Method Overloading Assignment:

1. Complete the class **Date**, which stores and manipulates dates. As specified below, write the required methods, including those needed for overloading operators. Exception messages should include the class and method names, and identify the error (including the value of all relevant arguments). See the **your_solution.py** file for one example, this is where you will write your solution.

You *may not* import/use the datetime or other similar modules in Python.
Reference: What is a Leap Year ↠ https://www.timeanddate.com/date/leapyear.html
Reference: vector.py posted on Discord on the Announcements channel

1. The class is initialized (**__init__**) with three int values (the year first, the month second, the day third.) If any parameter is not an int, or not in the correct range (the year must be 0 or greater, the month must be between 1 and 12 inclusive, and the day must be legal given the month and year: remember about leap years) raise an **AssertionError** with an appropriate string describing the problem/values. When initialized, the Date class should create exactly **three** self variables named **year**, **month**, and **day** (with these exact names and no other self variables).
2. Write the **__getitem__** method to allow Date class objects to be indexed by either:
(a) an str with value 'y' or 'm' or 'd'
(b) any length tuple containing any combinations of just these three values: e.g., ('m','d').
If the index is not one of these types or values, raise an **IndexError** exception with an appropriate string describing the problem/values. If the argument is 'y', returns the year; if the argument is 'm', return the month, and if the argument is 'd', return the day. If the argument is a tuple, return a tuple with year or month or day substituted for each value in the tuple. So if d = day(2016,4,15) then d['y'] returns 2016 and d['m','d'] returns (4,15).
Note that calling d['m'] will pass 'm' as its argument; calling d['m','d'] will pass the tuple ('m','d') as its argument.

3. Write methods that return:
(a) the standard repr function of a Date, remember the definition of repr is to return a string that when you pass it to eval it creates an instance object that has the same values as the current object.
(b) a str function of a Date: str for a Date shows the date in the standard format: str(Date(2016,4,15)) returns '4/15/2016'; it is critical to write the str method correctly, because it is used in the batch self-check file for testing the correctness of other methods.

4. Write a method that interprets the length of a Date as the number of days that have elapsed from January 1st in the year 0 to that date. So len(Date(0,1,1)) returns 0; len(Date(0,12,31)) returns 365; len(Date(2016,4,15)) returns 736,434. **Hint**: len(Date(0,3,14))

returns 73: 31 days in January, 29 days in February (it is a leap month), and 13 days in March, which sum to 73.
5. Overload the == operator to allow comparing two Date objects for equality (if a Date object is compared against an object from any other class, it should return False). Note that if you define == correctly, Python will be able to compute != by using ==.
6. Overload the < operator to allow comparing two Date objects. The left Date is less-than the right one if it comes earlier than the right one. Also allow the right operand to be an int: in this case, return whether the length (an int, see above) of the Date is less-than the right int. If the right operand is any other type, Python should raise a TypeError exception (hint: NotImplemented, so when you return NotImplement, it is the same as saying raise TypeError) with the standard error message. Note that if you define < correctly, Python will be able to compute Date > Date and int > Date by using <.
7. Overload the + operator to allow adding a Date object and an int (which can be positive or negative) producing a **new** Date object as a result (and not mutating the Date object + was called on). The result is a new Date that many days in the future for a positive int; in the past for a negative int. For example, Date(2016,4,15)+100 returns Date(2016,7,24), 100 days in the future of 4/15/2016. If the other operand is not an int, Python should raise a TypeError exception (hint: NotImplemented) with the standard error message. Both Date + int and int + Date should be allowed and have the same meaning. Hint: write code that repeatedly adds/subtracts one day at a time to get to the required Date.
8. Overload the - operator to allow subtracting two Date objects, producing an int object as a result (and not mutating the Date objects - was called on). The difference is the number of days from the left Date to the right Date (which can be negative). For example, Date(2016,6,8)-Date(2016,4,15) returns 54 and Date(2016,4,15)-Date(2016,6,8) returns -54. Hint: use the len function defined above. Also, allow subtracting an int from a Date. It should produce the same value as adding its negative value to a Date: for example, Date(2016,4,15)-1 should produce the same result as Date(2016,4,15)+-1.

———————————————————————————————————————

*** *IGNORE FOR NOW* 9. Write the __call__ method to allow an object from this class to be callable with three int arguments: update the year of the object to be the first argument, and the month of the object to be the second argument, and the day of the object to be the third argument. Return None. If any parameter is not legal (see how the class is initialized), raise an AssertionError with an appropriate string describing the problem/values. ***

———————————————————————————————————————

The overloading project folder contains a **tests.txt** file (examine it) to use for batch-self-checking your class, via the driver.py script. These are rigorous but not exhaustive tests. Incrementally write and test your class.