Año 2025

Universidad Nacional de San Juan

fcefn UNSJ

Facultad de Ciencias Exactas Físicas y Naturales

Trabajo de investigación:

Lista Python y Arreglos Numpy

Cátedra:

Programación Orientada a Objetos

ALUMNO: ENZO KOKOT

DNI: 39008472

CARRERA: LCC/LSI/TUPW

REGISTRO: E010-194/ E009-102/ E014-129

ALUMNO: JORGE LLORET

DNI: 33185501

CARRERA: LCC

REGISTRO: E010-89

Trabajo de investigación lista de python y arreglo de numpy

Introducción

En el presente trabajo de investigación se analizan y describen en detalle dos tipos de datos estructurados utilizados en programación: las listas de Python y los arreglos de NumPy. Se abordarán sus características principales, diferencias, ventajas y posibles aplicaciones dentro del desarrollo de software, con el objetivo de comprender su funcionalidad y utilidad.

Palabras claves:

Numpy - np.array - lista - python - arreglos

Definición de Arreglo de Numpy

NumPy es el paquete fundamental para la computación científica en Python. Es una biblioteca que proporciona un objeto de arreglo multidimensional.

Además, varios objetos derivados (como arreglos enmascarados y matrices), y una variedad de rutinas para operaciones rápidas en arreglos, incluyendo matemáticas, lógica, manipulación de forma, ordenamiento, selección, E/S, transformadas de Fourier discretas, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

Definición de lista de python.

Las listas en Python son estructuras de datos heterogéneas y dinámicas que forman parte del núcleo del lenguaje. Las listas son secuencias mutables que pueden contener elementos de diferentes tipos y cuya implementación interna utiliza arrays dinámicos.

	Listas de Python	**Arreglos de NumPy**
Tipo de datos	Puede mezclar tipos (heterogénea)	Homogéneo por diseño, pero puede contener objetos si se define dtype=object
Flexibilidad	Muy flexible (agregar, borrar, mezclar)	Menos flexible con objetos personalizados
Acceso a elementos	Acceso por índice (lista[i])	Acceso por índice (arreglo[i])
Inserción y eliminación	Fácil: append(), insert(), remove()	Poco práctico con objetos (requiere copiar el array)

Uso principal	Datos variados	Vectorizado (rápido, pero sólo para tipos numéricos)
Rendimiento con grandes volúmenes	Menor rendimiento	Alto rendimiento para datos numéricos
Soporte para objetos personalizados	Totalmente compatible	Compatible si se usa dtype=object, pero pierde ventajas de NumPy
Tamaño	Dinámico (crece o decrece fácilmente)	Tamaño fijo, requiere crear nuevo arreglo para cambiar tamaño
Simplicidad y legibilidad	Más simple para programadores nuevos	Más técnico, requiere saber NumPy
Integración con librerías científicas	No es directa	Alta (scikit-learn, pandas, etc.)

Análisis del Código Base

El código presentado implementa un sistema de gestión de productos que utiliza un arreglo NumPy para almacenar instancias de la clase Producto. Se identificaron varios aspectos relevantes:

1. El arreglo se declara con dtype=Producto, lo que técnicamente se traduce a dtype=object, eliminando las ventajas de homogeneidad de NumPy.
2. La operación self.__datos+indiceInflacion no aplica el método __add__ de cada producto como se esperaría, demostrando una limitación en la integración objeto-arreglo.
3. Sobrecarga de memoria: El uso de arreglos NumPy con objetos Python incurre en una sobrecarga similar a las listas pero sin sus ventajas operacionales.

```
# se declara la clase Producto
class Producto:
    __descripcion: str
    __cantidad: int
    __precioUnitario: float
def __init__(self, descripcion, cantidad, precioUnitario):
    self.__descripcion=descripción
    self.__cantidad=cantidad
    self.__precioUnitario=precioUnitario
```

```
def __add__(self, incremento):
    self.__precioUnitario=self.__precioUnitario*(1+incremento/100)
def __str__(self):
    return f"descripción: {self.__descripcion}, precio: {self.__precioUnitario}"

# Utilizando Numpy
from claseProducto import Producto
# se importa la librería de numpy para trabajar con ella
import numpy as np
class ManejadorArreglo:
    __datos: None
    __cantidad: int
def __init__(self, cantidad):
# Se crea un arreglo de NumPy vacío donde se van a guardar objetos del tipo Producto
    self.__datos=np.empty(cantidad, dtype=Producto)
    self.__cantidad=cantidad
def test(self):
    unProducto=Producto("Azúcar", 400, 867.50)
    self.__datos[0]=unProducto
    otroProducto=Producto("Harina", 100, 1250)
   self.__datos[1]=otroProducto
   ultimoProducto=Producto("Arroz", 120, 2130)
  self.__datos[2]=ultimoProducto
def incremento(self,indiceInflacion):
  self.__datos+indiceInflacion
def mostrarElementos(self):
#Muestra los elementos
   for indice in range(self.__cantidad):
       print(self.__datos[indice])

# Utilizando Lista
from claseProducto import Producto
class ManejadorLista:
def __init__(self):
# Crea un atributo privado llamado __datos, y lo inicializa como una lista vacía.
     self.__datos = []
```

```
def test(self):
# Se usa el método append() para agregar un nuevo elemento al final de la lista __datos.
    self.__datos.append(Producto("Azúcar", 400, 867.50))
    self.__datos.append(Producto("Harina", 100, 1250))
    self.__datos.append(Producto("Arroz", 120, 2130))


def incremento(self, indiceInflacion):
#Este método aplica un incremento a todos los productos
    for i in range(len(self.__datos)):
        self.__datos[i] + indiceInflacion


def mostrarElementos(self):
    for prod in self.__datos:
        print(prod)

from claseProducto import Producto
from arregloPoductos import ManejadorArreglo
def test():
  unManejador=ManejadorArreglo(3)
  unManejador.test()
  unManejador.incremento(2.4)
  unManejador.mostrarElementos()
if __name__=='__main__':
  test()
```

Conclusiones

Basado en la evidencia recolectada de múltiples fuentes académicas y técnicas, se concluye que:

1. Para el manejo de objetos de clases personalizadas como en el ejemplo de Producto, las listas nativas de Python son preferibles por su mayor compatibilidad.
2. Los arreglos NumPy no proporcionan beneficios significativos cuando se trabaja con objetos personalizados.
3. El código analizado podría mejorarse sustituyendo el arreglo NumPy por una lista Python, obteniendo mayor claridad y funcionalidad sin pérdida de rendimiento. Esto se propone en el análisis del código base.