PROGRAMACIÓN ORIENTADA A OBJETOS – TRABAJO DE INVESTIGACIÓN

Prof.:

- **Manuel Ortega**
- **María Isabel Masanete**
- **Marcelo Mondre**

Alumnas:

- **Carrizo Araya, Andrea Sofía (Reg. 16658)**
- **Flores Nuñez, Maira (Reg.19165, Reg. 20617)**

2022

Introducción

El ser humano todos los días enfrenta distintas situaciones en las actividades que hace, estas las tiene que resolverlos basándose en sus conocimientos y experiencias adquiridas, pero, los realiza usando alguna metodología o una serie de pasos con la finalidad de obtener una solución que le convenga. Los hombres en su quehacer diario tienen acciones rutinarias como prepararse para ir a estudiar o a trabajar, tomar el taxi o el bus, atender las tareas diarias del trabajo, preparar los alimentos del día, llevar a los niños a la escuela, responder los mensajes de los correos electrónicos; todas ellas siguen una secuencia y un propósito. Al conjunto de actividades ordenadas con un objetivo claro se le llama algoritmos.
En este trabajo se aborda el tema de Lista y algunos de sus métodos más utilizados.

Palabras Claves

- Lista
- Append
- Sort
- Remove

Desarrollo

Lista

Una lista pueden almacenar cualquier tipo de valor: números, cadenas de texto, valores binarios u objetos más complejos, como funciones o incluso otras listas.
Una lista es una sucesión de elementos en un cierto orden, que se define entre corchetes, esto quiere decir que cada elemento ocupa una posición en la lista, y esto es algo que no debe pasar desapercibido. De hecho, normalmente interiorizamos este concepto de «orden» al trabajar con listas.

Cuando en Python se crea una lista, lo que sucede «internamente» es lo siguiente:

1. El intérprete de Python reserva memoria para almacenar cada elemento de la lista. Estos elementos pueden estar alojados en cualquier lugar de la memoria.
2. El intérprete de Python busca un «hueco» en memoria con n posiciones consecutivas, donde n es el número de elementos de la lista, y lo reserva.
3. El intérprete de Python aloja en cada posición de memoria un puntero a la posición de memoria que contiene el elemento almacenado.

FUNCIONES DE LISTAS

- **Append:** Añade un único elemento al final de la lista

```
x = [1, 2]
x.append('h')
print(x)
#Salida: [1, 2, 'h']
```

- **Sort:** Ordena los elementos de manera ascendente

  ```
  numero = [5,1,3,2,4]
  numero.sort()
  print(numero)
  #Salida: [1,2,3,4,5]
  ```

- **Remove:** Remueve la primer coincidencia del elemento especificado.

  ```
  x = [1,2,'h',3,'h']
  x.remove('h')
  print(x)
  #Salida: [1,2,3,'h']
  ```

Ejemplo

```
#MODULO ALUMNO
class Alumno:        #clase que instancia los objetos de tipo Alumno
   __nombre = str            #atributos de la clase
   __registro = int
   __notaFinal = float

   def __init__ (self, nom, reg, nota):        #costructor de la clase
      self.__nombre = nom
      self.__registro = reg
      self.__notaFinal = nota

   def __str__ (self):            #metodo que permite imprimir los objetos con formato
      return ('Nombre: {} Registro: {} Nota Final: {}'.format( self.__nombre, self.__registro, self.__notaFinal ))

   def getNombre (self):        #se retorna el atributo nombre del objeto solicitado
      return self.__nombre

   def getRegistro (self):      #se retorna el atributo registro del objeto solicitado
      return self.__registro

   def getNota (self):          #se retorna el atributo nota del objeto solicitado
      return self.__notaFinal

   def ordenar (lista):            #metodo que ordena la lista de mayor a menor
      list.sort( lista, key = lambda Alumno: Alumno.__notaFinal, reverse = True )
      #se coloca key para indicarle el atributo que debe ordenar
      #se coloca reverse = True para que ordene de mayor a menor
      #si este atributo reverse no se colocase, el metodo sort() por defecto ordena de menor a mayor



#MODULO MANEJADOR
from claseAlumno import Alumno

class Manejador:          #clase que maneja la lista de objetos
   __lista = [] #atributo lista

   def __init__ (self):
      self.__lista = []            #inicializacion de la lista

   def __str__ (self):            #metodo que imprime la lista de objetos
```

```
        s = ""
        for alumno in self.__lista:
            s += str( alumno) + '\n'
        return s

    def agregar (self, unAlumno):               #agrega un objeto Alumno al final de la lista
        self.__lista.append( unAlumno )

    def ordenar (self):                 #llama al metodo de la clase Alumno y le envia la lista para ordenar
        Alumno.ordenar( self.__lista )

    def eliminar (self):                #metodo que elimina las notas menores a 6.00
        for i in range( len( self.__lista ) ):
            if self.__lista[i].getNota() < 6.00:
                self.__lista.remove( self.__lista[i] )


#MODULO PROGRAMA PRINCIPAL
from claseAlumno import Alumno
from claseManAlumno import Manejador

if __name__ == '__main__':                  #linea donde comienza a ejecutar el programa
    lista = Manejador()                 #se crea una instancia de la clase Manejador
    a1 = Alumno( 'Luis Perez', '19587', 7.45 )              #se crea instancias de la clase Alumno
    a2 = Alumno( 'Alamino Jesica', '17578', 1.25 )
    a3 = Alumno( 'Javier Torres', '20489', 8.41 )

    lista.agregar( a1 )         #se agrega los objetos a la lista
    lista.agregar( a2 )
    lista.agregar( a3 )

    lista.ordenar()             #ordena la lista por notas de mayor a menor nota
    print(lista)                #se imprime la lista ordenada por pantalla

    lista.eliminar()            #elimina las notas menores a 6.00
    print(lista)                #se imprime la lista con las notas mayores a 6.00
```

Conclusión

En los métodos abordados se observo que al utilizar el método append() este almacenaba los elementos al final de la lista. Por otro lado el método sort() ocasiona que el interprete informe un error si a este se le brindan elementos que nos son de un tipo homogéneo, por lo tanto, la forma correcta de ordenar una lista que contiene objetos es utilizar el método sort() de la clase list con los parámetros que se mencionaron en el ejemplo, este método siempre devuelve None.
En el método remove() se observo cómo se explico anteriormente, que toma un solo elemento como argumento y lo elimina de la lista, pero si este elemento no existe arroja un ValueError: list.remove(x): x not in listException, como valor de retorno: None

BIBLIOGRAFIA:

Agosto 12, 2021 – Algoritmo, Programación, Python – NOAX Academy:
https://blog.noaxacademy.com/todo-lo-que-debes-saber-sobre-las-listas-en-python/

Octubre 28, 2020 – Algoritmos resueltos con Python – Enrique Edgardo Condor Tino (Univ. Nac. José María Arguedas) y Marcos Antonio de la Cruz Rocca (Univ. Nac. Daniel Alcides Carrión):
https://www.editorialeidec.com/wp-content/uploads/2020/10/Algoritmos-resueltos-con-Python.pdf

Julio 5, 2020 – Operaciones sobre listas en Python – Hanzel Godinez H.:
https://medium.com/@hgodinez89/operaciones-sobre-listas-en-python-c19853a9d07b

Link donde se encuentra el código del ejemplo planteado:
https://github.com/MairaRomina/Ejercicios-POO/tree/main/Tarea%20Investigacion