# RELATORIA 3

November 17, 2022

## 1 RELATORIA 3 INFORMATICA 2

### 1.1 OBJETIVOS

### 1.2 13/10/2022

#### 1.2.1 MODULOS

Es la manera en la cual es posible mejorar la modularidad de un programa. Permite organizar mis funciones en diferentes temáticas, con el objeto de facilitar la escritura de código.

**¿cómo creo un módulo?** un modulo python simplemente es un archivo.py el nombre del modulo es el nombre del archivo

un modulo puede contener lo siguiente definido:

- documentacion
- variables
- funciones
- clases

Sin embargo, un modulo no debe tener ejecuciones de código.

Crear 2 modulos:

- interfaz
- logica

cree una variable y 3 funciones para interfaz -cantidadDeFunciones -separador -imprimirNombre -imprimirVariable

cree 3 funciones para logica que hagan lo siguiente: -sumar 3 numeros -sumar n numeros -sumar dos listas, elemento a elemento

Finalmente consuma todas las funciones y variables creadas, desde otro modulo (main.py) main.py será el único archivo desde donde se pueden consumir los modulos anteriores

[ ]:
```
"""
Este modulo se llama interfaz,
mediante el cual se imprime un separador,
un nombre, y unas variables de manera más
presentable utilizando funciones

```

```
tambien se almacena 1 variable
"""

cantidadDeFunciones = 3

def separador(caracter):
    sep = caracter * 50
    print(sep)

def imprimirNombre(nombre):
    saludo = "hola,mi nombre es interfaz y el tuyo " + nombre
    print(saludo)

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

[ ]:
```
"""
Este modulo contiene 3 funciones de lógica para hacer
lo siguiente:

* sumar 3 numeros
* sumar n numeros
* sumar 2 listas, elemento a elemento
"""

def sumar3Numeros(numero1, numero2, numero3): #entrada de la funcion
    resultado = numero1 + numero2 + numero3
    return resultado  #salida de la funcion

def sumarNumeros(*numeros):  #numeros se interpreta como lista
    #resultado = 0
    #for numero in numeros:
    #    resultado = resultado + numero
    resultado = sum(numeros)
    return resultado

def sumarListas(lista1, lista2):
    listaResultado = []
    for indice in range(0, len(lista1)):
        suma = lista1[indice] + lista2[indice]
        listaResultado.append(suma)
    return listaResultado
```

[ ]:
```
import interfaz


print(dir(interfaz))
```

```
print(interfaz.__doc__)
print(interfaz.__file__)

interfaz.separador("@")
interfaz.separador("-")
interfaz.imprimirNombre("Cristian")
respuesta = [1,2,3]
interfaz.imprimirVariable("Respuesta", respuesta)


import logica
resultado1 = logica.sumar3Numeros(1,2,3)
resultado2 = logica.sumarNumeros(1,2,3,4,5,6,7,8,9)
resultado3 = logica.sumarListas([1,2,3], [3,2,1])

interfaz.separador("-")
interfaz.imprimirVariable("sumar 3 numeros", resultado1)
interfaz.imprimirVariable("sumar N numeros", resultado2)
interfaz.imprimirVariable("sumar 2 listas", resultado3)
interfaz.separador("-")
```

### 1.3 18/10/2022

programar el juego triki o 3 en raya usando 3 modulos:

interfaz.py (es la parte visual y que da mensajes de turnos o ganador )
logica.py (como determinar quien gana? como guardar X o O?)
control.py (cerebro del juego, consume interfaz y logica y ordena la secuencia del juego )

[ ]:
```
"""este modulo es la logica y con este modulo haremos la parte logica del juego
de 3 en linea. tendremos funciones de obtener tablero logico, actualizar el␣
 ↪tablero logico,
determinar ganador y determinar si la posicion de una jugada esta ocupada """

def obtenerTableroLogico():

    tableroLogico=[None,None,None,None,None,None,None,None,None]

    return tableroLogico

def actualizarTableroLogico(tableroLogico, posicion, caracter):

    tableroLogico[posicion]=caracter

    return tableroLogico

def determinarGanador(tableroLogico):
```

```
    ␣
 ↪posicionesParaGanar=[(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    ganador=None
    for p1, p2, p3 in posicionesParaGanar:
        if (tableroLogico[p1] == tableroLogico[p2] == tableroLogico[p3]):
            if tableroLogico[p1] != None:
                ganador= tableroLogico[p1]

    return ganador

if __name__== "__main__":
    tablero = obtenerTableroLogico()
    tableroNuevo = actualizarTableroLogico(tablero, 0, "x")
    print(tableroNuevo)
    ganador =  determinarGanador(["x", "x", "x", None, None,None,None,None,None,␣
 ↪])
    print(ganador)

def determinarPosicionOcupada(tableroLogico, posicion):
    if tableroLogico[posicion] == None:
        validez=True
    else:
        validez=False
    return validez
```

[ ]:
```
"""este modulo es la interfaz y con este modulo intentaremos crear la parte␣
 ↪visual del juego triky
aqui deberá haber una pantalla de juego, decir quien tiene el turno y quien es␣
 ↪el ganador """

def explicarJuego():
    explicacion="""
    Este es el juego 3 en linea

    Es un juego de 2 jugadores, para ganar debe completar 3 caracteres iguales␣
 ↪en linea ("X"o "O")

    Para ingresar la posición en el juego, guiese con los siguientes numeros

    (0) | (1) | (2)
    ____|_____|_______
    (3) | (4) | (5)
    ____|_____|_______
    (6) | (7) | (8)

    """
    print(explicacion)
```

```
    input("...ingrese enter para empezar el juego....")


tableroLista = ["x", "o", None, None,None,None,None,None,None]

def dibujarTablero(tableroLogico:list):
    tableroVisual = """
                {}  | {} |  {}
                -------------
                {}  | {} |  {}
                -------------
                {}  | {} |  {}

    """.format(tableroLogico[0], tableroLogico[1],
               tableroLogico[2], tableroLogico[3],
               tableroLogico[4], tableroLogico[5],
               tableroLogico[6], tableroLogico[7],
               tableroLogico[8])

    print(tableroVisual)
    return tableroLogico

if __name__ == "__main__":
    explicarJuego()
    tablero1 = ["None"]*9
    tablero2= ["X"]*9

    dibujarTablero(tablero1)
    dibujarTablero(tablero2)

def imprimirTurno(caracter: str):

    if caracter == "None":
        print("es turno de X")
    elif caracter == "X":
        print("es turno de O")
    elif caracter == "O":
        print("es turno de X")
    else:
        print("Este caracter no está permitido")
```

[ ]:
```
import interfaz
import logica

interfaz.explicarJuego()
tableroJuego= logica.obtenerTableroLogico()
```

```
for turno in ["X","O","X","O","X","O","X","O","X"]:
    print("\nTurno jugador: ", turno)
    posicion= int(input("  Ingrese posicion de juego:  "))
    if logica.determinarPosicionOcupada(tableroJuego, posicion)== True:
        tableroLogico= logica.actualizarTableroLogico(tableroJuego, posicion,␣
↪turno)
        interfaz.dibujarTablero(tableroJuego)
        posibleGanador=logica.determinarGanador(tableroJuego)
        if posibleGanador in ["X","O"]:
            print("Felicidades ha ganado el juego jugador ", turno)
            break
    else:
        print("la posicion ingresada no es valida ")
```

## 1.4 01/11/2022

### 1.4.1 CLASES Y OBJETOS

Los objetos, clases y métodos son parte de un paradigma de programacion llamado: programacion orientada a objetos (POO). Que es diferente al anterior visto: programación funcional.

– La POO, consiste en modelar (abstraer) cualquier problema en: clases, objetos (instancias) y metodos (funciones) – La POO permite relacionar entidades distintas (estudiante, materias, profesores)

**DEFINICIONES INDISPENSABLES PARA ENTERNDER POO** CLASE: Es un modelo o plantilla genérica a partir del cual se crean objetos. Una clase dota a un objeto con dos ingredientes: Atributos Y METODOS

OBJETO O INSTANCIA: Es una entidad creada a partir de una clase, dotada con atributos y metodos

ATRIBUTOS O PROPIEDADES: Son aquellos datos que caracterizan a un objeto, se obtienen a partir de un clase generica. Sin embargo, cada objeto se puede diferenciar de otro, al atribuirle atributos distintos

METODOS: Son aquellas funcionalidades de los objetos, se heredan a partir de una clase genérica.

INSTANCIAR: Actividad escencial de la POO, consiste en crear objetos (instancias) a partir de una clase

**EJEMPLOS**

1) Determinar las clases, objetos, atributos y métodos de los siguientes elementos de python

- Cadenas de texto
- Los numeros enteros
- Secuencias de números
-

### 1.5 Cadenas de texto

CLASE => str (string) OBJETOS => “hola mundo” | “Unal” | “Reprobado2” ATRIBUTOS => Tamaño 10 | Tamaño 4 | Tamaño 10 Es Alfabetico | Es Alfabetico | Es alfa-numérico Posee minusculas | Posee mayus y minus | Posee mayus MÉTODOS => +, , *len()* / +, , len() | +, *, len()

•

### 1.6 Los numeros enteros

CLASE => int (entero) OBJETOS => 2 | 19 | 100 ATRIBUTOS => Posee 1 dígito | Posee 2 dígitos | Posee 3 dígitos Es primo | Es primo | Es no primo Es par | Es Impar | Es par MÉTODOS => +,-,, % / +,-,, % | +-*%

•

### 1.7 Secuencias de numeros

CLASE => list (lista) OBJETOS => [1.0 ,3.0 ,4.0] | [] | [0,0] ATRIBUTOS => Tamaño 3 | Tamaño 0 | Tamaño 2 Mayor es el 4.0 | No posee mayor | Mayor es el 0 Posee flotantes | No posee elementos | Posee enteros MÉTODOS => append,len,min.. | append,len,min.. | append,len,min..

2) Determinar las clases, objetos, atributos y métodos de los siguientes elementosde la vida real

- Bandeja paisa, sancocho de gallina, ajiaco
- BMW-I8, FERRARI-458, Un AutoLegal
- Profesora Elisabeth Restrepo, profesor Luis Fernando Mulcue, profesor Cristian Pachon

CLASE => Comida Colombiana OBJETOS => Bandeja paisa | Sancocho de gallina | Ajiaco ATRIBUTOS => chorizo | gallina | papa arepa | papa | pollo chicharron | caldo | aguacate MÉTODOS => nutrir, provocar,| nutrir, provocar | nutrir, provocar => enamorar | enamorar | enamorar

CLASE => Automoviles OBJETOS => FERRARI-458 | MCLAREN-720S | AUTOLEGAL ATRIBUTOS => Color rojo | color negro | Color blanco 2-personas | 2-personas | Las que quiera el conductor Deportivo | Deportivo | Transporte público MÉTODOS => Acelerar, frenar | Acelerar, frenar | Acelerar, frenar enamorar | enamorar | enamorar

CLASE => Profesores Unal OBJETOS => Elisabeth | Mulcue | Cristian Pachon ATRIBUTOS => Mujer | Hombre | Hombre Mayor de 30 | Mayor de 30 | Menor de 30 Exitosa | Exitoso | Exitoso MÉTODOS => Enseñar, evaluar | Enseñar, evaluar | Enseñar, evaluar

**COMO CREAR MIS PROPIAS CLASES EN PYTHON** Las clases en python, se definen utilizando las palabras reservadas:

clase, objetos, atributos, metodos class ==> Avisa a python que se creará una clase genérica (comida, autos, profesores.....) def **init**(self, ValorAtributo) ==> Mecanismo mediante el cual creamos atributos genéricos, se usa solo una vez def metodo(self, valorEntrada) ==> Mecanismo mediante el cual creamos métodos, se usa las veces que quiera, dependiendo de la cantidad de funcionalidades de esa clase self ==> Permite hacer referencia al objeto creado

**NOTACION** class : def **init**(self, ): self. =

def alimentar(self, ): return

def (self, ): return

..............

def (self, ): return

[ ]:
```
# 08-11-2022

"""
Crear una clase llamada ComidaColombiana
atributos: ingrediente1, ingrediente2, ingrediente3
metodos: nutrir y provocar
"""

class ComidaColombiana:
    #Para crear los atributos
    def __init__(self, ingrediente1,ingrediente2, ingrediente3):
        self.ingrediente1 = ingrediente1
        self.ingrediente2 = ingrediente2
        self.ingrediente3 = ingrediente3

    #Para crear metodos (funciones)
    def provocar(self, opcion):
        if opcion in ["huele bien", "se ve bien", "tengo hambre"]:
            return "Este alimento provoca"
        else:
            return "Este alimento no provoca"

    def nutrir(self, opcion):
        if opcion in ["estoy enfermo", "tengo nauseas", "el doctor me prohibio"]:
            return "Este almiento no me puede nutrir"
        else:
            return "Este alimento me nutre"


#Como creo el objeto ?
bandejaPaisa = ComidaColombiana("chorizo", "arepa", "chicharron")
sancochoDeGallina = ComidaColombiana("gallina", "papa", "caldo")
ajiacoSantafereño = ComidaColombiana("papa", "pollo", "aguacate")
print("Es instancia? =>", isinstance(ajiacoSantafereño, ComidaColombiana))

#Cómo accedo a los atributos de un objeto
atributoA = bandejaPaisa.ingrediente3
atributoB = sancochoDeGallina.ingrediente1
atributoC = ajiacoSantafereño.ingrediente2
```

```
print("atributos =>", atributoA, atributoB, atributoC)


#Cómo acceder a los métodos de un objetoingrediente3
salida1 = bandejaPaisa.provocar("tengo hambre")
salida2 = ajiacoSantafereño.nutrir("Muero de hambre")
salida3 = sancochoDeGallina.provocar("huele mal")
print("salidas =>", salida1, salida2, salida3, sep=" -- ")


"""
Ejercicio:
    1) Crear una clase llamada Profesores
        con los atributos: nombre, edad, salario
        con los metodos: enseñar, calificar
    2) Crear una clase llamada Decimal
       con un unico atributo: valorNumerico
       con los metodos convertirABinario, conventirAOctal,␣
 ↪convertirAHexadecimal,
"""
```

[ ]:
```
#10-11-2022

"""
Ejercicio:
    1)  Crear una clase llamada Profesores
        con los atributos: nombre, edad, salario
        con los metodos: enseñar, calificar
"""


class Profesores:
    def __init__(self, nombre, edad, salario):
        self.nombre = nombre
        self.edad = edad
        self.salario = salario

    def enseñar(self, calidad):
        return "estoy enseñando al {} %".format(calidad)

    def calificar(self, estadoAnimo):
        if estadoAnimo == "feliz":
            return "su no ta es 5.0"
        elif estadoAnimo == "triste":
            return "su nota es 0.0"

if __name__ == "__main__":
```

```
    profesor1 = Profesores("Elisabeth Restrepo", 20, 15000000)
    profesor2 = Profesores("Luis Mulcue", 50, 5000000)

    print(profesor1.nombre)
    print(profesor2.salario)
    print(profesor1.enseñar(20))
    print(profesor2.calificar("triste"))

    """Ejercicio:
    2) Crear una clase llamada Decimal
       con un unico atributo: valorNumerico
       con los metodos convertirABinario, conventirAOctal,␣
 ↪convertirAHexadecimal,  """


class Decimal:
    def __init__(self, valorNumerico):
        self.valorNumerico = valorNumerico

    def convertirABinario(self):
        conversion = bin(self.valorNumerico)
        return conversion[2:]

    def conventirAOctal(self):
        conversion = oct(self.valorNumerico)
        return conversion[2:]

    def convertirAHexadecimal(self):
        conversion = hex(self.valorNumerico)
        return conversion[2:]

if __name__ == "__main__":
    decimal1 = Decimal(8)
    decimal2 = Decimal(15)
    print("Decimal 8")
    print(decimal1.convertirABinario())
    print(decimal1.conventirAOctal())
    print(decimal1.convertirAHexadecimal())

    print("Decimal 15")
    print(decimal2.convertirABinario())
    print(decimal2.conventirAOctal())
    print(decimal2.convertirAHexadecimal())



"""
```

```
crear las clases Decimal,Binario Octal, Hexadecimal
e incluya los metodos necesarios para realizar las conversiones
a los otros sistemas numericos
"""
```

## 1.8 15/11/2022

### 1.8.1 Persistencia de datos

Es el alamacenamiento de la informacion de manera permanente Es almacenar la informacion en un archivo Es almacenar la informacion en disco duro

Extensiones para almacenar datos: .txt .xsml .csv .pdf .doc .sql .json (almacenamiento que utilizaremos)

**¿Qué es json?** Es un formato para almacenar datos tipo objeto javaScript

import json

with open(ruta, opcion) as file: <===Lectura notas = json.load(file)

with open(ruta, opcion) as file: <==Escritura json.dump(data, file)

[ ]:
```
#15-11-2022

#---------- Escritura de archivos json

data = list(range(1,1000,2))

import json
nombreArchivo = "DATA.json"
ruta = "EJERCICIOS/14_PersistenciaDatos/" + nombreArchivo
opcion = "w" #Escritura
with open(ruta, opcion) as file:
    json.dump(data, file)


"""
Almacene la siguiente informacion en un archivo estudiantes.json

   Nombre          Nota1   Nota2  Nota3
Maria Gonzalez       3.1    3.1    1.2
Camilo Suarez        3.2    4.0    1.1
Esteban Rodriguez    3.2    4.1    2.2
Mariana Rosero       5.0    5.0    5.0
Jose Nuñez           5.0    4.0    2.5
Esteban Quesada      3.4    4.0    2.6
Mauricio Velazquez   5.0    4.2    2.1
Julia Quintero       2.0    2.2    2.1
Mauricio Lizcano     3.7    4.1    4.7
```

```
Miguel Pineda        1.0    1.1    3.3
Angie Gomez          4.1    4.7    4.4
Angelica Lozano      3.1    1.0    2.6
Camilo Restrepo      5.0    5.0    1.0
"""


data = {
            "Maria Gonzalez"     :  [3.1,    3.1,    1.2],
            "Camilo Suarez"      :  [3.2,    4.0,    1.1],
            "Esteban Rodriguez"  :  [3.2,    4.1,    2.2],
            "Mariana Rosero"     :  [5.0,    5.0,    5.0],
            "Jose Nuñez"         :  [5.0,    4.0,    2.5],
            "Esteban Quesada"    :  [3.4,    4.0,    2.6],
            "Mauricio Velazquez" :  [5.0,    4.2,    2.1],
            "Julia Quintero"     :  [2.0,    2.2,    2.1],
            "Mauricio Lizcano"   :  [3.7,    4.1,    4.7],
            "Miguel Pineda"      :  [1.0,    1.1,    3.3],
            "Angie Gomez"        :  [4.1,    4.7,    4.4],
            "Angelica Lozano"    :  [3.1,    1.0,    2.6],
            "Camilo Restrepo"    :  [5.0,    5.0,    1.0]
        }

nombreArchivo = "estudiantes.json"
ruta = "EJERCICIOS/14_PersistenciaDatos/" + nombreArchivo
opcion = "w"

with open(ruta, opcion) as file:
    json.dump(data, file)
```

[ ]:
```
# DATA.json
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41,␣
 ↪43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79,␣
 ↪81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113,␣
 ↪115, 117, 119, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 143,␣
 ↪145, 147, 149, 151, 153, 155, 157, 159, 161, 163, 165, 167, 169, 171, 173,␣
 ↪175, 177, 179, 181, 183, 185, 187, 189, 191, 193, 195, 197, 199, 201, 203,␣
 ↪205, 207, 209, 211, 213, 215, 217, 219, 221, 223, 225, 227, 229, 231, 233,␣
 ↪235, 237, 239, 241, 243, 245, 247, 249, 251, 253, 255, 257, 259, 261, 263,␣
 ↪265, 267, 269, 271, 273, 275, 277, 279, 281, 283, 285, 287, 289, 291, 293,␣
 ↪295, 297, 299, 301, 303, 305, 307, 309, 311, 313, 315, 317, 319, 321, 323,␣
 ↪325, 327, 329, 331, 333, 335, 337, 339, 341, 343, 345, 347, 349, 351, 353,␣
 ↪355, 357, 359, 361, 363, 365, 367, 369, 371, 373, 375, 377, 379, 381, 383,␣
 ↪385, 387, 389, 391, 393, 395, 397, 399, 401, 403, 405, 407, 409, 411, 413,␣
 ↪415, 417, 419, 421, 423, 425, 427, 429, 431, 433, 435, 437, 439, 441, 443,␣
 ↪445, 447, 449, 451, 453, 455, 457, 459, 461, 463, 465, 467, 469, 471, 473,␣
 ↪475, 477, 479, 481, 483, 485, 487, 489, 491, 493, 495, 497, 499, 501, 503,␣
 ↪505, 507, 509, 511, 513, 515, 517, 519, 521, 523, 525, 527, 529, 531, 533,␣
 ↪535, 537, 539, 541, 543, 545, 547, 549, 551, 553, 555, 557, 559, 561, 563,␣
 ↪565, 567, 569, 571, 573, 575, 577, 579, 581, 583, 585, 587, 589, 591, 593,␣
 ↪595, 597, 599, 601, 603, 605, 607, 609, 611, 613, 615, 617, 619, 621, 623,␣
 ↪625, 627, 629, 631, 633, 635, 637, 639, 641, 643, 645, 647, 649, 651, 653,␣
 ↪655, 657, 659, 661, 663, 665, 667, 669, 671, 673, 675, 677, 679, 681, 683,␣
 ↪685, 687, 689, 691, 693, 695, 697, 699, 701, 703, 705, 707, 709, 711, 713,␣
 ↪715, 717, 719, 721, 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 743,␣
 ↪745, 747, 749, 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771, 773,␣
 ↪775, 777, 779, 781, 783, 785, 787, 789, 791, 793, 795, 797, 799, 801, 803,␣
 ↪805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 825, 827, 829, 831, 833,␣
 ↪835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859, 861, 863,␣
 ↪865, 867, 869, 871, 873, 875, 877, 879, 881, 883, 885, 887, 889, 891, 893,␣
 ↪895, 897, 899, 901, 903, 905, 907, 909, 911, 913, 915, 917, 919, 921, 923,␣
 ↪925, 927, 929, 931, 933, 935, 937, 939, 941, 943, 945, 947, 949, 951, 953,␣
 ↪955, 957, 959, 961, 963, 965, 967, 969, 971, 973, 975, 977, 979, 981, 983,␣
 ↪985, 987, 989, 991, 993, 995, 997, 999]
```

[ ]:
```
#estudiantes.json
{"Maria Gonzalez": [3.1, 3.1, 1.2], "Camilo Suarez": [3.2, 4.0, 1.1], "Esteban␣
 ↪Rodriguez": [3.2, 4.1, 2.2], "Mariana Rosero": [5.0, 5.0, 5.0], "Jose␣
 ↪Nu\u00f1ez": [5.0, 4.0, 2.5], "Esteban Quesada": [3.4, 4.0, 2.6], "Mauricio␣
 ↪Velazquez": [5.0, 4.2, 2.1], "Julia Quintero": [2.0, 2.2, 2.1], "Mauricio␣
 ↪Lizcano": [3.7, 4.1, 4.7], "Miguel Pineda": [1.0, 1.1, 3.3], "Angie Gomez": [4.
 ↪1, 4.7, 4.4], "Angelica Lozano": [3.1, 1.0, 2.6], "Camilo Restrepo": [5.0, 5.
 ↪0, 1.0]}
```

[ ]:
```
#15-11-2022

#---------- Lectura de archivos json
```

```python
"""Leer el archivo DATA.json
y luego calcular la media de los valores
"""

import json
archivo = "DATA.json"
ruta = "EJERCICIOS/14_PersistenciaDatos/" + archivo
opcion = "r" #lectura

with open(ruta, opcion) as file:
    data = json.load(file)

print("media=>", sum(data)/len(data))


"""
Leer el archivo estudiantes.json y luego calcular el promedio
de cada uno de los estudiantes
"""

archivo = "estudiantes.json"
ruta = "EJERCICIOS/14_PersistenciaDatos/" + archivo
opcion = "r"

with open(ruta, opcion) as file:
    data = json.load(file)

for nombre in data.keys():
    print(nombre, " => ", sum(data[nombre])/3)
```

## 1.9 CURSOS EXTRA

### 1.9.1 INTRODUCCIÓN A LAS CIENCIAS DE LA COMPUTACIÓN

**Tipo Abstracto de Datos** Con la programación orientada a objetos, sabemos que debemos darle a una clase en especial, unas ciertas propiedades esto lo hacemos con:

def **init**(self, iterable): super().__init__(str(item) for item in iterable)

hagamos un ejercicio.

(1) Implementar el TAD Polinomio, que se describe a continuación, representando internamente cada objeto abstracto (polinomio) con elementos del TAD Lista. Un polinomio, con coeficientes enteros $c_i$ , en la variable x : $P(x) = c_0 + c_1x + c_2x^2 + + c_nx^n$

se puede representar como una lista de coeficientes, $[c_0, c_1, c_2, ..., c_n]$ donde el valor del exponente va implícito por su posición dentro de la lista. Por ejemplo, el polinomio: $P(x) = 12x^5 + 3x^3x + 6$ se representa con la lista $[6, 1, 0, 3, 0, 12]$

Las operaciones del TAD Polinomio son las siguientes:

evalPolinomio: evalua un polinomio en un entero dado. sumarPolinomio: retorna la suma de dos polinomios. multiplicarPolinomio: retorna la multiplicación de dos polinomios. derivarPolinomio: retorna el polinomio que resulta de derivar un polinomio. integrarPolinomio: retorna el polinomio que resulta de integrar un polinomio. igualPolinomio: determina si dos polinomios son iguales

evalPolinomio: Polinomio × Integer ! Integer sumarPolinomio: Polinomio × Polinomio ! Polinomio multiplicarPolinomio: Polinomio × Polinomio ! Polinomio derivarPolinomio: Polinomio ! Polinomio integrarPolinomio: Polinomio ! Polinomio igualPolinomio: Polinomio × Polinomio ! Boolean

(2) Otra manera de representar polinomios como listas es utilizando, para cada término, una pareja de la forma (coeficiente, exponente), y así, colocar explícitamente el valor del exponente de cada elemento. Por ejemplo, el polinomio $P(x) = 12x^5 + 3x^3x + 6$ se representa con la lista $[(12, 5), (3, 3), (1, 1), (6, 0)]$ en la cual no existe un orden determinado entre sus componentes.
• Utilizando esta nueva representación implementar el TAD Polinomio.

[ ]:
```
"""
1. En el siguiente ejercicio vamos a tener una serie de funciones aplicadas a␣
 ↪polinomios,
   para esto es necesario que tenga en cuenta que la forma de un polinomio es:
   P(x) = c0 + c1x + c2x^2 + . . . + cnx^n
   recuerde que el signo " + " tiene espacio a ambos lados y que los␣
 ↪coeficientes estan a
   la izquierda de la " x ", esto se hace para que sea mas comodo para usted

"""



n=int( input("introduzca el grado del polinomio"))
pol=[]
for i in range(0,n+1):
    pol.append(int(input('ingrese el coeficiente del polinomio: ')))


class ejercicio1(list) :

    def evalPolinomio(pol: list, n: int):

        res=pol[0]
        for i in range(1,len(pol)):
            res=res+ pol[i]*(n**i)
        res= int(res)
        return res



    def sumarPolinomio(pol1: list,pol2: list):
```

```
        sum=[]
        sum.append(pol1[0] + pol2[0])
        if len(pol1)==len(pol2):
            for i in range(1,len(pol1)):
                sum.append( pol1[i] + pol2[i])
        elif len(pol1) < len(pol2):
            for i in range(1,len(pol1)):
                sum.append(pol1[i] + pol2[i])
            for i in range(len(pol1),len(pol2)):
                sum.append(pol2[i])
        else:
            for i in range(1,len(pol2)):
                sum.append(pol1[i] + pol2[i])
            for i in range(len(pol2),len(pol1)):
                sum.append(pol1[i])

        return sum


    def derivarPolinomio(pol: list):
        derivada=[]
        for q in range(1, len(pol)):
            derivada.append(q*pol[q])
        return derivada


    def integrarPolinomio(nomio: list):
        integral=["C"]
        for q in range(0, len(nomio)):
            integral.append(nomio[q]/(q+1))
        return integral

    def multiplicarPolinomio(pol1: list,pol2: list):
         producto=[]
         x=len(pol1)-1
         y=len(pol2)-1
         for j in range(x+y+1):
            k=0
         for i in range(j+1):
            if 0<=i<=x and 0<=j-i<=y:
                k+=pol1[i]*pol2[j-i]
            producto.append(k)
         return producto

    def igualPolinomio(pol1: list,pol2: list):
        veracite= True
        if len(pol1)==len(pol2):
```

```
            for j in range(0,len(pol1)):
                veracite =bool( veracite*(pol1[j] == pol2[j]))
        else:
            veracite= False

        return veracite




pol=[]
c= 0
while c!='':
    c=input("introduzca el coeficiente: ")
    n=input("introduzca el exponente: ")
    if c!='':
        pol.append((int(c),int(n)))

class ejercicio2:

    def evalPolinomio(pol: list, m: int):
        res=0
        for i in range(0,len(pol)):
            res=res+ pol[i][0]*(m**pol[i][1])
        res= int(res)
        return res


    def sumarPolinomio(pol1: int, pol2: int):
        sum=[]

        for i in range(0,len(pol1)):
            for j in range(0,len(pol2)):
                if pol1[i][1] == pol2[j][1]:
                    sum.append( (pol1[i][0] + pol2[j][0],pol1[i][1]))

            if pol1[i][1] != pol2[i][1]:
                sum.append(( pol2[i][0],pol2[i][1]))
                sum.append((pol1[i][0],pol1[i][1]))

        return sum


    def derivarPolinomio(pol: list):
        derivada=[]
        for q in range(0, len(pol)):
            if pol[q][1] != 0:
```

```
                derivada.append((pol[q][1]*pol[q][0], pol[q][1]-1 ))
        return derivada


    def integrarPolinomio(pol: list):
        integral=[("C",0)]
        for q in range(0, len(pol)):
            integral.append((pol[q][0]/(pol[q][1]+1), pol[q][1]+1 ))
        return integral


    def igualPolinomio(pol1: list,pol2: list):
        veracite= True
        if len(pol1)==len(pol2):
            for j in range(0,len(pol1)):
                veracite =bool( veracite*((pol1[j][0] ==␣
↪pol2[j][0])and(pol1[j][1] == pol2[j][1])))
        else:
            veracite= False

        return veracite

    def multiplicarPolinomio(nomio1: list,nomio2: list):
        product=[]
        x=len(nomio1)-1
        y=len(nomio2)-1
        for j in range(x+y+1):
            k=0
            e=0
            for i in range(j+1):
                if 0<=i<=x and 0<=j-i<=y:
                    k+=nomio1[i][0]*nomio2[j-i][0]
                    e=nomio1[i][1]+nomio2[j-i][1]
            product.append((k,e))
        return product


y= ejercicio2.multiplicarPolinomio([(1,0),(4,2)],[(4,3),(3,2),(7,0)])
print(y)
```

**Tuplas** Las tuplas son más rápidas que las listas. Si define un conjunto constante de valores y todo lo que va a hacer es iterar sobre ellos, use una tupla en lugar de una lista.

Una tupla es una secuencia de items ordenada e inmutable.

Los items de una tupla pueden ser objetos de cualquier tipo.

Para especificar una tupla, lo hacemos con los elementos separados por comas dentro de paréntesis.

Una tupla con únicamente dos elementos es denominada par.

Para crear una tupla con un único elemento (singleton), se añade una coma al final de la expresión.

Para definir una tupla vacía, se emplean unos paréntesis vacíos.

**Listas** Una lista es una secuencia ordenada de elementos mutable.

Los items de una lista pueden ser objetos de distintos tipos.

Para especificar una lista se indican los elementos separados por comas en el interior de CORCHETES.

Para denotar una lista vacía se emplean dos corchetes vacíos.

**Diccionarios** Los diccionarios de Python son una lista de consulta de términos de los cuales se proporcionan valores asociados. En Python, un diccionario es una colección no-ordenada de valores que son accedidos a traves de una clave. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipo. Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior. No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves. La informacion almacenada en los diccionarios, no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario. Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

tanto las Tuplas como las Listas y los Diccionarios , ya los hemos visto, pero no esta demas ver sus propiedades, bondades y contras. ya hemos implementado muchos ejemplos entonces no ahondaremos mas.

**Funciones** Python es un lenguaje indentado, no usa corchetes para delimitar el alcance de las estructuras de programación sino que se fija en los cambios de indentación.

No se declara el tipo de los argumentos de las funciones. La semática de la implementación ha de estar preparada para funcionar con los tipos de datos que quieres.

por otro lado podemos jugar mas con las funciones.

[ ]:
```
import numpy as np
def funcion_1(a,b):
    r = a**2
    return r+b

def greatest(a,b):
    if a>b:
        return a
    else:
        return b
```

```
m1 = np.array([[3,4],[1,1]])
m2 = np.array([[5,6],[0,0]])
print(funcion_1 (10.4,2))
print(funcion_1 (10.4, np.array([2,4])))
print(funcion_1 (m1,m2))
print(greatest(10,2))
```

Podemos definir valores por defecto para los argumentos de las funciones y llamarlas usando explícitamente el nombre de los argumentos. Además, las funciones pueden devolver varios valores.

[ ]:
```
import numpy as np
def f_power(x, p=2):
    return x**p, x*p

r = f_power(p=4, x=3)
print(r[0],r[1])

r1, r2 = f_power(p=4, x=3)
print(r1)
print(r2)


a,b = 10, np.array([10,4,-3])
print(a)
print(b)
```

**Paso por Valor y Referencia** Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

Paso por valor: Se crea una copia local de la variable dentro de la función.

Paso por referencia: Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente:

Los tipos simples se pasan por valor: Enteros, flotantes, cadenas, lógicos...

Los tipos compuestos se pasan por referencia: Listas, diccionarios, conjuntos...

Ejemplo de paso por valor:

Como ya sabemos los números se pasan por valor y crean una copia dentro de la función, por eso no les afecta externamente lo que hagamos con ellos:

[ ]:
```
def doblar_valor(x):
    x = 2*x
    return x
x=10
doblar_valor(x)
```

```
print(x,doblar_valor(x))

# Para modificar los tipos simples podemos devolverlos modificados y␣
 ↪reasignarlos:
x = 10
x = doblar_valor(x)
print(x)
```

Ejemplo de paso por referencia:

Sin embargo las listas u otras colecciones, al ser tipos compuestos se pasan por referencia, y si las modificamos dentro de la función estaremos modificándolas también fuera:

[ ]:
```
def doblar_valores(y):
    for i,n in enumerate(y):
        y[i] = 2*y[i]

y = [10,50,100]
doblar_valores(y)
print(y)

# Para el caso de los tipos compuestos, podemos evitar la modificación enviando␣
 ↪una copia:

y = [10,50,100]
doblar_valores(y[:])  # Una rapida de una lista con [:]
print(y)
```

**Matrices y Vectores** Buscar un elemento de un arreglo: dado un arreglo con componentes $A[1], ..., A[n]$ y un valor x, el siguiente algoritmo retorna en la variable q el valor False si no existe k tal que $A[k] = x$; en caso contrario, retorna en la variable q el valor True y en la variable i el valor k

[ ]:
```
import numpy as np
A=np.array(['a','b','c','d','e'])
x='e'
i=0
q=True
n=len(A)
while q and (i!=n) :
  q=not(A[i]==x)
  i=i+1
if i==n and q==True:
    print('No se encontró el elemento')
else:
    print('la posición es:',i-1,'El valor es:',A[i-1])
```

otra forma es: 1) se le adiciona una componente al arreglo A 2) a la nueva componente se le asigna el valor x, que acotua como centinela para la finalizacion de la busqueda

[ ]:
```
A=np.array(['a','b','c','d','e'])
x='z'
n=len(A)
A=np.append(A,x)

i=0
while not(A[i]==x):
  i=i+1
if i==n:
    print('No se encontró el elemento')
else:
    print('la posición es:',i,'El valor es:',A[i])
```

Producto Escalar:

dados las sucesiones de numeros $(x_1, ..., x_n)$ y $(y_1, ..., y_n)$ calcular el producto escalar

teniendo en cuenta la definicion de esta suma en terminos de la relacion de recurrencia

$s_i = s_{i-1} + x_i * y_i$

se tiene el siguiente programa

[ ]:
```
v1=[1,2,3,4]
v2=[4,5,6,7]
n=len(v1)
s=0
i=0
while i!=n:
  s=s+v1[i]*v2[i]
  i+=1
print('el producto escalar es: ',s)
```

**Numpy** Con la librería numpy se trabaja con matrices de forma natural. Fíjate cómo se declaran y cómo descubrimos sus dimensiones.

[ ]:
```
import numpy as np

a = np.array([[1,2,3,4,5],
              [5,4,3,2,1],
              [9,8,7,6,5],
              [7,6,5,6,7],
              [2,2,2,3,3],
              [4,3,4,3,4],
              [5,1,1,4,1]]).astype('int32')
print(a, type(a),a.nbytes)
print("a shape", a.shape)
print("a rows", a.shape[0])
print("a cols", a.shape[1])
```

Con la notación de índices accedemos a columas o filas enteras, rangos de columnas o filas, elementos individuales o porciones de una matriz o un vector.

Muchas funciones de la librería numpy operan sobre una matriz completa, o de forma separada por columnas o filas según el valor del argumento axis.

[ ]:
```
print("una fila        " ,a[2])
print("una fila        ",a[2,:])
print("una columna     ",a[:,2])
print("un elemento     ",a[2,2])
print("varias filas    \n",a[2:5])
print("varias columnas \n",a[:,1:3])
print("una porcion     \n",a[2:5,1:3])

print(a)
print("suma total", np.sum(a))
print("suma eje 0", np.sum(a, axis=0))
print("suma eje 1", np.sum(a, axis=1))
print("promedio total", np.mean(a))
print("promedio eje 0", np.mean(a, axis=0))
print("promedio eje 1", np.mean(a, axis=1))
```

**Generacion de Matrices y Vectores**

[ ]:
```
print("matrix identidad\n", np.eye(3))
print("vector de ceros", np.zeros(4))
print("matriz de ceros\n", np.zeros((3,2)))
print("matriz de unos\n", np.ones((2,3)))
print("vector rango", np.arange(10))
print("vector rango", np.arange(5,10))
print("vector espacio lineal", np.linspace(0,7,5))
print("matriz aleatoria según distribución uniforme [0,1]\n", np.random.
 ↪random(size=(3,5)))
print("vector aleatorio de enteros entre 0 y 5", np.random.randint(5, size=10))
```

**Operaciones Vectorizadas** Las operaciones vectorizadas también funcionan con expresiones booleanas. Fíjate cómo se indexa un vector con una expresión booleana para seleccionar un conjunto de elementos.

[ ]:
```
import numpy as np
v = np.array([10,12,13,15,20])
a =  np.random.randint(100, size=5)
print(v)
print(a)
print(v+1)
print(v*2)
print(v.dot(a))
print(np.cross(v1,v2))
```

```
m1=np.array([[1,2],[3,4]])
m2=np.array([[5,6],[7,8]])

print(m1*m2)
print(m1.dot(m2))
print(m1.T)
print(a,a>4,a[a>4])
```

**Expresiones Compactas** Fíjate cómo las siguientes expresiones son equivalentes:

[ ]:
```
a=15
if a > 10:
    s = "mayor que 10"
else:
    s = "menor que 10"

print(s)
#vs
s = "mayor que 10" if a > 10 else "menor que 10"
print(s)

l=[]
for i in range(5):
    l.append(i)
print(l)
#vs
l=[i for i in range(5)]
print(l)

a = [10, -4, 20, 5]

#o = ["10A", "-4B", "20A", "5A"]

o = []
for i in a:
    if i<0:
        o.append(str(i)+"B")
    else:
        o.append(str(i)+"A")

print(o)
#vs
a = [10, -4, 20, 5]
def convert(x):
    return str(x)+"B" if x<0 else str(x)+"A"

o = [convert(i) for i in a]
```

```
print(o)
```

### Matplotlib

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

x = np.linspace(-4,4,20)# El vector de tabulación
y=x**2
z=x**3

plt.plot(x, y, label="x^2 plot")
plt.scatter(x, y, label="muestras")
plt.plot(x, z, label="$x^3$", color="red")
plt.xlim([-4,4])
plt.ylim([-15, 15])
plt.legend()
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Gráficas')


x = np.linspace(-4,4,20)
y=x**2
plt.figure(figsize=(5,5))
plt.scatter(x, y)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Parabola')

plt.figure(figsize=(5,5))
plt.plot(x, x**3)
plt.grid(True)
plt.xlabel('eje X')
plt.ylabel('eje y')
plt.title('Cúbica')

%matplotlib inline
plt.figure(figsize=(5,5))
x = np.linspace(-4,4,10)


plt.plot(x, x**2, color="black", linewidth=5)
plt.scatter(x, x**2, c="green", s=300)
```

```
x_r = np.linspace(-4,4,100)
x_ruido = x_r**2 + (np.random.random(x_r.shape)-0.5)*10
plt.scatter(x_r,x_ruido, c="red", alpha=0.2)
```