Hearts (top)
```

```
War!

Card 1:  Three of Clubs
Card 2:  Two of Clubs
(bottom) Two of Clubs, Three of Clubs, Three of Diamonds, Three of Hearts
   (top)
Piff wins the turn

Card 1:  Three of Spades
Card 2:  Four of Hearts
(bottom) Four of Hearts, Three of Spades (top)
Puff wins the turn

Card 1:  Four of Spades
Card 2:  Two of Hearts
(bottom) Two of Hearts, Four of Spades (top)
Piff wins the turn

Card 1:  Two of Spades
Card 2:  Four of Hearts
(bottom) Four of Hearts, Two of Spades (top)
Puff wins the turn
...

And the winner is ... Piff!
```

Plagiarism detection at an art school

Estimated time required: 45 minutes

At the School of Modern Art originality is the most valued attribute. The students regularly take originality exams. The exams are conducted by seating the students in a classroom arranged in n columns and m rows. The students are then told to write an original integer on a piece of paper. When collecting the results the teacher noticed that sometimes the students that sat next to each other return integers that are close in value and thus unoriginal.

Task to solve

B.15. [5 points]

Write a function `detect_cheaters` that will receive exam answers as a list of `m` rows of `n` columns and determine the number of students suspected of cheating. The teacher considers a pair of students that sit close to each other to be cheating if their answers differ by less than `t` in absolute value. You should assume that a student can only plagiarise the results from students sitting in front and behind him in the same column or to the left and to the right of him in the same row.

Example 1

Input:

```
exam_results=[[1, 10], [2, 15]]
detect_cheaters(exam_results, 3)
```

Output:

```
2
```

Example 2

Input:

```
exam_results=[[1, 4, 7], [3, 6, 7], [23, 12, 5]]
detect_cheaters(exam_results, 3)
```

Output:

```
7
```

Example 3

Input:

```
exam_results=[[1, 4, 5, 26, 34], [53, 4, 2, 3, 5], [63, 2, 35, 4, 33]]
detect_cheaters(exam_results, 3)
```

Output:

```
8
```

Text wrapping

Estimated time required: 45 minutes

Text wrapping is breaking a section of text into lines so that it will fit into the available width of a page, window or other display area. In text display this is achieved by continuing on a new line when a line is full, so that each line fits into the viewable window, allowing text to be read from top to bottom without any horizontal scrolling.

Task to solve

B.16. [5 points]

We are given a string that we want to display in a window that can display w characters per line. When displaying the text in the window we can break the line only on spaces that are in front of a new word. The space character is displayed as a part of the previous line and can extend past the edge of the window. A new line always begins with a character that is not a space.

Write a function `wrap_text(display_text, w_max)` that will calculate for each window width from 1 to `w_max` the number of lines required to display the text. The text is passed to the function as a string `display_text`. You can assume that it only contains alphanumeric characters, separators and space characters. The text will not contain any newline or tabulator characters and will never start with a space. The function should return a list of length `w_max` that gives for each width between 1 and `w_max` the number of lines required to display the string. If it is not possible to display the text for certain display window width place the value of -1 in corresponding position in the list.

Example 1

Input:

```
text="Every novel is a mystery novel if you never finish it."
n_lines=wrap_text(text, 20)
print(n_lines)
```

Output:

```
[-1, -1, -1, -1, -1, -1, 9, 9, 7, 6, 6, 5, 5, 5, 4, 4, 4, 4, 4, 3]
```

Example 2

Input:

```
text="Many were increasingly of the opinion that they had all made a big
    mistake in coming down from the trees in the first place. And some
    suggested that even the trees had been a bad move, and that no one
    should ever have left the oceans."
n_lines=wrap_text(text, 20)
print(n_lines)
```

Output:

```
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 22, 21, 18, 17, 15, 15, 15,
   14, 13]
```

Appendix

Excerpt from documentation for module `random`:

```
choice(self, seq) method of Random instance
    Choose a random element from a non-empty sequence.

gauss(self, mu, sigma) method of Random instance
    Gaussian distribution.

    mu is the mean, and sigma is the standard deviation.  This is
    slightly faster than the normalvariate() function.

    Not thread-safe without a lock around calls.

randint(self, a, b) method of Random instance
    Return random integer in range [a, b], including both end points.

random(...)
    random() -> x in the interval [0, 1).

randrange(self, start, stop=None, step=1, _int=<type 'int'>, _maxwidth
   =9007199254740992L) method of Random instance
    Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

sample(self, population, k) method of Random instance
    Chooses k unique random elements from a population sequence.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    To choose a sample in a range of integers, use xrange as an argument.
    This is especially fast and space efficient for sampling from a
    large population:   sample(xrange(10000000), 60).

shuffle(x)
```

```
    Shuffle the sequence x in place.
```

Excerpt from the Python documentation for some the methods of list objects:

```
append(x)
    Add an item to the end of the list.

extend(iterable)
    Extend the list by appending all the items from the iterable.

insert(i, x)
    Insert an item at a given position. The first argument is the index of
    the element before which to insert.

remove(x)
    Remove the first item from the list whose value is equal to x. It
   raises a ValueError if there is no such item.

pop([i])
    Remove the item at the given position in the list, and return it. If
   no index is specified, a.pop() removes and returns the last item in the
    list. (The square brackets around the i in the method signature denote
    that the parameter is optional, not that you should type square
   brackets at that position. You will see this notation frequently in the
    Python Library Reference.)

clear()
    Remove all items from the list.

index(x[, start[, end]])
    Return zero-based index in the list of the first item whose value is
   equal to x. Raises a ValueError if there is no such item.

    The optional arguments start and end are interpreted as in the slice
   notation and are used to limit the search to a particular subsequence
   of the list. The returned index is computed relative to the beginning
   of the full sequence rather than the start argument.

sort(*, key=None, reverse=False)
    Sort the items of the list in place (the arguments can be used for
   sort customization).

reverse()
    Reverse the elements of the list in place.
```