uc3m | Universidad Carlos III de Madrid

Data Structures and Algorithms.
Author: Isabel Segura Bedmar

Unit 1 – Abstract Data Types

Problem - Our first Python class, CreditCard.

Design and implement a class to represent the credit cards for bank customers. Every credit card must have the following data:

- the name of the customer.
- an id for the credit card.
- the balance of the credit card, which is the money spent.
- the limit of the credit card, which is the total amount that the customer can spend.

Moreover, the following operations must be considered:

- charge, which takes an amount of money and increases the balance of the credit card with this amount. If the new balance (current balance plus charge) exceeds the limit of the credit card, the operation must no be performed.
- make_deposit, which takes an amount of money and decreases the balance. The deposit should not exceed the current balance.

Note: Please, stick to the specifications mentioned above, do not make different assumptions about how credit cards should work. This is only an exercise to practice programming classes with Python.

Problem - Multidimensional Vector Class

Please, implement a class, **Vector**, to represent the coordinates of a vector in a multidimensional space. For example:

In a three-dimensional space, we might wish to represent a vector with the following coordinates: 5, −2, 3 .

In a five-dimensional space, a possible vector may have the following coordinates: 0,1,-1,3,2.

The class must contain the following methods:

- **__init__**(**self**,dim): constructor methods that creates a vector of dimension dim. In this method,, all coordinates of the vector are equal to 0.
- **__len__**(**self**): returns the dimension of the vector.
- **__str__**(**self**): returns a string that represents the vector. For example, if the coordinates of the vector are: 3,5,0, the method should return the string "(3,5,0)".
- getItem(**self**,i): returns the ith coordinate of the vector. The first coordinate is always represented by the index 0.
- setItem(**self**,i,newValue): modifies the ith coordinate of the vector to the given newValue.
- __eq__(**self**,other): returns True if the invoking vector and the other vector are equal, and false otherwise
- sumVector(**self**,other): returns a new vector, which is the sum of the invoking vector and the param other.

Problem - Date ADT

Implement a class, Date, to represent dates. A date represents a single day in our calendar (e.g December 25, 2018 AC).The operations are:

- **__init__**(*day,month,year*): creates a new date instance.
- **_str__**(): returns a string representation in the format '*dd/mm'/yyyy*'.
- **day**(): returns the day number of this date.
- **month**(): returns the month number of this date.
- **year**(): returns the year number of this date.
- **monthName**(): returns the month name of this date.
- **isLeapYear**(): return True if this date falls in a leap year, and False otherwise.
- **compareTo**(other): compares this date to the *other* to determine their logical order.
    - If this date is before *other*, returns -1.
    - If both dates are the same, return 0. I
    - If this date is after *other*, returns 1.

Problem – The Polynomial ADT

A polynomial is an expression representing a mathematical sum of several terms. Each term has a number called the coefficient, a variable and a power of the variable called the exponent.

$Q(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$

For example, a_1 is the coefficient for term of degree 1, a_2 is the coefficient for term of degree 3, and so on. a_0 is the term of degree 0, is also named as constant term of the polynomial.

The operations for the polynomial ADT are:

- **__init__**(coef): creates a new polynomial whose coefficients are the elements of the input list coef. The element at index 0 is the coefficient of term with degree 0 (constant term), the element at index 1 is the coefficient of term with degree 1, and so on.
- **getDegree**(), which returns the polynomial grade. For example, Q(x)=5 has degree 0. $Q(x)=x^2+5$ has degree 2.
- **getCoefficient**(n), which returns the coefficient of the term, which is squared to n. For example, given the polynomial $Q(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$, this call getCoefficient(3) returns a_3.
- **setCoefficient**(n, newValue), which modifies the coefficient of the term whose power is n by the value newValue. For example, given the polynomial $Q(x)=a_0+a_1x+a_2x^2+a_3x^3+\ldots+a_nx^n$, this call setCoefficient(3,b) does that now Q(x) is $a_0+a_1x+a_2x^2+bx^3+\ldots+a_nx^n$
- **evaluate**(x), which takes x as param and returns the value of the polynomial functions for this value. For example, $Q(3)=a_0+a_13+a_29+a_327+\ldots+a_n3^n$
- **sum**(p), which returns the sum of the invoking polynomial and the polynomial p. The invoking polynomial must not be modified. For example, $Q(x)=3x^2+4x+5$, $p(x)=x^3+4x^2-2x-3$. Q.sum(p) -> x^3+7x^2+2x+2 .

Note: Use a Python list to store the polynomial coefficients. What is the best way to store the coefficients into the list?. In what position of the list is it better to store the constant term?. Why?