INFORME 3 INFORMATICA 3

Clase No: 1

Título: MODULOS Y PAQUETES EN PYTHON

Fecha: 12/10/2022

RESUMEN.

En este capítulo o clase vimos lo que son los distintos módulos y paquetes de py, haciendo énfasis en el uso de estos para diferentes aplicaciones y además conociendo más afondo la estructura que estos elementos importantes a la hora de programar, introduciendo un poco a concepto tenemos que un *módulo* es un archivo de Python cuyos objetos (funciones, clases, excepciones, etc.) pueden ser accedidos desde otro archivo.

Marco teórico.

MODULOS: Un *módulo* es un archivo de Python cuyos objetos (funciones, clases, excepciones, etc.) pueden ser accedidos desde otro archivo. Se trata simplemente de una forma de organizar grandes códigos.

Ejemplos de módulos: definimos el módulo esteesmimodulo.py con cualquiera de estas dos funciones: sum(), rest().

Para crear los siguientes dos módulos, por un lado uno llamado interfaz.py y otro llamado logica.py . Después un archivo principal main.py donde pueda ejecutar las funciones de los módulos anteriores.

Modulo interfaz

```
def imprimirMensaje(nombre):
    mensaje = "hola, soy holaMundo otra vez" + nombre
    print(mensaje)
    return None

def imprimirSeparador(tipo):
    print("\n" + tipo * 50 + "\n")

def imprimirVariable(nombre, variable):
    print(nombre + " ==> " + str(variable))

def imprimirListado(lista):
    for elemento in lista:
        imprimirSeparador("-")
        print(elemento)
```

Modulo Lógica: Sirve para realizar operaciones y se almacenan algunas variables, contiene dos funciones

```
def sumar2Nums(numero1, numero2):
    resultado = numero1 + numero2
    return resultado

def sumarNNumeros(*numeros): #numeros se interpreta como un listado
    suma = sum(numeros)
    return suma

##### Para hacer el testeo del modulos ####
if __name__ == "__main__":
    a = sumar2Nums(1,2)
    print(a)
    b = sumarNNumeros(1,2,3,4,5,6,7,8)
    print(b)
```

A continuación obtenemos el documento principal main.py, y se ejecutan las funciones ya mencionadas.

```
from termios import FF1
import interfaz
import logica

#print(logica.__doc__)
#print(interfaz.__doc__)

#print(dir(interfaz))

interfaz.imprimirSeparador("#")
interfaz.imprimirMensaje("Cristian Pachon")
interfaz.imprimirSeparador("#")

variable1 = logica.sumar2Nums(2,3)
interfaz.imprimirVariable("suma2Nums", variable1)
interfaz.imprimirSeparador("-")

variable2 = logica.sumarNNumeros(2,3,4)
interfaz.imprimirVariable("sumaNNumeros", variable2)
interfaz.imprimirSeparador("-")
```

Clase No.2

Fecha: 26/10/2002

Título: MODULOS Y PAQUETES EN PYTHON

Resumen: En esta clase vamos a ver módulos incorporados en py para que el programador pueda desarrollar códigos más óptimos, también vimos que funciones cumplen dentro de la programación y como Python puede facilitarnos algunos de ellos.

Marco teórico.

Módulos: En el siguiente cuadro podremos ver algunos de los módulos más utilizados en py, en la clase anterior los habíamos definido ahora veremos que significan cada uno de ellos.

math:	Provee funciones matemáticas útiles y constantes numéricas, pi, euler
random	Generador de números aleatorios, con sus respectivas distribuciones estadísticas
skitlearn:	Funciones y herramientas útiles hacer machine learning
pandas:	Permite hacer hojas de cálculo fáciles de manipular semejantes a excel y estructuras sql
tqdm:	Herramienta para que el programador estime el tiempo de ejecución de un programa
matplotlib:	Generador de distintos tipos de graficas, basado en matlab
numpy:	Provee estructuras de datos potentes
os:	Para realizar operaciones en el terminal desde python
sys:	Organiza la ubicación de las herramientas, módulos funciones, o paquetes de Python

Ejemplos:

- Con este código podemos imprimir las variables de constantes matemáticas como pi y Euler:

```
import math
print("pi => ", math.pi)
print("euler => ", math.e)
print("seno(3.14/2) => ", math.sin(math.pi/2))
```

- Con el código a continuación podemos imprimir el calendario.

```
4  import calendar
5
6  print(calendar.calendar(2022))
```

Clase No.3

Fecha: 28/10/2022

Título: Estructuras de datos en py.

Resumen: En este capítulo vamos a ver estructuras de py para analizar datos, como es su lenguaje y de qué manera las podemos utilizar ya que este tipo de estructuras de datos son demasiado útiles en el ámbito de la programación científica.

Marco teórico.

Las estructuras de datos en Python se pueden entender como un tipo de dato compuesto, debido a que en una misma variable podemos almacenar una estructura completa con información. Dichas estructuras, pueden tener diferentes características y funcionalidades. De hecho, existen múltiples tipos de estructuras de datos en Python
Las estructuras de datos más comunes en Python son las listas, las tablas y los diccionarios. Aunque tienen otro nombre, en otros lenguajes, son los arreglos o vectores, las matrices y los arreglos indexados, respectivamente. Son en esencia lo mismo, aunque como es habitual en Python, con varias facilidades y funcionalidades ya incluidas.

En esta serie de contenidos vamos entonces a explorar cada una de esas estructuras de datos en Python (listas, tablas y diccionarios), en ese mismo orden

Las listas en Python forman parte de lo que se conoce como estructuras de datos que nos ofrece Python (son un tipo de array). En otros lenguajes de programación, se los conoce como vectores.
Las listas, son una estructura de datos de vital importancia, pues son útiles para resolver múltiples problemas que nunca podríamos solucionar sin ellas. Las listas en Python son utilizadas para almacenar múltiples valores en una única variable. Esto puede parecer innecesario, pero en realidad resuelve muchos desafíos. Veamos.

En múltiples ocasiones es necesario almacenar gran cantidad de información en una variable y a menudo sucede que no conocemos con exactitud la cantidad de datos que debemos almacenar en ella. Sin embargo, sabemos que sí sería más de uno, como por ejemplo almacenar los nombres de las personas ingresadas en tu programa. En este caso, las listas nos permiten solucionar este tipo de problemas, pues en una misma lista podremos almacenar todos esos valores, sin importar cuántos sean.

Ejemplos:

- **Numpy:**

```
import numpy as np

matriz1 = np.array([[1, 1, 1],
                    [1, 1, 1],
                    [1, 1, 1]])

matriz2 = np.array([[1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9]])
```

```
5   import numpy as np
6   tensor1= np.array([ [[0, 0, 0],
7                        [0, 0, 0],
8                        [0, 0, 0]],
9
10                      [[2, 2, 2],
11                       [2, 2, 2],
12                       [2, 2, 2]]
13                  ])
14  tensor2= np.array([ [[1,  2,  3],
15                       [4,  5,  6],
16                       [7,  8,  9]],
17
18                      [[10, 11, 12],
19                       [13, 14, 15],
20                       [16, 17, 18]]
```

¿Cómo crear un diccionario en Python?

Para declarar un diccionario en Python, se deben seguir las mismas normas básicas que se siguen para declarar una lista, PERO en lugar de usar corchetes "[]" usaremos llaves "{}" para declararlos:

```
nombre_del_diccionario = {}
```

```
otro_diccionario = {
   "nombre": "Alberto",
   "usuario": "alb_123",
}
```

Una vez el diccionario es creado, podemos agregarle elementos, quitarlos, obtenerlos, y varias cosas más, tal como hacemos con las listas (usando los mismos métodos).

Clase No.4

Fecha: 02/11/2022

Título: La estructura y tipos de Arrays de numpy.

Resumen: En esta clase conocimos la estructuro y los tipos de arrays ya que estos son importantes para los iterables y las funciones que pueden actuar en ella, además aprendimos a manejar su estructura y a darle algunas aplicaciones.

Marco teorico.

A continuación veremos una definición corta de Los arrays de numpy, ya que estas estructuras dan soporte a los iterables de py. Poseen mayor cantidad funcionalidades, están optimizadas y son de gran utilidad para el cálculo numérico. Mediante los arreglos, es posible crear vectores, matrices y tensores.

Numpy dispone de lo siguiente para trabajar:

- paquete de extensión Python para matrices multidimensionales
- cercano al hardware (eficiente)
- diseñado para cálculo científico (conveniente)

Funciones integradas de numpy: En la siguiente tabla resumiremos como se organizan los arreglos de numpy y como clasificarlos:

● **Creacion de arreglos** - np.array(<iterable>) - np.arange(<inicio>,<fin>,<salto>) - np.ones((<filas>,<columnas>)) - np.zeros(<filas>, <columnas>)	● **Redimensionamiento** - np.reshape((<filas>, <columnas>))	● **Apilamiento** - np.hstack((<elemento1>, <elemento2>)) - np.hstack((<elemento1>, <elemento2>))
● **Indexado y slicing** - vector[<columna>] - matriz[<fila>,<columna>] - tensor[<profundidad>,<fila>,<columna>]	● **Almacenamiento** - np.savetxt("ruta", <arreglo>) - np.loadtxt("ruta")	

Valor minimo arreglo.min()	**Valor máximo** arreglo.max()
Desviacion estándar arreglo.std()	**Suma del arreglo** arreglo.sum()
Promedio arreglo.mean()	

Ejemplos: Ahora veremos este código que indica cómo acceder a valores en 3 columnas distintas de vector1

```
print("------------indexado------------ \n\n")
value1 = vector1[0]
value2 = vector1[2]
value3 = vector1[-1]
print(value1, value2, value3)
```

- código en el cual se acceder al valor ubicado en la fila 1 (indice 0), columna 3 (indice 2) de matriz2"

```
value1 = matriz2[0,2]
print(matriz2, value1)
value2 = matriz2[1,1]
print(matriz2, value2)
```

- código que apila el vector a la matriz y luego extrae una sección compuesta por la fila 2 (índice 1) y fila 3 (índice 2):

```
matriz = np.arange(1,16).reshape(3,5)
vector =  np.array([1,2,3]).reshape(3,1)
print(matriz, "\n",vector)
matrizApilada = np.hstack((matriz, vector))
print(matrizApilada)
#Realizar extraer seccion
```

Métodos estadísticos mean, std, min, max

Ejemplos

- En este código calculamos la media de los vectores 1 y 2

```
print("medias => ",vector1.mean(), vector2.mean())
```

- En este código que calcula la media de matriz 2 por fila, por columna y total

```
print(matriz2)
print(matriz2.mean())        # media de todos los valores
print(matriz2.mean(axis=0))  # axis 0 es por columnas,
print(matriz2.mean(axis=1))  # axis 1 es por filas
```

Clase No.5

Fecha: 04/11/2022.

Título: Series en Python.

Resumen: En este capítulo de clase aprendimos una nueva estructura de datos denominada series en Python aprendimos su lenguaje y como utilizarla para las diferentes funciones que vamos a ocupar en estructuras y análisis de datos.

Marco teórico.

SERIE: Una serie es una estructura unidimensional eficiente. Se construye sobre un arreglo numpy y está compuesto por índices numericos (0,1,2,3...) e índices claves ("Enero", "Febrero",...)

Además se pueden indexar, opcionalmente utilizando ya sea índices numéricos o índices claves.

A continuacion vamos a ver como creamos una serie:

```
import pandas as pd


serie = pd.Series(data = <arregloNumpy>, index = [....])
serie = pd.Series(data = np.Array([5,4,3.5]),  index = ["Estu 1", "Estu 2", "Estu 3"])
```

Herramientas de una serie: En el siguiente cuadro vamos a resumir las herramientas de una serie.

ATRIBUTOS	• serie.index : retorna los indices clave • serie.values : retora la data

	• serie.shape : retorna el tamaño
METODOS	• serie.describe() : devuelve otra serie, describiendo la serie original • serie.mean() : devuelve la media • serie.std(): devuelve la desviacion estandar • serie.min(): devuelve el valor minimo • serie.max(): devuelve el valor maximo • serie.idxmin(): devuelve la clave del minimo • serie.idxmax(): devuelve la clave del maximo • serie.value_counts(): devuelve las frecuencias de la serie
INDEXADO	• serie[0] • serie["Enero"]

Ejemplos

- código que crea una serie

```
import pandas as pd
import numpy as np

index = ["001", "002", "003", "004", "005", "006", "007", "008", "009", "010", "011", "012", "013", "014", "015"]
data =np.array([22, 3,  21, 10, 15, 18, 14, 13, 22, 12, 98, 32, 51, 45, 60])
serieVentas = pd.Series(data = data, index=index)
print(serieVentas)
```

- código que Imprime los índices, data y tamaño de la serieVentas utilizando sus atributos.

```
print(serieVentas)
print("indices => ", serieVentas.index)
print("data =>", serieVentas.values)
print("tamaño =>", serieVentas.shape)
```

Clase No. 6

Fecha: 09/11/2022.

Título: Data frames.

Resumen: En este capítulo vimos cómo utilizar estructuras de datos en 2D llamados Data frames y aprendimos su estructura para ser manejado en diferentes funciones que puede usar el programador.

Marco teórico:

Data frames : Si Dataframes es el tipo de datos fundamental de la librería *pandas*, está claro que la habilidad principal que debemos tener con esta librería es **la manera de crear un** Dataframes **a partir de datos**. Son estructuras de datos en 2D. (Filas y columnas) De manera semejante a los archivos excel, sql, csv.

Nota: funcionan como las hojas de excel, Para indexar se trabaja por indice numerico o por indice clave.

¿Cómo crear un DataFrame de pandas?

```
import pandas as pd


hoja1 = pd.DataFrames(data = <arreglo 2D numpy>,
                      columns = ["columna1", ...., "columnaN"],
                      index = ["fila1",..., "filaN" ])
```

Ejemplo:

- código que crea una serie

```
import numpy as np
import pandas as pd

data = np.array([[1, 12000],
                 [2, 15000],
                 [3, 21000],
                 [2, 32000],
                 [1, 11000],
                 [2, 90000]])

hoja1 = pd.DataFrame( data = data,
                      columns = ["tamaño", "precio"],
                      index = ["001","002","003","004","005","006"])
print(hoja1)
```

- como crear columnas en data frames:

Ten en cuenta que la dimensión principal del DataFrame son las columnas, con lo que el acceso a las columnas siempre es un poco más directo que al de las filas. De hecho, usando la típica notación de corchetes accedemos antes a las columnas que a las filas, al contrario de lo que es habitual.

Una manera de añadir una nueva columna a un DataFrame es asignarle directamente los valores que debe tener esa columna, tal como haríamos en un diccionario y la notación de corchetes. Como en este caso no queremos introducir valores indico simplemente None.

```
df['Nombre'] = None
print(df)
```

```
Empty DataFrame

Columns: [Nombre]
```

```
Index: []
```

Cómo insertar datos en un *DataFrame*

Una vez que tenemos un Dataframes ya creado con sus columnas solo nos queda poder añadirle algunos datos. Existen múltiples formas de hacerlo. Vamos a ver un par de ellas.

Supongamos que tenemos los datos a insertar en listas, es decir, una lista para cada columna con los valores de cada fila para esa columna**.** Podemos hacer una asignación sencilla de la siguiente manera:

```
nombres = ['Juan', 'Laura', 'Pepe']
edades = [42, 40, 37]

df['Nombre'] = nombres
df['Edad'] = edades

print(df)
```

El resultado de esta operación será el siguiente:

```
Nombre  Edad
0   Juan    42
1  Laura    40
2   Pepe    37
```

Clase No7

Fecha: 11/11/2022

Título: Manipulación de datos.

RESUMEN: En esta clase vimos cómo usar herramientas para la programación orientada al manejo de datos, vimos cómo es importante determinar valores estadísticos para poder llegar al campo de la investigación y usar herramientas poderosas para el análisis de datos.

Marco teórico:

Manipulación de datos

Para poder manipular los datos, se utilizarán herramientas presentes en los arreglos, series y dataframes . Es importante determinar algunos valores estadisticos : media, mediana, desviacion, valor min/max ...

Ejemplos: código en el cual se visualiza la información, para ello se emplea matplotlib

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(x, y, style)  #donde x, y contienen la data,
plt.show()             #style: el estilo y color de linea
```

Métodos numéricos para calcular algunas derivadas:

La derivada es la pendiente a la recta tangente que pasa por una curva determinada por unos límites o cierto intervalo que pertenezca al dominio de la función derivada.

$$f'(x) = (y2-y1) / (x2 -x1)$$

Ejemplos:

- código con dataFrame de 3 columnas: ["F1", "F2", "F3"]

```
import pandas as pd
import numpy as np

"""
Creamos un dataFrame con 3 columnas: ["F1", "F2", "F3"]
                        100 indices: [0.01, 0.02 ..0.99, 1] (valores de x)

F1(x) = xsin(x) + 0.2*random(x)
F2(x) = cos(x) +  0.1*random(x)
F3(x) = x + 1/2
 """

x = np.arange(0.01, 1.01, 0.01)
columna1 = x * np.sin(x) + 0.2 * np.random.rand(100)
columna2 = np.cos(x) + 0.1 * np.random.rand(100)
columna3 = x * 1/2
data = np.hstack((columna1.reshape(100,1), columna2.reshape(100,1), columna3.reshape(100,1)))

hoja1 = pd.DataFrame(data= data,
                     index= np.arange(0.01, 1.01, 0.01),
                     columns= ["F1", "F2", "F3"]
)
```

```
import matplotlib.pyplot as plt

"""
Graficar F1(x), F2(x), F3(x)
"""

def graficaGenerica(x, y, marca):
    plt.figure(figsize=(8,4))  #Para crear lienzo vacio
    plt.plot(x, y, marca)
    plt.xlabel("x")
    plt.title("f(x) DataFrame")
    plt.ylabel("f(x)")
    plt.grid()
    plt.show()     #Para mostrar la figura

x = hoja1.index
y = hoja1["F1"]
graficaGenerica(x, y, "or")

x = hoja1.index
y = hoja1["F2"]
graficaGenerica(x, y, "*b")

x = hoja1.index
y = hoja1["F3"]
graficaGenerica(x, y, "-k")
```

```
import matplotlib.pyplot as plt
def graficaGenerica(x, y, y_prima, titulo):
    plt.figure(figsize=(8,4))
    plt.title(titulo)
    plt.plot(x,y, "b", label="funcion f(x)")
    plt.plot(x[:-1], y_prima, "r", label="derivada f '(x)")
    plt.xlabel("x")
    plt.legend()  #para mostrar los labels f(x) y f '(x)
    plt.show()

x = hoja1.index
y = hoja1["f1"]
y_prima = derivada1
graficaGenerica(x, y, y_prima, "f(x) = xsen(x)")

x = hoja1.index
y = hoja1["f2"]
y_prima = derivada2
graficaGenerica(x, y, y_prima, "f(x) = cos(x)")

x = hoja1.index
y = hoja1["f3"]
y_prima = derivada3
graficaGenerica(x, y, y_prima, "f(x) = x + 1/2")
```