Assignment Guidelines.

- This assignment covers material in Module 09.
- Submission details:
  - Solutions to these questions must be placed in files `a08q1.py`, and `a08q2.py`, respectively, and must be completed using Python 3.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - Helper functions need design recipe elements but not examples and tests.
- Download the testing module from the course web page. Include `import check` in each solution file.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - While you may use global `constants` in your solutions, **do not** use global `variables` for anything other than testing.
  - Read each question carefully for additional restrictions.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

This assignment has only two questions as each question requires defining several functions, lot of testing and cases to consider. Pay attention to details and remember that passing the basic tests doesn’t mean that your solution is correct. Test thoroughly and think carefully about black box and white box testing.
Question 1:

FunBoardGame: In this question you will create a function to play a round of FunBoardGame along with several helper functions. This game consists of a square board NXN where each cell could be empty, has a black disk or a white one (A complete definition to follow).

The game requires two players where the first one has black disks while the second one has white disks. The first player starts by placing a black disk on any empty cell on the board, then the second player places a white disk on an empty cell, and so on. The game ends once one of the players wins the game or there are no empty cells on the board (No winner).

A player wins the game if there is a complete row, col, or diagonal that is full of the player’s disks

Here is a complete definition the type of the board which is called BoardGame:

```plaintext
# BoardGame is a (dictof Str (listof (anyof 'B' 'W' 'E'))) and represents a board of size NxN (N>=2), B for Black, W for White # requires:
#   - it has exactly 2N+2 keys named 'r1','r2',...,'rN','c1','c2',...,'cN','d1','d2'
#     d1 is the main diagonal, d2 is the secondary diagonal
#   - the value of each key is a list of length N
#   - the key 'r1' represents the first row on the board, and so on...
#   - the key 'c1' represents the first col on the board, and so on...
#   - the key 'd1' represents the main diagonal
#   - the key 'd2' represents the secondary diagonal.
```
Check important notes at the end of page 7.

a) Write a Python function `is_winner` that consumes a `BoardGame (brd)` and a string (color) (any of 'W' 'B'), and **returns** True if `brd` is a winner board for color, False otherwise.

Example:

```python
brda = { 'r1' : ['B','W','W'],
    'r2' : ['E','B','W'],
    'r3' : ['E','E','B'],
    'c1' : ['B','E','E'],
    'c2' : ['W','B','E'],
    'c3' : ['W','W','B'],
    'd1' : ['B','B','B'],
    'd2' : ['W','B','E']}

is_winner(brda, "B") => True
is_winner(brda,"W") => False
```

b) Write a Python function `add_disk` that consumes a `BoardGame (brd)`, two natural numbers between [1,sizeof brd] (row,col), and a string (color) (any of 'W' 'B'), and **mutates** the board by “placing” a disk with color (color) on the board on row (row) and column (col).

Example:

```python
brdb = { 'r1' : ['E','W','W'],
    'r2' : ['E','B','W'],
    'r3' : ['E','E','B'],
    'c1' : ['E','E','E'],
    'c2' : ['W','B','E'],
    'c3' : ['W','W','B'],
    'd1' : ['E','B','B'],
    'd2' : ['W','B','E']}

add_disk(brdb,1,1,"B") =>  None
and brdb becomes equal to brda defined in part (a)
c) Write a Python function `game_over` that consumes a `BoardGame (brd)`, returns `True` if `brd` full of disks, False otherwise.

Example:
```
brdc1= { 'r1' : ['E','W'],
       'r2' : ['E','B'],
       'c1' : ['E','E'],
       'c2' : ['W','B'],
       'd1' : ['E','B'],
       'd2' : ['W','E']}
game_over(brdc1) => False
```

d) Write a Python function `play` that consumes a `BoardGame (brd)` and plays the game as described in the examples.

Example1:
```
brd1 = { 'r1' : ['E','E','E'],
       'r2' : ['E','E','E'],
       'r3' : ['E','E','E'],
       'c3' : ['E','E','E'],
       'c1' : ['E','E','E'],
       'c2' : ['E','E','E'],
       'c3' : ['E','E','E'],
       'd1' : ['E','E','E'],
       'd2' : ['E','E','E']}
```
calling `play(brd1)` prints the following (The player’s input is in *Italic* and **bold**)

Check important notes at the end of page 7.
Welcome to Board-Game version 1
You need two players; player1 will play with black disks, while player2 will play with white disks.

Player1: enter row number (between 1-3): 1
Player1: enter column number (between 1-3): 1
Player2: enter row number (between 1-3): 1
Player2: enter column number (between 1-3): 2
Player1: enter row number (between 1-3): 2
Player1: enter column number (between 1-3): 2
Player2: enter row number (between 1-3): 1
Player2: enter column number (between 1-3): 3
Player1: enter row number (between 1-3): 3
Player1: enter column number (between 1-3): 3

We have a winner! Congratulations Player1

and if you call print(brd1) in the interaction window, it prints:

```python
dict = {'r1': ['B', 'W', 'W'],
       'r2': ['E', 'B', 'E'],
       'r3': ['E', 'E', 'B'],
       'c1': ['B', 'E', 'E'],
       'c2': ['W', 'B', 'E'],
       'c3': ['W', 'E', 'B'],
       'd1': ['B', 'B', 'B'],
       'd2': ['W', 'B', 'E']}
```
Example 2:

```python
brd1 = { 'r1' : ['E','E','E'],
        'r2' : ['E','E','E'],
        'r3' : ['E','E','E'],
        'c1' : ['E','E','E'],
        'c2' : ['E','E','E'],
        'c3' : ['E','E','E'],
        'd1' : ['E','E','E'],
        'd2' : ['E','E','E']}
```

calling `play(brd1)` prints the following (The player’s input is in Italic and bold)

Welcome to Board-Game version 1
You need two players; player1 will play with black disks, while player2 will play with white disks.
Player1: enter row number (between 1-3): 1
Player1: enter column number (between 1-3): 1
Player2: enter row number (between 1-3): 1
Player2: enter column number (between 1-3): 2
Player1: enter row number (between 1-3): 2
Player1: enter column number (between 1-3): 2
Player2: enter row number (between 1-3): 1
Player2: enter column number (between 1-3): 3
Player1: enter row number (between 1-3): 3
Player1: enter column number (between 1-3): 2
Player2: enter row number (between 1-3): 2
Player2: enter column number (between 1-3): 1
Player1: enter row number (between 1-3): 3
Player1: enter column number (between 1-3): 1
Game Over: No winner

and if you call print (brd1) in the interaction window, it prints:

```python
{'r1': ['B', 'W', 'W'],
 'r2': ['W', 'B', 'B'],
 'r3': ['B', 'B', 'W'],
 'c1': ['B', 'W', 'B'],
 'c2': ['W', 'B', 'B'],
 'c3': ['W', 'B', 'W'],
 'd1': ['B', 'B', 'W'],
 'd2': ['W', 'B', 'B']}
```

Notes:

- The program assumes that players are entering valid input and are placing their disks only on empty cells.
- Use the constants provided in the interface file for printing to avoid any typos.
Question 2:

In this question you will implement a class that represents a date. The interface has already been created for you including partial documentations. Make sure to read them carefully.

Do not change the provided info other than adding the implementation, completing the design recipe, and adding extra methods/constants as needed. (You also need to implement __repr__)

Make sure you read and understand the following examples in order to implement the methods correctly and create correct tests.

We also provided a pdf file for you to learn how to find the day of the week for a given date. In other words, once you learn and understand the algorithm, you will be able to answer the question: what is the day of the week for the date 1/8/2018 (dd/mm/yyyy)? and your answer should be: Wednesday.

```python
>>> d=Date(28,2,2018)
>>> d
Wednesday, February 28, 2018
>>> d.nextDay()

>>> d
Thursday, March 1, 2018
>>> d.nextYear()

>>> d
Friday, March 1, 2019

>>> d=Date(28,2,2008)
>>> d
Thursday, February 28, 2008
>>> d.nextDay()

>>> d
Friday, February 29, 2008
>>> d.nextYear()

>>> d
Saturday, February 28, 2009
```
Note for both questions (Q1 and Q2):

- Those examples don’t cover all the unique cases that you should be aware of.
- Spend some time thinking about all the unique cases and test them thoroughly.

Tip: Definition of a Leap year

if (year is not divisible by 4) then (it is a common year)
else if (year is not divisible by 100) then (it is a leap year)
else if (year is not divisible by 400) then (it is a common year)
else (it is a leap year)