UNIVERSIDAD NACIONAL DE SAN JUAN
FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES
DEPARTAMENTO DE INFORMÁTICA

LICENCIATURA EN SISTEMAS DE INFORMACIÓN - LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN
TENICATURA UNIVERSITARIA EN PROGRAMACIÓN WEB

ASIGNATURA
PROGRAMACION ORIENTADA A OBJETOS

TAREA DE INVESTIGACION – UNIDAD 2

Alumno/a: Candela Zumel

Registro: 20576 **- Carrera:** LCC

Alumno: Peña Juan

Registro: 21151 **- Carrera:** LCC

AÑO 2021

INTRODUCCIÓN

Python es un lenguaje de programación de alto nivel, interpretado y multipropósito. En los últimos años su utilización ha ido constantemente creciendo y en la actualidad es uno de los lenguajes más empleados para el desarrollo de software en todos los sectores. Dicho lenguaje no tiene un ámbito específico, ya que puede ser utilizado para aplicaciones científicas, comunicaciones de red, aplicaciones de escritorio y web. Todo ello lo convierte en un lenguaje utilizado por grandes compañías como Google, Walt Disney, NASA, etc.

Una de las estructuras de datos más importantes de este lenguaje son las Listas, que es un conjunto modificable ordenado de objetos. Dicho de otro modo, es una estructura que contiene de cero a varios objetos, no necesariamente del mismo tipo, además de establecerse una relación de orden entre los mismos. Dicha estructura, permite gestionar sus elementos a través de determinados métodos, de los cuales se explican: insert(), sort(), extend(), pop(), index()

PALABRAS CLAVE

Python, Listas, Lenguaje de Programación, Métodos, Objetos.

DESARROLLO

Método Insert:

El método insert agrega un elemento, precisando su índice. El primer argumento es el índice del ítem delante del cual se insertará, Por Ejemplo: **a.insert(0, x)** inserta al principio de la lista el elemento x, y **a.insert(len(a), x)** equivale a **a.append(x).**

He aquí como agregar un objeto de la clase Persona, en una posición intermedia, por ejemplo:
insert (posición, elemento)

```
def agregarPersona (self, unaPersona):
      self.__listaP.append(unaPersona)
def testListaPersona (self):
          unaPersona = Persona('Roberto', 'Sanchez', 44, 'Informatico')
          otraPersona = Persona ('Julian', 'Fernandez', 33, 'Abogado')
          personaX = Persona ('Alvaro', 'Gimenez', 37, 'Docente')

    self.agregarPersona(unaPersona)
    self.agregarPersona(otraPersona)
   self.__listaP.insert (1,personaX)
```

Primero se utiliza el método de la lista agregar persona para agregar las primeras dos instancias y finalmente se utiliza insert, para agregar la ultima instancia, donde el índice se pasa como parámetro (1,personaX) .

Método Sort:

Ordena la lista por algún criterio a especificar. En caso de ser simplemente tipos primitivos (int,float, etc) simplemente hay que pasar la lista al método sort(), de esta forma devuelve una lista ordenada.

Este método puede alterar su forma de ordenación utilizando el valor verdadero de la propiedad reverse. Por ej: *sort (lista,reverse = True)*. En este caso, la lista es ordenada de forma inversa.

Para poder ordenar una lista de objetos como se presentó en los documentos, es posible mediante el uso de la función __gt__ (sobrecarga de operadores) que permite la comparación de dos datos específicos de los objetos utilizados (Persona.edad) para lograr el ordenamiento con el uso de sort() como se muestra en el ejemplo. Si no se aplica dicho método, al aplicar sort() muestra el siguiente mensaje *" '<' not supported between instances of 'Persona' and 'Persona' "*

```
class Persona:
    __nombre = ''
    __apellido = ''
    __edad = 0
    __profesion = ''
    def __init__ (self, nom= '', app='', ed=0, prof=''):
        self.__nombre = nom
        self.__apellido = app
        self.__edad = ed
        self.__profesion = prof
     def getEdad (self):
       return self.__edad

    def __gt__(self,otro):
        if (type(self) == (type(otro))):
            return self.__edad > otro.getEdad()
```

En clase ManejadorPersona se observa:

```
def testListaPersona (self):
            #Se crean las instancias persona, en este caso desde teclado, pero puede ser por archivo
csv
      print("Se ordena la lista por sort: ")
     self.__listaP.sort()
```

Método Extend:

Extend también permite agregar elementos dentro de una lista, pero a diferencia de append, al momento de agregar una lista, cada elemento de esta lista se agrega como un elemento más dentro de la otra lista.

Extiende la lista añadiendo elementos del argumento iterable. Itera sobre el argumento y luego añade cada elemento a la lista. El argumento dado debe ser de tipo iterable, como list, de lo contrario levantará TypeError.

El método extend itera los elementos en el objeto iterable y luego los añade uno a uno al final de la lista. Por Ejemplo: A.extend([unObjeto, OtroObjeto])

```
unaPersona = Persona('Roberto', 'Sanchez', 44, 'Informatico')
PersonaX2 = Persona ("Manuel","Ortega",25,"El mejor docente")
self.__listaP.extend([unaPersona, PersonaX2])
```

Habiendo eliminado previamente a unaPersona con la función pop, se vuelve a agregar junto con una nueva instancia con la función extend, que como su palabra lo dice, extiende la lista permitiendo agregar mas de un elemento, es importante colocarlos entre corchetes porque significa que corresponden a objetos de la lista.

Método Pop:

Quita el objeto en la posición dada de la lista y lo retorna. Si no se especifica un índice, a.pop() quita y retorna el último elemento de la lista. El índice i por ej., denota que el parámetro es opcional, además no es necesario agregarle corchetes, como en algunos casos.

```
from clasePersona import Persona
class ManejadorPersona:
    __listaP = []
    def __init__ (self):
        self.__listaP = []
    def agregarPersona (self, unaPersona):
        self.__listaP.append(unaPersona)
    def testListaPersona (self):
        otraPersona = Persona ('Julian', 'Fernandez', 33, 'Abogado')
        unaPersona = Persona('Roberto', 'Sanchez', 44, 'Informatico')
        perY = Persona ('Juan', 'Martin', 19, 'Estudiante')

        self.agregarPersona(unaPersona)
        self.agregarPersona(otraPersona)
        self.agregarPersona(perY)
        print("\nSe elimina la Persona que se encuentra en la primera  posicion de la lista\n")
        self.__listaP.pop(0)
```

Como se puede apreciar, se han colocado algunas instancias en el método testListaPersona, después de ello, dichas instancias son agregadas a la lista, y se prueba el método pop para eliminar un elemento a través del índice, en este caso, se elimina la primera componente.

Método Index:

El índice (index en inglés) es el número correspondiente al lugar que ocupa un objeto en la secuencia, partiendo de su inicio y siguiendo la relación de orden. Puede obtenerse, simplemente, utilizando el método index, pasándole como parámetro el objeto deseado:

```
from clasePersona import Persona
class ManejadorPersona:
    __listaP = []
```

```
    def __init__ (self):
        self.__listaP = []
def agregarPersona (self, unaPersona):
        self.__listaP.append(unaPersona)
def testListaPersona (self):
        otraPersona = Persona ('Julian', 'Fernandez', 33, 'Abogado')
        self.agregarPersona(otraPersona)
        print("\nValor de index de Persona: ", self.__listaP.index(otraPersona))
```

Se observa cómo se agrega un elemento de la clase Persona en la lista, después de ello se lleva a cabo el método index, donde este devuelve el índice que ocupa el objeto especificado por parámetro.

CONCLUSIONES

En síntesis, por medio de este trabajo han sido desarrollados algunos métodos de listas de Python, para manipular y ordenar no solo los datos primitivos que provee el lenguaje, sino también los datos definidos por el programador, en este caso, clases de objetos. La utilización de los métodos mencionados, permite una mayor optimización a la hora de ejecutar el programa y resolver una determinada problemática. Además se implementa el uso de sobrecarga de operadores para ordenar la lista en forma ascendente, por medio de métodos definidos para comparar los objetos de clase pertenecientes a una lista.

BIBLIOGRAFÍA

[1] Chazallet, S. (1900). Python 3 Los fundamentos del lenguaje. EDICIONES ENI.

[2] 5. Estructuras de datos — documentación de Python - 3.9.4. (2021). Python.org.

[3] Fernandez, A. (2013). Python 3 al descubierto - 2a ed. (1.a ed.). Alfaomega Grupo Editor.

[4] Rodríguez, D. (2020, 9 febrero). Ordenar listas de objetos en Python. Analytics Lane. https://www.analyticslane.com/2020/03/16/ordenar-listas-de-objetos-en-python/

REPOSITORIO: https://github.com/penxa01/TareaDeInvestigacion