Nivel 4 Introduccion a Programacion

Tomas Rodriguez - 202212868

Mayo 2022

1 Tuplas

Son estructuras de datos lineales en las cuales cada elemento tiene un indice o una posicion, es decir funcionan como las listas. Sin embargo, se sabe que las listas son mutables, es decir, se pueden agregar, eliminar y modificar sus elementos), pero las tuplas son inmutables, es decir, sus elementos no pueden ser modificados.

1.1 Creacion de tuplas

Al ser parecidas a las listas estas se declaran de la misma manera solo que con una ligera diferencia, y es que en vez de usar los corchetes cuadrados que se vieron en el N3, se usan son los parentesis a continuacion:

```
numeros = (1,2,3)
```

Listing 1: Declaracion Tupla parentesis

Otra forma de crear tuplas es por medio del empaquetado, que funciona sin los parentesis:

```
numeros2 = 1,2,3
```

Listing 2: Empaquetado en tuplas

De igual forma como hay un empaquetado, las tuplas tienen la forma de desempaquetado:

```
x, y, z = numeros2
```

Listing 3: Desempaquedato en tuplas

En este caso las variables x, y, z tomaran los valores de $1, 2, 3$

1.2 Operaciones sobre tuplas

Al funcionar como las listas, en las tuplas se pueden usar operaciones como:

1. Indexacion

2. Slicing
3. funcion len()

2 Matrices

Son estructuras bidimensionales de que contienen datos que se organizan en filas y columnas. En python, las matrices se representan como una lista de listas:

```
matriz = [[1,2],[3,4]]
```

Listing 4: Matriz en python

Esto tendria la representacion en:

$$matriz = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2.1 Creacion de matrices

Las matrices se pueden crear usando ciclos y tecnicas de listas:

```
M = []
for i in range(0,3):
    a = [0]*6
    M.append(a)
```

Listing 5: Matriz en python con ciclos

De esta forma se combinan las tecnicas de listas como la del operador de $*$, los ciclos y el metodo append() de las listas para crear una matriz de 3×6 llena de 0. El resultado es el siguente:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.2 Operaciones sobre Matrices

Las operaciones de matrices son las mismas que en las listas, solo que en este caso, las matrices pueden contener datos mas complejos, como por ejemplo una matriz de tuplas para representar una imagen.

3 Librerias

Las librerias son extensiones de python que facilitan procesos u operaciones que en python normal, tomarian muchisimo tiempo. Algunas de las librerias mas usadas son:

- NumPy: Facilita la creacion de matrices, vectores multidimensionales junto con funciones matematicas de alto nivel.
- SciPy: Contiene herramientas y algoritmos matematicos.
- Matplotlib: Generacion de visualizaciones estaticas, animadas e interactivas.
- SymPy: Usada para matematica simbolica.
- Pandas: Manipulacion y analisis de datos.

3.1 Matplotlib

Es una libreria para crear y manipular graficos y contiene varios modulos.

3.1.1 Modulo Image

Con este modulo se puede leer una imagen y convertirla en una matriz de tuplas, donde cada tupla esta dada por (R, G, B).

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
def cargar_imagen(ruta:string)->list:
    imagen = mpimg.imread(ruta).toList()
    return imagen
```

Listing 6: Cargar una imagen

De esta forma, se carga la imagen alojada en algun directorio de nuestro pc dado por la ruta, y esta es cargada y convertida en una matriz de tuplas.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
def visualizar_image(imagen:list)->None:
    plt.imshow(imagen)
    plt.show()
```

Listing 7: Visualizar una imagen

De esta manera se puede visualizar la imagen con el metodo imshow() que indica image show.

3.1.2 Modulo Pyplot

Este modulo es uno de los mas importantes, ya que no solo es que que permite la visualizacion de las imagenes o graficos por medio del metodo show(), sino que tambien, es aquel que permite crear graficos como:

- Diagrama de lineas
- Scatter plots
- Diagramas de barras

- Diagramas de caja y bigotes o boxplot
- Graficos 3D

3.2 Pandas

Su rol mas importante esta dado que fue creado para remplazar las hojas de calculo, es decir, excel. Este nos ayuda en la limpieza, analisis y procesamiento de datos en bruto.

3.2.1 Estructuras de datos de Pandas

Pandas posee 3 estreucturas caracteristicas:

1. : Son de 1 dimension, normamento son las columnas de una tabla
2. El DataF Las Srame: Son de 2 dimensiones, es decir, representa filas y columnas como una matriz
3. El Panel: Son de 3 dimensiones y no son muy usuados.

3.2.2 Tipos de datos en pandas

Tipo en Pandas	Tipo equivalente en Python	Descripción
object	string	Es el tipo más general. Será asignado a la columna si la columna tiene valores de varios tipos (números y strings)
int64	int	Valores numéricos
float64	float	Valores numéricos con decimales. Si una columna contiene números y NaNs, pandas lo pondrá por defecto en float64.
datetime64, timedelta[ns]	N/A	Valores destinados a contener datos de tiempo

Figure 1: Tipos de datos en Pandas

3.2.3 Series

En las series, existen los indices y las posiciones o etiquetas. Estos son:

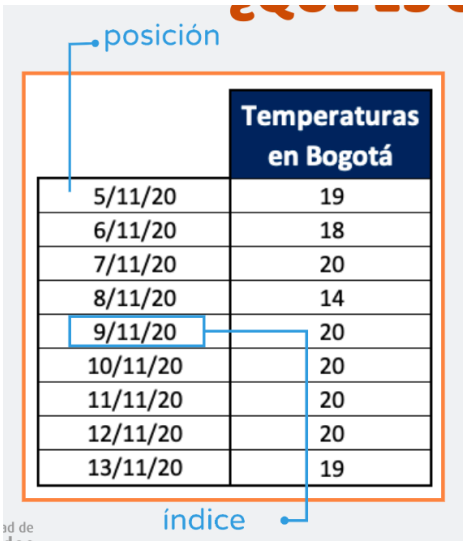

Figure 2: Indices vs Posicion

La creaion de series esta dada por el siguente codigo:

```
import pandas as pd
datos = [19,18,29,14,20]
temperaturas = pd.Series(datos, name = 'Temperaturas de Bogota')
```

Listing 8: Creacion de una Serie

Y se vera una serie de la forma:

$$\begin{bmatrix} 1 & 19 \\ 2 & 18 \\ 3 & 29 \\ 4 & 14 \\ 5 & 20 \end{bmatrix}$$

Donde los numeros del 1 al 5 indican los indices (primera columna) y los otros son los valores.

```
import pandas as pd
datos = [19,18,29,14,20]
fechas = ["5/11/20", "6/11/20", "7/11/20", "8/11/20", "9/11/20"]
temperaturas = pd.Series(datos, index=fechas, name = 'Temperaturas
    de Bogota')
```

Listing 9: Creacion de una Serie con indices

Y se vera una serie de la forma:

$$\begin{bmatrix} 5/11/20 & 19 \\ 6/11/20 & 18 \\ 7/11/20 & 29 \\ 8/11/20 & 14 \\ 9/11/20 & 20 \end{bmatrix}$$

```
parejas = {'5/11/20': 19, '6/11/20': 18, '7/11/20': 29,
'8/11/20': 14,  '9/11/20': 20}
temperaturas = pd.Series(parejas)
```

Listing 10: Creacion de una Serie con diccionarios

Y el resultado sera el mismo que el del Listing 9.
Ahora bien, existen ciertas operaciones en las Series que pueden ser utilies:

- Reconstruir indices:

```
            temperaturas = temperaturas.reset_index(drop=True)
```

Listing 11: Reconstruccion de indices

El resultado sera la serie del Listing 8.

- Extarer valor dado el indice:

```
            valor = temperaturas.get(2)
```

Listing 12: Extraer valor de un indice

El resultado sera el elemnto al cual pertenece el indice 2, es decir, 18.

- Extraer valores usando indices:

```
            valores = temperaturas.loc[2:4]
```

Listing 13: Extraer valores de indices

El resultado sera:

$$\begin{bmatrix} 2 & 18 \\ 3 & 29 \\ 4 & 14 \end{bmatrix}$$

- Extraer valores usando posiciones:

```
            valores2 = temperaturas.iloc[2:4]
```

Listing 14: Extraer valores de posiciones

El resultado sera:

$$\begin{bmatrix} 3 & 29 \\ 4 & 14 \end{bmatrix}$$

- Extraer todos los valores en una lista:

```
valoresLista = temperaturas.values
```

Listing 15: Extraer todos los valores

El resultado sera:

[19 18 29 14 20]

- Hacer una copia de la Serie:

```
copia _= temperaturas.copy()
```

Listing 16: Hacer copia de una serie

El resultado sera una copia identica a la serie pero almacenada en otro lugar en memoria

- Operaciones estadisticas:

```
temperaturas.max()
temperaturas.min()
temperaturas.mean()
temperaturas.std()
temperaturas.median()
```

Listing 17: Operaciones Estadisticas

3.2.4 DataFrames

Los dataframes son muchos registros ordenados por columnas,m la cual cada una posee un tipo. Se pueden construir dataframes a partir de:

- Lista de diccionarios
- Diccionario de listas
- Diccionario de series
- Archivos csv

pasra filtrar los datos y extarer las columnas de interes de un dataframe grande a uno mas pequeños se haria de la siguente manera:

```
columnas_valores = ['nombres de las columnas separados por comas']
valores = dataframe_original[columnas_valores].copy()
```

Listing 18: Copia Dataframe con columnas especificas

Aparte de estas operaciones, los dataframes tienen operaciones de analisis basico:

- head(n): Deja ver las n primeras filas del dataframe

- tail(): Deja ver los ultimos 5 registros o filas del dataframe
- info(): Da informacion acerca de cada columna del dataframe
- describe(): Describe valores importantes del dataframe
- Filtrado de columnas:

```
dataframe[nombre columna]
dataframe[[nombre columna 1, nombre columna 2]]
```

Listing 19: Filtrado de columnas

En este filtrado, se puede hacer por una o varias columnas.

- unique(): Extrae un array con los valores que aparecen al menos una vez en la columna por la cula se filtra.
- value_counts(): Cuenta cuantas veces aparece el valor en la columna filtrada
- Metodos estadisticos: mean(), mode(), std(), max(), min(), idxmax(), idxmin()
- Metodos matematicos: sum(), prod(), div(). Se aplican a columnas numericas como los estadisticos.
- ordenamiento: Se ordena el dataframe por los valores de una de las columnas:

```
ordenados = dataframe.sort_values('columna')
```

Listing 20: Ordenamiento por una columna

En estos casos ocurrira que los valores de los indices no corresponden a las posiciones.

- iloc[inicio:fin]: Es el mismo de la seccion de Series
- loc[inicio:fin]: Es el mismo de la seccion de Series
- Seleccion con condiciones: Es el mismo filtrado por columna, sin embargo, en este caso se usaran expresiones booleanas:

```
dataframe[dataframe['columna'] == algo]
```

Listing 21: Filtrado Booleano

En este caso se puede usar cualquier expresion booleana vista en el N2. Otra opcion es un filtrado multiple que se da de la siguente manera:

```
dataframe[(dataframe['columna 1'] == algo) | (
    dataframe['columna 2'] > algo mas)]
dataframe[(dataframe['columna 1'] == algo) & (
    dataframe['columna 2'] > algo mas)]
```

Listing 22: Filtrado Booleano multiple

En este filtrado multiple son cruciales los parentesis que separan las expresiones booleanas y los simbolos de | y & significan **OR** y **AND**

- Agrupamientos: Agrupar los datos por los valores de una columna:

```
agrupado = dataframe.groupby('columna')
```

Listing 23: Agrupamiento

y asi como se pueden agrupar los valores de la columna, asi mismo pueden ser llamados los datos:

```
valor_columna_df = agrupado.get_group('valor_columna')
```

Listing 24: Obtencion de grupo

Asi como en el filtrado, las agrupaciones pueden ser multiples:

```
agrupado = dataframe.groupby(['columna1','columna2
    '])
```

Listing 25: Agrupamiento multiple