Hoss- Versión: -7

01/10/2024

```
# -*- coding: utf-8 -*-
"""
Created on Sábado 24/06/2024
```

ph_principal.py

```
HOSS (Human Organizations Simulation Start)
José Quintás Alonso
@author: invat
"""
import mysql.connector
import ph_herramientas
import ph_pantallasES
import ph_sociedad
import ph_ERED
import ph_seleDist
import ph_salidas
import random

from sinfo import sinfo

import warnings

warnings.filterwarnings('ignore')

#estructuras globales. Inicializar variables

#nombrePh tiene como clave el id y como valor el puntero al objeto nacido
nombrePh={}
```

#nombreOrg tiene como clave el id y como valor el puntero al objeto Organizacion creada

nombreOrg={}

datosIniciales=[] #Poblacion Inicial, año de inicio, etc...

datosReprPoblPib={} #iteracion,Poblacion, Nacimientos, defunciones, PIB

tiempo=1 #tiempo Global,tiempoabsoluto, en todo caso, comenzará por UNO: Poblacion inicial

#Diccionario individual que se guarda en tabla MySQL, puede cambiar anualmente

miDicActividades={1:'Comer',2:'Pareja',3:'Hijos',4:'Descanso',5:'Sexo',6:'Salud',7:'Amistad',8:'Estudio',\

9:'Preparar oficio',10:'Redes sociales',11:'Vestirte',12:'Casa',13:'Coche',\

14:'Milit pol-sindi',15:'Deporte',16:'Adicciones', 17:'Escribir leer',18:'Asac recreati',\

19:'Satis Curiosid',20:'Jugar Azar',21:'Familia',22:'Manualidades',23:'Militancia ONG',24:' Arte Ciencia',\

25:'Viajar Emigrar', 26:'Seguir la moda',27:'Ejercer poder',28:'Investigar',\

29:'Emprender',30:'Deporte riesgo',31:'Lider público',32:'Espiritualidad',33:'Estudios medios',34:'Estudios superiores'}

diccioArquet={'amigo':[7,16],'cuidador':[23] ,'explorador':[25] ,'heroe':[15,30] ,\

'inocente':[26] ,'rebelde':[14] ,'sabio': [19],'gobernante':[27,31],\

'creador':[17,24,28] ,'bufon':[18],'mago':[8,29],'amante':[5]}

#Diccionario individual que se guarda en tabla MySQL y da su Arquetipo y Caracter de gestación

miDicPhBasi={'amigo':0,'cuidador':0 ,'explorador':0 ,'heroe':0 , 'amante':0 ,\

'inocente':0 ,'rebelde':0 ,'sabio':0,'gobernante':0,'creador':0 ,'bufon':0,'mago':0, \

'arquetipo1':0,'arquetipo2':0, 'caracter1':0,'caracter2':0 }

miDicPhBasiModelo = {'amigo':0,'cuidador':0 ,'explorador':0 ,'heroe':0 , 'amante':0 ,\

```
        'inocente':0 ,'rebelde':0 ,'sabio':0,'gobernante':0,'creador':0 ,'bufon':0,'mago':0, \
       'arquetipo1':0,'arquetipo2':0, 'caracter1':0,'caracter2':0 }

diccioCaract={'amorfo':[16],'apasionado':[27,31] ,'apatico':[26] ,'colerico':[17,29,30] ,\
         'flematico':[8,15,19,24,28] ,'sanguineo':[7,14] ,'sentimental':[23,25,32] }

listaNoAsignadas=[2,3,5,10,12,13,20,21,22,33,34]   #lista de importantes actividades

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
  user="invat",password="jquintas49",auth_plugin="mysql_native_password")

class ph():
  def __init__(self,dni):
    self.dni=dni
    self.inicioAbsoluto=0 # Debe marcar en que iteración nacio este ph(): por ej naci en la iteracion 1949 (por eso tengo 75 años, dado que estamos en 2024)
    self.edad=0  # su edad . Ej:  75 años;
    self.iteracion = 0 #iteracion actual, normalmente la llamo: tiempo en el main menu
    self.sexo='' #biologico
    self.intencionalidad='No plantea' #Reforzar, Búsqueda, Flotar, Abandono
    self.idFiscalFamilia=0
    self.dniPareja=0  # dni pareja reproductora actual
    self.idFiscalEmpresa=0 #Empresa en la que trabaja
    self.riesgoTolerancia=0 #toleracia personal al riesgo en general; cero: intolerante al riesgo: aversión;
    self.idPadre=0 #dato que no cambia. -1 si es de la PI
    self.idMadre=0 #dato que no cambia. -1 si es de la Población Inicial
    self.coefiInt=0 #coeficiente inteligencia
    self.resiliencia=0
    self.potencial='0-Ba-Me'
    self.atraPers='N' #se le calculado en nacer y nacerPI
```

```
    self.sumaVa3Año=0.0 #de cada actividad ejecutada anualmente, suma de va3 -indice de éxito

    self.sumaVa9Año=0.0 #de cada actividad ejecutada anualmente, suma de va9 -índice de eficacia

    self.situlaboposi=1 #situacion laboral, cuando naces: infancia (Menos en PI!!) . Los tres campos se guardan en tabla iteracionecon

    self.antiguedad=0 #antiguedad en esa situlaboposi

    self.climLabo='Bien' #clima laboral (incluida la infancia, jubilación...) MBien,Bien,Mal,MMaL

    self.palp=0 #ahorros, inversiones, activos... Actualmente es el AHORRO acumulaado

    self.gastoAnyo = 0  #en formaGasto().- el gasto del año en curso

    self.ingrxRBU=0   #¿cuanto cobra en total?: ingrxTrab + ingrxRBU

    self.ingrxTrabB = 0 #solo salario.

    self.plusProductividad =0 #se renuava cada año, sube o baja...incluido en ingrxTrab

    self.ingrxTrab=0  #incluye la parte salarial ingrxTrabB + plusProductividad

    self.ingrTotal= self.ingrxTrab + self.ingrxRBU

    # ¿Cuanto es la parte salarial? = ingrxTrab - plusProductividad

    #diccionarios INDIVIDUALES

    self.miDicPhBasi={'amigo':0,'cuidador':0 ,'explorador':0 ,'heroe':0 , 'amante':0 ,\

          'inocente':0 ,'rebelde':0 ,'sabio':0,'gobernante':0,'creador':0 ,'bufon':0,'mago':0, \

          'arquetipo1':0,'arquetipo2':0, 'caracter1':0,'caracter2':0 }

    self.miDicPh={1:['X',0,0,0,0,0,0,0,0,0]}


  def getDatos(self):

    self.dni,  self.inicioAbsoluto, self.edad, self.iteracion,self.sexo,  self.intencionalidad, self.idFiscalFamilia, \

       self.dniPareja,self.idFiscalEmpresa, self.riesgoTolerancia, self.idPadre, self.idMadre, self.coefiInt, self.resiliencia,\

       self.potencial,self.atraPers, self.sumaVa3Año,self.sumaVa9Año,self.situlaboposi ,self.antiguedad, self.climLabo, self.palp,self.gastoAnyo,\

       self.plusProductividad, self.ingrxTrabB,self.ingrxTrab, self.ingrxRBU, self.ingrTotal,self.miDicPhBasi,self.miDicPh

    return
```

```
class Organizacion():
   def __init__(self,dniFiscal):
      self.dniFis=dniFiscal
      self.tipo='' #pyme, empresa, familia, narcos, secta, ejercito, estado, banco, universidad
      self.creacion=0  #iteracion global de creacion, tiempo global
      self.liquidacion=0 #iteracion global de liquidacion, divorcio
      self.objetivo='' #proposito, objetivo principal
      self.ingresos=0
      self.gastos=0
      self.inversion=0
      self.deuda=0
      self.caracter='' #Pu-publico, Co-concertado,Pr-privado.- La financiación pública a Concertada es por servicio a ph() concreto
      #diccionario INDIVIDUAL
      self.miDicOrg={1:['X',0,0,0,0,0,0,0,0,'X']}

   def getDatos(self):
      self.dniFis,self.tipo, self.creacion,self.liquidacion,self.objetivo,\
      self.ingresos,self.gastos,self.inversion,self.deuda,self.caracter,self.miDicOrg
      return
#FIN declaraciones

#CAJA DE HERRAMIENTAS
def borrado():
   # Borra los datos de toda tabla de MySQL
   miCursor=miConexion.cursor()
   sql= 'delete FROM tercerbd.desastres;'
   miCursor.execute(sql)
   sql= 'delete FROM tercerbd.atractivopersonal;'
   miCursor.execute(sql)
   sql= 'delete FROM tercerbd.sociedad;'
```

```
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.iteracion;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.iteracionecon;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.nacer ;'
    miCursor.execute(sql)
    sql = 'ALTER TABLE tercerbd.nacer  AUTO_INCREMENT = 1;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.midicphbasi;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.organizaciones;'
    miCursor.execute(sql)
    sql= 'alter table tercerbd.organizaciones AUTO_INCREMENT=1;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.orgafami;'
    miCursor.execute(sql)
    sql= 'alter table tercerbd.orgafami AUTO_INCREMENT=1;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.organiza_miembros;'
    miCursor.execute(sql)
    sql= 'delete FROM tercerbd.organiza_miembrosfami;'
    miCursor.execute(sql)
    miConexion.commit()
    miCursor.close()
    return

def sexoBiol():
    azar = round(random.random(), 2)
    sex = 'V' if azar <0.5 else 'M'
    return sex
```

```
def formaDiccDecisionGeneral(regActi):
   global anyosVidaMax
   diccionario = {}
   reg = regActi
   conta = 0
   VM = anyosVidaMax - 18 #debe tener acceso al valor concreto calculado en ph_principal
   actividadesPosiblesUnidades
=[2,3,5,7,9,10,12,13,14,15,16,17,18,19,20,22,23,24,25,26,27,28,29,30,31,32,33,34] #
actividades
   for a in reg:
      if a[0] in actividadesPosiblesUnidades:
         conta += 1
         clave = a[1].rstrip()
         valor1 = int(a[6])
         valor2 = anyosVidaMax  if a[7] == 'VM' else int(a[7])  #NP
         valor3 = VM if a[8] == 'VM' else int(a[8]) #VM
         diccionario [clave] = [a[0],valor1,valor2,valor3]
   return diccionario

def recuperarTablasitulaboposi():
   #se cargan los 32 primeros registros de la tabla actividades
   sqlselect= "Select * from tercerbd.situlaboposi "
   regActi = ph_herramientas.selectBDNoWhere(sqlselect)
   regis = [list(a) for a in regActi]
   return  regis


def proxIdOrganizacion(familia):
   familia = familia
   #return el próximo id o dniFiscal de la Organizacion a crear
   miCursor=miConexion.cursor()
```

```
    if familia == True:
        miCursor.execute("SELECT `AUTO_INCREMENT` FROM
INFORMATION_SCHEMA.TABLES \
                WHERE TABLE_SCHEMA = 'tercerbd' AND   TABLE_NAME   = 'orgafami'; ")
    else:
        miCursor.execute("SELECT `AUTO_INCREMENT` FROM
INFORMATION_SCHEMA.TABLES \
                WHERE TABLE_SCHEMA = 'tercerbd' AND   TABLE_NAME   = 'organizaciones'; ")
    idOrg=miCursor.fetchall()
    idOrganizaciones=idOrg[0][0]
    miConexion.commit()
    miCursor.close()
    return idOrganizaciones




def formaEmpresa():
    #forma empresas de estos 10 tipos incluidos en la lista
    tipo = ['industria','servicios','campo','enseñanza','sanidad',\
        'software','seguridad','investigacion','finanzas','administración']
    azar = random.randint(0,9)
    tipo_empresa = tipo[azar]
    unidadxAnyo = 100
    precioUnidad = 13
    persTrab =15
    sql="INSERT INTO tercerbd.organizaciones
(tipo,creacion,objetivo,unidadxAnyo,precioUnidad,persTrab) VALUES
(%s,%s,%s,%s,%s,%s)"
    valor=(tipo_empresa,tiempo,'Producción',unidadxAnyo,precioUnidad,persTrab)
    ph_herramientas.insertUno(sql,valor)
    return
```

```
def atractivoPersonal():
   #se calcula el INDICE de cada Actividad para cada ph y se graba en Mysql. No cambia.
   #en el diccionario miDicPh, campo  va3, formará parte del indiceExito de cada Actividad de cada pH.
   listaTerna = []
   dicGraba={1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,16:0, 17:0,18:0,\
         19:0,20:0,21:0,22:0,23:0,24:0, 25:0, 26:0,27:0,28:0, 29:0,30:0,31:0,32:0,33:0,34:0}
   #tendras que seleccionar el DNI a los que no tengan calculado al atractivoPersonal
   for dni in nombrePh:
      objeGene = nombrePh.get(dni)
      if objeGene.atraPers == 'N':
         objeGene.atraPers = 'S'

listaTerna.append((dni,objeGene.miDicPhBasi['arquetipo1'],objeGene.miDicPhBasi['caracter1']))


   for lista in listaTerna: #calculo
      matriz = []
      dni=lista[0]   #inicializacion
      arquetipo=lista[1]
      caracter=lista[2]
      for i in [1,4,6,9,11]:
         azar=random.randint(6,10)
         dicGraba[i]=azar   #comer,descansar,salud,vestido, preparar oficio...
      #se supone que las actividades de arquetipo, caracter, obligatorias, no asignadas son conjuntos disjuntos...
      #diccioArquet NO es modificado en esta funcion: solamente se LEE
      for arquetipo in diccioArquet:
         listaA = []
         listaA = diccioArquet[arquetipo] #quiero que meta los valores numericos
         for ii in listaA:
```

```
            azar=random.randint(5,9)
            dicGraba[ii]=azar
        for caracter in diccioCaract:
          listaC=[]
          listaC=diccioCaract[caracter]
          for ii in listaC:
            azar=random.randint(4,8)
            dicGraba[ii]=azar
        for iii in listaNoAsignadas:
          listaNA=[]
          listaNA=listaNoAsignadas
          for iii in listaNA:
            if iii in [3,5,10,21]:
              azar=random.randint(6,9)
              dicGraba[iii]=azar
            else:
              azar=random.randint(1,7)
              dicGraba[iii]=azar
        #graba en "atractivopersonal". En miDicPh() no graba, lo hará a través de Indice de
Exito de Actividad
        for i in range(1,35):
          valor3=(dni,i,dicGraba.get(i))
          matriz.insert(dni,tuple(valor3))
        sql3='INSERT INTO tercerbd.atractivopersonal (dni,actividad,atractivo) \
            VALUES (%s,%s, %s)'
        ph_herramientas.insertBD(sql3,matriz) #graba en tabla atractiviPersonal tipo
executemany
    return


def asignaArquetipo(i):
    dicOrdenado = {}
    miDicPhBasi = i
```

```
    # lo llama  gestarPI y gestar. Se asigna por  azar en PI
    dicOrdenado = sorted(miDicPhBasi.items(), key= lambda i:i[1],reverse=True)
    miDicPhBasi['arquetipo1'] = dicOrdenado[0][0]
    miDicPhBasi['arquetipo2'] = dicOrdenado[1][0]
    return

def asignaCaracter(i):
    #lo llama  gestarIP y gestar
    #el azar determina el caracter
    dentro=True
    miDicPhBasi = i
    caract=['amorfo','colerico' ,'apasionado', 'flematico' ,'sanguineo' ,'sentimental','apatico' ]
    while dentro:
        for i in range (0,2):
            azar1=random.randint(0,3)
            azar2=random.randint(4,6)
        if azar1 != azar2:
            dentro=False
    miDicPhBasi['caracter1'] = caract[azar1]
    miDicPhBasi['caracter2'] = caract[azar2]
    return



def riesgoTolerancia(dni):
    # Puede tenerse aversión al riesgo o puede irse a buscarlo
    # según el arquetipo: se asigna una tolerancia al riesgo
    # la maxima tolerancia es 5, la minima es 0 (aversión al riesgo)
    # se graba en nacer; por gestarPInicial y Gestar.- En OBJETO en nacer_ph
    # se utiliza en la asignación de trabajo y en S-Dificil
    dni = dni
    miAC = []
```

```
    objeGene1=nombrePh.get(dni)
    miAC = [objeGene1.miDicPhBasi['arquetipo1'],objeGene1.miDicPhBasi['arquetipo2']]

    if 'heroe'  in miAC:
        riesgoToler = 5
    elif 'gobernante'in miAC:
        riesgoToler = 5
    elif 'mago'  in miAC:
        riesgoToler = random.randint(3, 4)
    elif  'explorador' in miAC:
        riesgoToler = random.randint(3, 4)
    elif  'rebelde' in miAC:
        riesgoToler = random.randint(3, 4)
    else:
        riesgoToler = random.randint(0,3)

    return riesgoToler

def potencialCiResi(ci,re):
    potencial = '0-Ba-Me' #Bajo o casi Medio
    coefCI = ci
    coefResi = re
    sumaCoef = coefCI + coefResi
    if 1.39 < sumaCoef < 1.69:
        potencial = '1-Medio'
    elif 1.69 < sumaCoef < 1.9:
        potencial = '2-Altos'
    elif 1.9 < sumaCoef < 2.01:
        potencial = '3-Malto'
    return potencial
```

```
#FIN CAJA DE HERRAMIENTAS. He comprobado que todas son llamadas 14-7-2024



def cargarActiDat1245(objeGene,numActi,regActi):
    #DAR DE ALTA UNA ACTIVIDAD -numActi- EN  miDicPh() del ph(objDni)
    #objeGene=nombrePh.get(objDni)
    i = numActi
    #actvidades que tienen 0 en  coste de tiempo
    regActi[i-1][5] =  random.randint(8,25) if regActi[i-1][5] == 0 else regActi[i-1][5]
    objeGene.miDicPh[i] = ['X',0,0,0,0,0,0,0,0,0]
    objeGene.miDicPh[i][0]=regActi[i-1][1]  #nombre Actividad
    objeGene.miDicPh[i][1]=regActi[i-1][4]  #impacto economico
    objeGene.miDicPh[i][4]=regActi[i-1][5]  #coste tiempo/ lo que le va a durar
    objeGene.miDicPh[i][7]=regActi[i-1][2]  #riesgo actividad


    return



def calderaDesastres(pobl,itera):
    #Está implementado pandemias y crisis económicas.
    #tienen implicaciones sobre muerte y sobre trabajo
    def pandemia():
        tipo='P'
        denomPand=[200,100,70,50,25,15,10,5,2] # va de CERO a OCHO
        #personasAfectadas-->pA
        azar1=random.randint(5,7) #5 y 7 incluidos
        azar2=random.randint(3,5)
        azar3=random.randint(3,8)
```

```
    denom1=denomPand[azar1]
    denom2=denomPand[azar2]
    denom3=denomPand[azar3]
    pAMuerte=int(poblacion/denom1)
    pATrabajo=int(poblacion/denom2)
    pAPalp=int(poblacion/denom3)
    #anota desastre en tabla desastres
    sql1= "INSERT INTO desastres (iteraGlobal,tipo,muertos,paro,palp) \
        VALUES (%s,%s, %s,%s,%s)"
    valor1=(iteraGlobal,tipo,pAMuerte,pATrabajo,pAPalp)
    ph_herramientas.insertUno(sql1,valor1)
    #ejecuta muertes.-
    sqlselect1="select id,edad from nacer where fallecido='N'"
    registro=ph_herramientas.selectBDNoWhere(sqlselect1)  #como siempre: lista de
tuplas
    long=len(registro)
    if pAMuerte >= long:
      pAMuerte=int(long+1) # +1 preparando para el for y entero no float
    else:
      pAMuerte =int(pAMuerte+1) # +1 preparando para el for y entero
    for i in registro[0:pAMuerte]:
      idNacer=0
      puntoHumano=0
      idNacer=i[0] #id o dni.- idNacer es entero
      edad=i[1]
      puntoHumano=idNacer #idNacer es int
      dni=(idNacer,)
      sqlupdate2= "UPDATE tercerbd.nacer set fallecido='S' where id=%s"
      ph_herramientas.updateUno(sqlupdate2,dni)  #al morir se da de baja definitiva de
toda organización
      sqlupdate2= "UPDATE tercerbd.organiza_miembros set esBaja='S' where
id_nacer=%s"
```

```
        ph_herramientas.updateUno(sqlupdate2,dni)
        sqlupdate2= "UPDATE tercerbd.organiza_miembrosfami set esBaja='S' where
id_nacer=%s"
       #ph_herramientas.updateUno(sqlupdate2,dni)
       # sqlupdate2= "UPDATE tercerbd.iteracionecon set va2='S' where dni=%s and
tiempo=%s and actividad=33"
        #valor2=(idNacer,edad)
        #ph_herramientas.updateBD(sqlupdate2,valor2)
        objeto=nombrePh.get(puntoHumano) #En objeto está el  objeto
        if objeto is not None:
          #quitar referencia a Pareja
          pareja = objeto.dniPareja
          if pareja != 0:
            objePareja= nombrePh.get(pareja)
            if objePareja is not None:
              objePareja.dniPareja = 0
          del objeto #para BORRAR el objeto  que ha fallecido

        #ER_AD se encargará luego de completar el proceso, grabar...etc
      #ejecuta perdida de trabajo y reducción del ingresoAnual
      sqlselect1="select id from tercerbd.nacer where fallecido='N'"
      registro=ph_herramientas.selectBDNoWhere(sqlselect1)  #como siempre: lista de
tuplas
      long=len(registro)
      if pATrabajo >= long:
        pATrabajo=int(long+1) # +1 preparando para el for y entero no float
      else:
        pATrabajo =int(pATrabajo+1) # +1 preparando para el for y entero
      for i in registro[0:pATrabajo]:
        idNacer=0
        puntoHumano=0
        idNacer=i[0] #id o dni.- idNacer es entero
```

```
        puntoHumano=idNacer #idNacer es int
        objeto1=nombrePh.get(puntoHumano) #En objeto1 está el  objeto
        if objeto1 is not None:
          if objeto1.situlaboposi >= 4:
            objeto1.situalaboposi = 4
            objeto1.ingrxTrab = 0
          else:
            pass
        #ER_AD se encargará luego de completar el proceso, grabar...etc
      return pAMuerte


    def crisis():
      tipo='E'
      denomCrisis=[200,100,70,50,25,15,10,5,2]
      #personasAfectadas-->pA
      azar2=random.randint(3,5)
      azar3=random.randint(3,8) #3 y 8 incluidos
      denom2=denomCrisis[azar2]
      denom3=denomCrisis[azar3]
      pATrabajo=round(poblacion/denom2,0)
      pAPalp=round(poblacion/denom3,0)
      #anota desastre en tabla desastres
      sql1= "INSERT INTO desastres (iteraGlobal,tipo,muertos,paro,palp) \
            VALUES (%s,%s, %s,%s,%s)"
      valor1=(iteraGlobal,tipo,0,pATrabajo,pAPalp)
      ph_herramientas.insertUno(sql1,valor1)
      #ejecuta perdida de trabajo y reducción del ingresoAnual
      sqlselect1="select id from nacer where fallecido='N'"
      registro=ph_herramientas.selectBDNoWhere(sqlselect1)  #como siempre: lista de
tuplas
      long=len(registro)
```

```
    if pATrabajo >= long:
      pATrabajo=int(long+1) # +1 preparando para el for y entero no float
    else :
      pATrabajo =int(pATrabajo+1) # +1 preparando para el for y entero
    for i in registro[0:pATrabajo]:
      idNacer=0
      puntoHumano=0
      idNacer=i[0] #id o dni.- idNacer es entero
      puntoHumano=idNacer #idNacer es int
      objeto1=nombrePh.get(puntoHumano) #En objeto1 está el  objeto
      if objeto1 is not None:
        if objeto1.situlaboposi >= 4:
          objeto1.situalaboposi = 4
          objeto1.ingrxTrab = 0
        else:
          pass
      #ER_AD se encargará luego de completar el proceso, grabar...etc
    return


  poblacion=pobl
  iteraGlobal= tiempo #es el tiempo Global, diferente de la edad de cada persona
  pAMuerte = 0
  #ruleta de la Fortuna
  ruleta=random.randint(1,10)
  if ruleta<=7:
    tipo='N'
    sql1= "INSERT INTO desastres (iteraGlobal,tipo,muertos,paro,palp) \
          VALUES (%s,%s, %s,%s,%s)"
    valor1=(iteraGlobal,tipo,0,0,0)
    ph_herramientas.insertUno(sql1,valor1)
  elif ruleta >7 and ruleta <=9: #falta modelizar el largo de Kondratiev
```

```
      crisis()
   elif ruleta==10:
      pAMuerte = pandemia()
   return pAMuerte

#FIN REGLAS SOCIEDAD

#POBLACIÓN INICIAL Y SIGUIENTES GENERACIONES: GESTACION Y NACIMIENTO

def azarCI():
   dentro=True
   while dentro:
      azarCI = round(random.random(), 2)
      if azarCI > 0.3:
         dentro = False
   return azarCI

def azarResi():
   dentro=True
   while dentro:
      azarResi = round(random.random(), 2)
      if azarResi > 0.3:
         dentro = False
   return azarResi

def nacerPI(poblacionInicial,regActi):
   # Población Inicial (PI).
   #Como proceden de Creación (no de evolución) tienen que estar YA en edad de procrear
y trabajar
   # Crea los ph(), graba en nacer, miDicPhBasi, iteracion (miDicPh)
   def edadPI():
```

```
    edadPI = random.randint(18,40)
    return edadPI

  def valoresmiDicPhBasiPI(a):
    relacion = ['arquetipo1','arquetipo2', 'caracter1','caracter2']
    aa = a
    objeGene = nombrePh.get(aa)
    for i in miDicPhBasi: # rellena  con random, que serán arquetipos y caracteres
      objeGene.miDicPhBasi[i]  = round(random.random(), 4) if i not in relacion else 0
    asignaArquetipo(objeGene.miDicPhBasi)
    asignaCaracter(objeGene.miDicPhBasi)
    return

  #MAIN nacerPI
  poblacionInicial = poblacionInicial
  regActi = regActi
  lista = []
  listaMDPB=[]
  matriz1 = []
  matriz2 = []
  PlusP = 11400/10  #Deberia existir un limite maximo productividad: el 10% del
ingrxTrabB



  #al ser poblacion inicial, el primer dni=1 y el ultimo dni=poblacionInicial
  matriz2 = []
  for i in range(0, 17): #16 campos; warning:(range(a,b))es hasta b-1.
    listaMDPB.append(0)
  for i in range(0, 10): # warning:(range(a,b))es hasta b-1.
    lista.append(0)
  for a in range(1, poblacionInicial+1):
```

```
    #nacen los ph() de la Poblacion Inicial
    rbuCapita = 11400
    dni = a
    #crea objeto hijo/a
    nombrePh[dni] = ph(dni)
    #actuo sobre el objeto creado
    objeGene=nombrePh.get(dni)  #objeGene apunta al objeto hijo



    objeGene.dni = dni
    objeGene.edad = edadPI()  # la edad del ser que se hace por random en este caso para procrear
    objeGene.inicioAbsoluto = 0  #tiempo Global
    objeGene.sexo = sexoBiol()
    objeGene.intencionalidad = 'No plantea'
    objeGene.idFiscalFamilia = 0
    objeGene.dniPareja = 0
    objeGene.idFiscarEmpresa = 0
    objeGene.riesgoTolerancia = 5    # DEBE rellenarse cuando sepas arquetipos con riesgoToler
    objeGene.idPadre = -1
    objeGene.idMadre = -1
    objeGene.coefiInte = azarCI()
    objeGene.resiliencia = azarResi()
    objeGene.potencial = potencialCiResi(objeGene.coefiInte,objeGene.resiliencia)
    objeGene.atraPers = 'N'
    objeGene.sumaVa3Año=0.0
    objeGene.situlaboposi=1
    objeGene.palp = 0
    objeGene.gastoAnyo = 0
    objeGene.plusProductividad =  PlusP + (PlusP/random.randint(8,10)) - (PlusP/random.randint(5,10))
```

```
    objeGene.ingrxTrab = 0
    objeGene.ingrXRBU = rbuCapita
    objeGene.miDicPhBasi = miDicPhBasi
    #resumen economico
    valoresmiDicPhBasiPI(a)  # Asigna Arquetipo y Caracter
    listaMDPB[0] = a
    indiceLista = 1
    for i in miDicPhBasi:
      listaMDPB[indiceLista] = objeGene.miDicPhBasi[i]
      indiceLista +=1
    matriz2.insert(a,tuple(listaMDPB))
    #AHORA puede calcularse el riesToles !!!
    riesToler = riesgoTolerancia(dni)
    objeGene.riesgoTolerancia = riesToler

    listaMDPB =  []
    for i in range(0, 17): #16 campos; warning:(range(a,b))es hasta b-1.
      listaMDPB.append(0)

    lista[0] = objeGene.edad  # la edad del ser que se HIZO por random en este caso para
procrear
    lista[1] = 0  # es nacerPI
    lista[2] = objeGene.sexo
    lista[3] = 'N' #fallecido NO
    lista[4] = objeGene.riesgoTolerancia #Tolerancia  al riesgo
    lista[5] = -1 #idPadre
    lista[6] = -1 #idMadre
    lista[7] = objeGene.coefiInte  #coeficiente intelectual
    lista[8] = objeGene.resiliencia #resiliencia
    lista[9] = objeGene.potencial
    matriz1.insert(a,tuple(lista))
```

```
    lista = []
    for i in range(0, 10): #warning:(range(a,b))es hasta b-1.
      lista.append(0)

  sql2 = "INSERT INTO tercerbd.midicphbasi (dni,amigo,cuidador,explorador ,heroe ,
amante ,\
          inocente ,rebelde ,sabio,gobernante,creador ,bufon,mago, \
          arquetipo1,arquetipo2, caracter1,caracter2 ) \
          VALUES (%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s, %s,%s,%s,%s,%s,%s)"
  tupla2 = matriz2
  ph_herramientas.insertBD (sql2,tupla2)
  #se graba en la tabla nacer

  tupla1 = matriz1
  sql1 = "INSERT INTO tercerbd.nacer (edad,inicioAbsoluto,sexo,fallecido,
riesgoTolerancia,\
    idPadre,idMadre,coefiInte, resiliencia, potencial) VALUES (%s,%s,%s,
%s,%s,%s,%s,%s,%s,%s, %s)"
  dni = ph_herramientas.insertBD(sql1, tupla1)  #dni del ph() que se acaba de grabar en
ultimo lugar

  for a in range(1, poblacionInicial+1):
    matriz4 = []
    matriz5 = []
    objeGene = nombrePh.get(a)
    #actividades base
    #¿Cuanto importan las necesidades básicas de cada ph() de la PI este año?. Todos
iguales
    relaActi=[1,2,3,4,6,7,8,9,10,11] #relacion de actividades que se les activan;
    #PI todos pueden trabajar, tener hijos,  estudiar... pues su edad ha sido elevada.
    #Sus hijos: normal :relaActi=[1,4,6,7,10,11]
    for i in relaActi: #se escribe en objeGene.miDicPh
      cargarActiDat1245(objeGene,i,regActi)
```

```
    #grabar en tabla iteración el contenido de miDicPh ; para el dni en curso
  for i in objeGene.miDicPh.keys():
    if i == 3:
      objeGene.miDicPh[i][5] = 9  #quieren tener 9 hijos; si no es PI, es x azar en decision
    iterGraba = []
    tupla4 = ()

    for ii in range (0,14): #inicializa iterGraba
      iterGraba.append(0)

    iterGraba[0] = a
    iterGraba[1] = tiempo #tiempo Global: vueltas hasta duracionSimulacion
    iterGraba[2] = lista[0] #edad
    iterGraba[3] = i
    iterGraba[4] = objeGene.miDicPh[i][0] #va0
    iterGraba[5] = objeGene.miDicPh[i][1] #va1
    iterGraba[6] = objeGene.miDicPh[i][2] #va2
    iterGraba[7] = objeGene.miDicPh[i][3] #va3
    iterGraba[8] = objeGene.miDicPh[i][4] #va4
    iterGraba[9] = objeGene.miDicPh[i][5] #va5
    iterGraba[10] = objeGene.miDicPh[i][6] #va6
    iterGraba[11] = objeGene.miDicPh[i][7] #va7
    iterGraba[12] = objeGene.miDicPh[i][8] #va8
    iterGraba[13] = objeGene.miDicPh[i][9] #va9
    matriz4.insert(a,tuple(iterGraba))

  sql = "INSERT INTO tercerbd.iteracion (dni,tiempo, edad, actividad ,\
      va0 , va1,va2, va3, va4, va5, va6, va7, va8, va9 ) \
      VALUES (%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s, %s,%s,%s)"
  tupla4 = matriz4
  ph_herramientas.insertBD (sql,tupla4)
```

```
    lista2 = []
    for i in range (0,18):
      lista2.append(0)
    lista2[0] = a #dni
    lista2[1] = tiempo #iteracion
    lista2[2] = objeGene.edad   #edad
    lista2[3] = 4 #situacion laboral
    lista2[4] = 'S' #¿es la ultima?
    lista2[5] = 0 #suma va3 - no calculada aún -PI-
    lista2[6] = 0 #♠suma de va9, no calculado aún
    lista2[7] = 0 #palp
    lista2[8] = rbuCapita #gasto este primer anyo
    lista2[9] = rbuCapita #ingrxRBU .
    lista2[10] = 0 #ingrx Trabajo báse
    lista2[11] = 0 #plus productividad
    lista2[12] = 0 #.- ingrxTrab
    lista2[13] = lista2[12] + lista2[9]  #ingrTotal, estamos en nacerPI,
    lista2[14] = objeGene.intencionalidad
    lista2[15] = 0 #idFiscal Familia
    lista2[16] = 0 #idPareja
    lista2[17] = 0 #idFiscalEmpresa
    matriz5.insert(a,tuple(lista2))

    sql = "INSERT INTO tercerbd.iteracionecon (dni,tiempo,
edad,situlabo,ultima,sumava3,sumava9,palp,\

gastoanyo,ingrxRBU,ingrxTrabB,ingrxPlusP,ingrxTrab,ingrTotal,intenciona,idFiscalFamilia,i
dPareja,idFiscalEmpresa) \
      VALUES (%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s, %s,%s,%s,%s,%s,%s,%s)"
    ph_herramientas.insertBD(sql,matriz5)
```

```
    return



def muerte(tiempo):
    tiempo = tiempo
    #establezce fallecido='S' para señalar registros en nacer e iteracion y POO
    #en "nacer" y en POO, cada id SOLO tiene un registro; NO así en iteracion
    def teHaTocado():
        edad = objeGene.edad
        if edad>edadMax and random.randint(1,100)>30:
            fallecido='S'
        elif 1 <  edad <=18 and random.randint(1,100)>98: #niños  pueden morir...dificil
            fallecido='S'
        elif 18 < edad < edadMax and random.randint(1,100)>95: #adultps pueden
morir...dificil
            fallecido='S'
        else:
            fallecido='N'
        return fallecido


    #Todos los vivos cumplen años y pueden MORIR
    edadMax = anyosVidaMax # la esperanza de vida aumenta con el t
    numMuertos = 0
    listaBajas = []
    nombrePhClavesCongeladas = []
    nombrePhClavesCongeladas = list(nombrePh)
    for i in nombrePhClavesCongeladas:
        objeGene = nombrePh.get(i)
        dni = i
        muerto = teHaTocado()
        #si ha fallecido hay que actualizar nacer, iteracion y la POO
```

```
    if muerto == 'S':
      valor = (dni,)
      sqlupdate2= "UPDATE nacer set fallecido='S' where id=%s"
      ph_herramientas.updateUno(sqlupdate2,valor)
      sqlupdate2= "UPDATE organiza_miembros set esBaja='S' where id_nacer=%s"
      ph_herramientas.updateUno(sqlupdate2,valor)
      sqlupdate2= "UPDATE organiza_miembrosfami set esBaja='S' where id_nacer=%s"
      ph_herramientas.updateUno(sqlupdate2,valor)
      # Todas las iteracionecon del fallecido pondrán en ultima 'N' pues está fallecido,
      #no hay que hacer cálculos actuales con él
      valorIte = (dni,tiempo)
      sqlupdate2= "UPDATE tercerbd.iteracionecon set ultima='N' where dni=%s and
tiempo=%s"
      ph_herramientas.updateUno(sqlupdate2,valorIte)
      listaBajas.append(dni)
      numMuertos +=1
  for baja in listaBajas:
    #debemos comprobar que no constan como pareja de alguien vivo. Si constaran:
dniPareja=0
    objeGeneB = nombrePh[baja]
    dniPare = objeGeneB.dniPareja
    objeGenePare = nombrePh.get(dniPare)
    if objeGenePare !=0 and objeGenePare is not None:
      objeGenePare.dniPareja = 0
    else:
      pass
    #ahora podemos quitar a la Baja de la lista
    del nombrePh[baja]

  return numMuertos

def reproducirse():
```

```
   #se buscan V y M que tienen S en hijos, se escoge uno con una
   #se toman cinco genes de uno y cinco de otra y se forma su genoma y se asigna un
   #sexo por azar. Se graba en la BD; despues de gestar,  hay que nacer
   #al mundo de los objetos: se llamará nacer_ph
   #Cuando tiempo=1 se trata de la Poblacion Inicial que NO ha pasado por ER-ED
   #En PI se les asignó a todos la edad para tener y en la actividad 3 se puso-->Hijos='S'
   def formaFamilia(idOrganizacion,ingr,va17,va18,idDesc):
      #Crea Familia o completa otra ya formada con nuevos miembros
      def org_mie():
         sql="INSERT INTO tercerbd.organiza_miembrosfami (idFiscal,id_nacer,tipo) VALUES
(%s,%s,%s)"
         valor=(idOrganizacion,vv,tipo)
         ph_herramientas.insertUno(sql,valor)
         return
      def busca_idFami(vaXX):
         sql="select idFiscal from tercerbd.organiza_miembrosfami where id_nacer=%s and
tipo='Familia' and esBaja = 'N'"
         valor=(vaXX,) #padre
         idFiscal_Familia = ph_herramientas.selectBD(sql,valor)
         return idFiscal_Familia
      objeGeneP=nombrePh.get(va17) #padre
      objeGeneM=nombrePh.get(va18) #madre
      tipo='Familia'
      idDesc = idDesc[0] if idDesc is tuple else idDesc  #evitar que dni aparezca como tupla
      if objeGeneP.idFiscalFamilia == 0 and objeGeneM.idFiscalFamilia ==0:
         #Ni padre ni madre estan en una familia como progenitores
         vv=0
         creacion = tiempo
         sql="INSERT INTO tercerbd.orgafami (creacion,tipo,ingresos) VALUES (%s,%s,%s)"
         valor=(creacion,tipo,ingr)
         ph_herramientas.insertUno(sql,valor)
         vv=va17 #padre
```

```
        org_mie()
        vv=va18 #madre
        org_mie()
        vv=idDesc #hijo
        org_mie()
        #es un diccionario:key es idOrganizacion y Value el puntero al objeto
        nombreOrg[idOrganizacion]=Organizacion(idOrganizacion)
        nombreOrg[idOrganizacion].tipo='Familia'
        nombreOrg[idOrganizacion].objetivo='Procrear'
        #nombreOrg[idOrganizacion].ingresos=ingr

        objeGeneP.idFiscalFamilia=idOrganizacion
        objeGeneP.dniPareja = va18
        objeGeneM.idFiscalFamilia=idOrganizacion
        objeGeneM.dniPareja = va17
    elif objeGeneP.idFiscalFamilia != 0 and objeGeneM.idFiscalFamilia !=0:
        #No debe formar familia, pero debe añadir el hijo a la existente, supongo estan en la
misma
        idFiscal_Familia = busca_idFami(va17)
        if idFiscal_Familia != []:
            idOrganizacion = idFiscal_Familia[0][0]
            vv = idDesc
            org_mie()
    elif objeGeneP.idFiscalFamilia == 0 and objeGeneM.idFiscalFamilia !=0:
        #No debe formar familia, pero debe añadir al hijo y al padre a la existente de la
madre
        idFiscal_Familia = busca_idFami(va18)
        if idFiscal_Familia != []:
            idOrganizacion = idFiscal_Familia[0][0]
            vv = va17
            org_mie()
            objeGeneP.idFiscalFamilia=idOrganizacion
```

```
            objeGeneP.dniPareja = va18
            vv = idDesc
            org_mie()
        elif objeGeneP.idFiscalFamilia != 0 and objeGeneM.idFiscalFamilia ==0:
          #No debe formar familia, pero debe añadir al hijo y a la madre a la existente del
padre
            idFiscal_Familia = busca_idFami(va17)
            if idFiscal_Familia != []:
              idOrganizacion = idFiscal_Familia[0][0]
              vv = va18
              org_mie()
              objeGeneM.idFiscalFamilia=idOrganizacion
              objeGeneM.dniPareja = va17
              vv = idDesc
              org_mie()
        return


    def reproxLista(listaZip,dni):
        listaZipIn = listaZip
        dniInicial=dni #dobo saber donde comienzo
        dni=dni #este irá variando
        lista=[]


        matriz1 = []
        matriz2 = []
        matriz3 = []


        for tupla in listaZipIn:
            lista = []
            listaMDPB = []
            for i in range(0, 17): #16 campos; warning:(range(a,b))es hasta b-1.
```

```
            listaMDPB.append(0)
        #crea objeto hijo/a
        nombrePh[dni] = ph(dni)
        #actuo sobre el objeto creado
        objeGene=nombrePh.get(dni)  #objeGene apunta al objeto hijo


        #cada elemento de listaZipIn viene como (dni1,dni2) una tupla que llamo tupla (en
el for)
        if nombrePh.get(tupla[0]) is None or nombrePh.get(tupla[1]) is None:
            continue
        objeGeneP = nombrePh.get(tupla[0])
        objeGeneM = nombrePh.get(tupla[1])
        v19 = tupla[0] #dni de V
        v20 = tupla[1] #dni de M
        primero = sexoBiol() #sexo bio que tenga el nuevo ser


        relacion1 = ['rebelde','sabio','gobernante','creador' ,'bufon','mago',\
                'arquetipo1','arquetipo2', 'caracter1','caracter2']
        objeGene.miDicPhBasi = objeGeneM.miDicPhBasi #todos los random de la madre
        for i in objeGene.miDicPhBasi:
            if i not in relacion1  : # rellena  con random,los seis primeros
                objeGene.miDicPhBasi[i] = objeGeneP.miDicPhBasi[i]


        relacion2 = ['arquetipo1','arquetipo2', 'caracter1','caracter2']
        for i in objeGene.miDicPhBasi:
            if i  in relacion2:
                objeGene.miDicPhBasi[i] = 0


        asignaArquetipo(objeGene.miDicPhBasi)
        asignaCaracter(objeGene.miDicPhBasi)
        listaMDPB[0] = dni
```

```
        indiceLista = 1
        for i in miDicPhBasi:
          listaMDPB[indiceLista] = objeGene.miDicPhBasi[i]
          indiceLista +=1
        matriz2.insert(dni,tuple(listaMDPB))

        #Crea Familia procrear (true); tb graba en organiza_miembros
        ingr = objeGeneP.ingrxRBU + objeGeneM.ingrxRBU
        familia = True if objeGeneP.idFiscalFamilia ==0 and objeGeneM.idFiscalFamilia ==0
else False
        #if familia:
          #idOrganizacion=proxIdOrganizacion(familia)
          #formaFamilia(idOrganizacion,ingr,v19,v20,dni)
        idOrganizacion=proxIdOrganizacion(familia)
        formaFamilia(idOrganizacion,ingr,v19,v20,dni)
        riesgoToler = riesgoTolerancia(dni)  #ya tiene arquetipo1 y arquetipo2

        #se rellena el objeto en RAM con sus valores
        objeGene.dni = dni
        objeGene.edad = 1  # la edad del ser que se gesta ahora
        objeGene.inicioAbsoluto = tiempo  #global: num vuelta actual cuando nace: la única
vez que se escribe aqui
        objeGene.iteracion = tiempo #debe cambiar con cada iteracion
        objeGene.sexo = primero #sexo biologico
        objeGene.intencionalidad = 'Reforzar'
        objeGene.idFiscalFamilia = idOrganizacion
        objeGene.dniPareja = 0
        objeGene.idFiscarEmpresa = 0
        objeGene.riesgoTolerancia = riesgoToler #riesgoToler riesgoTolerancia
        objeGene.idPadre = v19 #idPadre
        objeGene.idMadre = v20
        objeGene.coefiInte = azarCI()
```

```
objeGene.resiliencia = azarResi()
objeGene.potencial = potencialCiResi(objeGene.coefiInte,objeGene.resiliencia)
objeGene.atraPers = 'N'
objeGene.situlaboposi=1
objeGene.palp = 0
objeGene.gastoAnyo = 0
objeGene.plusProductividad = 0
objeGene.ingrxTrab = 0
objeGene.ingrXRBU = rbuCapita
objeGene.ingrTotal = objeGene.ingrxTrab + rbuCapita
#objeGene.miDicPhBasi = miDicPhBasi
#objeGene.miDicPh = {1:['X',0,0,0,0,0,0,0,0,0]}  -lo debe crear automáticamente

#Para tabla nacer
for i in range(0, 10): # warning:range(a,b) es hasta b-1.
   lista.append(0)
#Tabla nacer, es un recien nacido
lista[0] = 1  # la edad del ser que se gesta ahora
lista[1] = tiempo  #global: num vuelta actual
lista[2] = primero #sexo biologico
lista[3] = 'N' # Va para tabla nacer es fallecido;
lista[4] = riesgoToler #riesgoTolerancia
lista[5] = v19 #idPadre
lista[6] = v20 #idMadre
lista[7] = objeGene.coefiInte #coeficiente intelectual
lista[8] = objeGene.resiliencia  #resiliencia
lista[9] = objeGene.potencial
matriz1.insert(dni,tuple(lista))
lista = []

#actividades base: miDicPh
```

```
        #¿Cuanto importan las necesidades básicas de cada ph() de la PI este año?. Todos
iguales
        relaActi=[1,4,6,7,8,10,11] #relacion de actividades que se activan;
        for i in relaActi: #se escribe en objeGene.miDicPh
          cargarActiDat1245(objeGene,i,regActi)
        #grabar en tabla iteración el contenido de miDicPh ; para el dni en curso
        for i in objeGene.miDicPh.keys():
          iterGraba = []
          for ii in range (0,14):
            iterGraba.append(0)
          iterGraba[0] = dni
          iterGraba[1] = tiempo
          iterGraba[2] = 1 #edad: se gesta ahora
          iterGraba[3] = i
          iterGraba[4] = objeGene.miDicPh[i][0] #va0
          iterGraba[5] = objeGene.miDicPh[i][1] #va1
          iterGraba[6] = objeGene.miDicPh[i][2] #va2
          iterGraba[7] = objeGene.miDicPh[i][3] #va3
          iterGraba[8] = objeGene.miDicPh[i][4] #va4
          iterGraba[9] = objeGene.miDicPh[i][5] #va5
          iterGraba[10] = objeGene.miDicPh[i][6] #va6
          iterGraba[11] = objeGene.miDicPh[i][7] #va7
          iterGraba[12] = objeGene.miDicPh[i][8] #va8
          iterGraba[13] = objeGene.miDicPh[i][9] #va9
          matriz3.insert(dni,tuple(iterGraba))
        dni +=1


      tupla1 = matriz1
      sql1 = "INSERT INTO tercerbd.nacer (edad,inicioAbsoluto,sexo,fallecido,
riesgoTolerancia,idPadre,\
        idMadre,coefiInte, resiliencia, potencial) VALUES (%s,%s,%s, %s,%s,%s,%s,%s,%s,
%s)"
```

```
    dniFinal = ph_herramientas.insertBD(sql1, tupla1)  #dni del ph() que se acaba de
grabar en ultimo lugar



    sql = "INSERT INTO tercerbd.midicphbasi (dni,amigo,cuidador,explorador ,heroe ,
amante ,\
                inocente ,rebelde ,sabio,gobernante,creador ,bufon,mago, \
               arquetipo1,arquetipo2, caracter1,caracter2 ) \
               VALUES (%s,%s, %s,%s,%s,%s,%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s,%s,%s)"
    tupla2 = matriz2
    ph_herramientas.insertBD (sql,tupla2)



    sql = "INSERT INTO tercerbd.iteracion (dni,tiempo, edad, actividad ,\
          va0 , va1,va2, va3, va4, va5, va6, va7, va8, va9 ) \
          VALUES (%s,%s, %s,%s,%s,%s,%s,%s, %s,%s,%s,%s, %s,%s,%s)"
    tupla3 =matriz3
    ph_herramientas.insertBD (sql,tupla3)


      #Aún no existe una contabilidad familiar
      #if familia:
        #ingr = objeGeneP.ingrxRBU + objeGeneM.ingrxRBU + objeGene.ingrXRBU
        #objeGeneP.ingrxRBU = objeGeneM.ingrxRBU = objeGene.ingrXRBU = 0
        #nombreOrg[idOrganizacion].ingresos=ingr


    return


  #construir Lista con la info de POO
  listaV = []
  listaM = []
  listaZip = []
  nombrePhClavesCongeladas = []
```

```
  estadosInactivos=['X','N']
        # va[0] -->X: no se está ejecutando / N: no se quiere ejecutar / Texto de actividad: se está ejecutando
  nombrePhClavesCongeladas = list(nombrePh)
  for i in nombrePh: #se eliminan los dni de las parejas en la copia
    objeGene = nombrePh.get(i)
    if objeGene.dniPareja != 0 and objeGene.dniPareja in nombrePhClavesCongeladas:
      nombrePhClavesCongeladas.remove(objeGene.dniPareja)
  #En lista hay sin pareja (V o M) y con pareja de dniPareja (pero la pareja NO está en lista)
  for i in nombrePhClavesCongeladas:
    objeGene = nombrePh.get(i)
    #miDicPh `3 +-->Hijos
    if 18 < objeGene.edad < 65:
      if 3 in objeGene.miDicPh and  objeGene.miDicPh[3][0] not in estadosInactivos :
        if  objeGene.dniPareja == 0 : #No tiene pareja, soltero/a
          if objeGene.sexo =='V':
            listaV.append(objeGene.dni)
          else:
            listaM.append(objeGene.dni)
        elif objeGene.dniPareja != 0 : #SI tiene pareja y Familia formada
          if objeGene.sexo =='V':
            listaV.append(objeGene.dni)
            listaM.append(objeGene.dniPareja)
          else:
            listaM.append(objeGene.dni)
            listaV.append(objeGene.dniPareja)

  for tupla in zip(listaV,listaM):
    listaZip.append(tupla) #si distintas longitudes, zip toma la de menor tamaño
  #sql = "SELECT `AUTO_INCREMENT` FROM  INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'tercerbd' \
```

```
    #AND  TABLE_NAME  = 'nacer';"   He observador no coincidencia entre valor autoincrement
    #de tabla nacer y este de informatio_schema
  sql = "SELECT count(*) FROM tercerbd.nacer;"
  DNI = ph_herramientas.selectBDNoWhere(sql)
  dni = DNI[0][0]
  dni +=1 #será el dni del próximo ser nacido; el resto sera dni+1
  nacidos = len (listaZip)

  reproxLista(listaZip,dni) #Se formaran ALGUNAS familias, se completaran otras, naceran hijos...
  return nacidos
```

#FIN DE POBLACIÓN INICIAL Y DE SIGUIENTES GENERACIONES: GESTACION Y NACIMIENTO

#INICIO MOVIMIENTOS ANUALES, ITERACIONES anuales

```
def iteracionGeneral(plusP,matrizL):

  # Esta función debe añadir una iteraciónde a todos los nacidos no muertos
  #cada año una iteración mas
  #en 'tiempo' viene el valor actual que tiende a duracionSimulacion
  plusP = plusP
  matriz1 = []
  matriz2 = []
  matriz3 = []
  matriz4 = []
  matrizLabo = matrizL
```

```
   for ind in nombrePh:
      edadNew = 0
      objeGene = nombrePh.get(ind)
      dni=objeGene.dni #el id o dni del que va a escribirse una nueva iteracion
      edad = objeGene.edad #la edad anterior
      edadNew = edad + 1   #esta será su edad ahora, para grabarla ahora
      objeGene.edad = edadNew
      objeGene.iteracion = tiempo
      # ERED: modulo central en decisiones ph() y asuntos economicos individuales

ph_ERED.ERealEDeseo(dni,edadNew,nombrePh,salarioMinimo,cobranRBU,rbuCapita,plu
sP,tiempo,pobl,diccDecisionGeneral,matrizLabo,objeGene)


      #podemos hacerla con MANY
      #señalar que la iteracion ya no será la ultima: vamos a grabar la nueva
      tiempoAnterior = tiempo- 1  #el "tiempo" actual es un  año más que la ultima iteracion
grabada
      valorIterUlt=(dni,tiempoAnterior)
      matriz1.insert(dni,valorIterUlt)


      asignaIntencionalidad(dni,edad)   #calcula la SUMA y cambia, si procede,
intencionalidad


      for i in objeGene.miDicPh.keys():
         iterGraba = []
         tupla3 = ()
         for ii in range (0,14):
            iterGraba.append(0)
         iterGraba[0] = dni
         iterGraba[1] = tiempo #tiempo Global: vueltas hasta duracionSimulacion
         iterGraba[2] = edadNew
         iterGraba[3] = i #actividad
```

```
    iterGraba[4] = objeGene.miDicPh[i][0] #va0
    iterGraba[5] = round(objeGene.miDicPh[i][1],2) #va1
    iterGraba[6] = round (objeGene.miDicPh[i][2],2) #va2
    iterGraba[7] = round(objeGene.miDicPh[i][3],2) #va3
    iterGraba[8] = int(objeGene.miDicPh[i][4]) #va4
    iterGraba[9] = int(objeGene.miDicPh[i][5]) #va5
    iterGraba[10] = int(objeGene.miDicPh[i][6]) #va6
    iterGraba[11] = int(objeGene.miDicPh[i][7]) #va7
    iterGraba[12] = objeGene.miDicPh[i][8] #va8
    iterGraba[13] = round(objeGene.miDicPh[i][9],2) #va9
    matriz2.insert(dni,tuple(iterGraba))

#Grabacion datos economicos de la iteracion en tabla iteracionEcon
lista2 = []
for ii in range (0,20): #inicializa lista2
    lista2.append(0)
lista2[0] = dni #dni
lista2[1] = tiempo #iteracion
lista2[2] = objeGene.edad   #edad
lista2[3] = objeGene.situlaboposi #situacion laboral
lista2[4] = objeGene.antiguedad    # antiguedad en esa situlaboposi
lista2[5] = objeGene.climLabo      #como se siente es ese puesto
lista2[6] = 'S' #¿es la ultima?
lista2[7] = objeGene.sumaVa3Año #suma va3
lista2[8] = objeGene.sumaVa9Año
lista2[9] = objeGene.palp #palp
lista2[10] = objeGene.gastoAnyo #gasto anyo
lista2[11] = objeGene.ingrxRBU
lista2[12] = objeGene.ingrxTrabB #ingrx Trabajo báse
lista2[13] = objeGene.plusProductividad #plus productividad
lista2[14] = objeGene.ingrxTrab #suma de los anteriores ingresos
```

```
    lista2[15] = objeGene.ingrTotal if objeGene.ingrTotal != 0 else lista2[11] # ingrTotal =
ingrxTrab +ingrxRBU
    lista2[16] = objeGene.intencionalidad
    lista2[17] = objeGene.idFiscalFamilia
    lista2[18] = objeGene.dniPareja
    lista2[19] = objeGene.idFiscalEmpresa
    matriz3.insert(dni,tuple(lista2))

    lista4 = []
    for ii in range (0,2): #inicializa lista4
      lista4.append(0)
    lista4[0] = objeGene.edad
    lista4[1] = dni
    matriz4.insert(dni,tuple(lista4))

  print('Voy a grabar matriz1 de iteracionecon')
  sqlupdat2="Update iteracionecon Set ultima='N' where dni=%s and tiempo=%s"
  ph_herramientas.updateBD(sqlupdat2,matriz1)
  print('Voy a grabar matriz2 de iteracion General')
  sql = "INSERT INTO iteracion (dni,tiempo, edad, actividad ,\
    va0 , va1,va2, va3, va4, va5, va6, va7, va8, va9 ) \
    VALUES (%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s, %s,%s,%s)"
  ph_herramientas.insertBD (sql,matriz2)
  print('Voy a grabar matriz3 de iteracionecon')
  sql = "INSERT INTO tercerbd.iteracionecon (dni,tiempo, edad,situlabo,antiguedad,
climLabo,ultima,sumava3,sumava9,palp,\

gastoanyo,ingrxRBU,ingrxTrabB,ingrxPlusP,ingrxTrab,ingrTotal,intenciona,idFiscalFamilia,i
dPareja,idFiscalEmpresa) \
      VALUES (%s,%s, %s,%s,%s,%s,%s, %s,%s,%s,%s,
%s,%s,%s,%s,%s,%s,%s,%s,%s)"
  ph_herramientas.insertBD (sql,matriz3)
  print('Voy a grabar matriz4 de iteracion General')
```

```
    sqlupdat4="Update nacer Set edad =%s where id=%s "
    ph_herramientas.updateBD(sqlupdat4,matriz4)

    return

def asignaIntencionalidad(dni,edad):
    #Asigna y graba intencionalidad, en su caso, y GRABA sumaVa3 en ph().
    asignaIntencionalidad.llamadas += 1
    #Intencionalidad y Palp definen el movimiento de un ph()
    #en POO (parte declarativa del ph y en su miDicPh) y en tabla nacer está intencionalidad
    #en la tabla iteración en cada ph, en la fila 33
    #lo llama iteracionGeneral, justo despues de llamar a ER_ED
    def asignaInten(arquetipo,caracter,dni, suma,sumava9,edad):
        objeGene=nombrePh.get(dni)
        potencial = objeGene.potencial
        #habria que variar tabla nacer, nacerPI y reproducirse
        arqOscuFana = ['explorador','heroe','gobernante','rebelde'] #arquetipo
        arqDepresivo = ['inocente','cuidador','creador'] #arquetipo
        ciResi = ['1-Medio','2-Altos','3-Malto']  #todos cinco char
        caraF = ['amorfo','apatico','flematico']  ,#caracter
        caraBR = ['apasionado','sanguineo','sentimental','colerico'] #caracter
        oF=False
        depre=False
        ciRe = False
        inten = ''
        #debo afinar más
        if arquetipo in arqOscuFana:
            oF=True
        if arquetipo in arqDepresivo:
            depre=True
        if (potencial in ciResi) or (caracter in caraBR):
```

```
        ciRe = True
    #la intencionalidad cambia de vez en cuando
    if edad in [17,25,30,45,55,65]:
      if (suma <= 1 or sumava9 < 3) and oF:
        inten='Acabemos'
      elif 2 <= suma < 20 or 4<= sumava9< 60 or depre:
        inten='Busqueda'
      elif suma >= 20 or ciRe or sumava9 >=60  or oF:
        inten='Reforzar'
    elif edad >= 65 and (suma >= 20 or sumava9 >= 60):
      inten = 'Busqueda'
    else:
      #inten = objeGene.intencionalidad
      inten = 'Flotar' if (random.randint (1,11) < 6) or (caracter in caraF) else 'Busqueda'

    #Intencionaliodad está en TRES lugares: Ph(), iteracionecon, nacer

     #guardar en iteración, va9=intencionalidad y va3--> suma de los indices de EXITO de
sus actividades

    sql2='UPDATE tercerbd.iteracionecon SET sumava3=%s,sumava9=%s,intenciona=%s
Where dni=%s and edad=%s'
    valor2=(sumaVa3,sumaVa9,inten,dniLocal,edadLocal)
    ph_herramientas.updateUno(sql2,valor2)
    #en POO
    objeGene.intencionalidad = inten
    #objeGene.sumaVa3Año = sumaVa3 ya está grabado
    return inten

  def Suma(dni,edad):
    #suma va3 para las actividades activas de cada DNI por año de edad.- entre cero y
diez
```

```
    #va3 es el índice de éxito de ese año, de esa actividad activa: (va2/va4)*atractivo personal
    valorSuma = ()
    sql1="select SUM(va3), SUM(va9)  FROM tercerbd.iteracion where dni=%s and edad = %s"
    valor1=(dni,edad)
    valorSuma = ph_herramientas.selectBD(sql1,valor1)
    return valorSuma[0][0] , valorSuma[0][1]


  #He adaptado el procedimiento general a UN solo caso. Habrá declaracionbes redundantes...
  #toma dni y edad
  sumaVa3 = 0
  sumaVa9 = 0
  dniLocal=dni #ecibes el parametro
  edadLocal=edad  #ecibes el parametro
  #se calcula la suma, por cada dni, de los indices de Exito, (va2/va4)*atractivopersonal, de cada Actividad
  resultado, resulVa9 = Suma(dniLocal,edad)
  if resultado != None:
    sumaVa3 = round(resultado,2)
  if resulVa9 != None:
    sumaVa9 = round(resulVa9,2)
  #debe dar UN valor
  #Estructura dicSumas: {533: [(349.56, 69)],..,1588: [(2.8800000000000003, 9)], 1589: [(2.79, 8)]}


  objeGene = nombrePh.get(dniLocal)
  objeGene.sumaVa3Año = sumaVa3
  objeGene.sumaVa9Año = sumaVa9
  arquetipo = objeGene.miDicPhBasi['arquetipo1']
  caracter = objeGene.miDicPhBasi['caracter1']
  if sumaVa3 != None:
```

```
      asignaInten(arquetipo,caracter,dniLocal, sumaVa3,sumaVa9,edadLocal)
   return


#INICIO SALIDAS
#FIN SALIDAS


#MAIN CÓDIGO
if __name__ == '__main__':
   ph_herramientas.insertBD.llamadas =  asignaIntencionalidad.llamadas =  0
   historico = {}   #se usa en sociedad()
   poblCont = {}   #contabilidad de nacimientos y muertes...
   extincionSociedad=False
   pibViejo = 0
   tiempo = 0 #tenderá a duracionSimulacion
   #Borrado DB MySQL
   borrado()
   #Presentacion. Archivo de voz
   ph_pantallasES.preludioTraviata()
   #Pantalla captación datos iniciales
   datosIniciales=ph_pantallasES.capturaDatosIniciales()
   duracionSimulacion=int(datosIniciales[0])
   poblacionBegin=int(datosIniciales[1]) #poblacion Inicial. No cambia
   pobl=int(datosIniciales[1]) #poblacion que varia a cada vualta; ahora tienen el mismo
valor
   esperanzaEscolari=int(datosIniciales[2]) #para IDH. Esperanza años de escolarización
   anyosVidaMax = int(datosIniciales[3])
   VM = anyosVidaMax - 18 #se utiliza en tabla Actividades y para asignar actividad en
campo DuraMax
   cobranRBU = int(datosIniciales[4]) # 1 : la cobran todos; diferente de 1: solamente los >=
18
   salarioMinimo = int(datosIniciales[5])
   iAE = datosIniciales[6] #impuesto actividades económicas
```

```
   iSP = datosIniciales[7] #iSP: importancia relativa del sector publico respecto del
privado.


   #11400 es el coste INDIVIDUAL de las actividades asignadas a PI ; Todos son mayores
de 18 años: 25-40.-

   rbuCapita = 11400 if cobranRBU == 1 else 0

   pibCapita = 11400   #Comida, hijos, descanso, vestido, amistad arrojaría un total de
11400 unidades monetarias : minimo a producir

   pibNuevoSoc = pibCapita * poblacionBegin


#la sociedad, la comunidad, les provee de lo indispensable; menor de 14 años: 1/3:3800

   ingrxRBUSoc = rbuCapita * poblacionBegin

   #FIN datos iniciales pedidos a usuario o calculados x supuestos

   #info tabla actividades en regActividades;acceso global

   sqlselect= "Select * from actividades where id<%s"

   valorselect= (35,)

   regActi = ph_herramientas.recuperarTablaActividades()


   matrizLabo = []

   registroLabo = recuperarTablasitulaboposi()

   for indice,item in enumerate(registroLabo):

      matrizLabo.insert(indice,tuple(item))


   diccDecisionGeneral = formaDiccDecisionGeneral(regActi)

   #Tratamiento especial de Poblacion Inicial

   nacerPI(poblacionBegin, regActi)

   #graba en tabla nacer y en miDicPh

   atractivoPersonal() #graba en tabla nacer y tabla atractivo personal

   #necesito iniciar la tabla Sociedad para tiempo cero

   pibNuevoSoc,iAE,iSP =
ph_sociedad.Sociedad(esperanzaEscolari,pibNuevoSoc,cobranRBU,ingrxRBUSoc,iAE,iSP,
tiempo,nombrePh,pobl,anyosVidaMax)
```

```
  #la primer empresa
  formaEmpresa()

  for i in range (1,duracionSimulacion+1):

    if pobl >= 50:
      pAMuerte = 0
      tiempo = i
      print('Tratamiento general del año o iteración número: ', i)
      #solo los vivos fallecido='N' cumplen años, iteran

      registro = ph_herramientas.recuperarRegistroSociedad(tiempo-1)
      plusP = registro[21] #se toma la productividad GLOBAL (Soc) del año anterior
      plusP = int(plusP/pobl)
      #cada año forma una  empresa (nada de Familia)
      formaEmpresa()
      pibViejo=pibNuevoSoc

      nacidos = reproducirse() #Tb se forman Familias
      muertos = muerte(tiempo)
      pAMuerte = calderaDesastres(pobl,tiempo) #generacion de crisis,
pandemias...desastres.
      pobl = pobl + nacidos - muertos -pAMuerte #población de la iteración
      poblCont[i] =[pobl,nacidos,muertos,pAMuerte]

      atractivoPersonal() #graba  tabla atractivo personal

      iteracionGeneral(plusP,matrizLabo)    #asigna intencionalidad, llama a ER_ED

      #sociedad llama a calculoIDH, calculo Gini...magnitudes macroeconómicas
```

```
        pibNuevoSoc,iAE,iSP =
ph_sociedad.Sociedad(esperanzaEscolari,pibNuevoSoc,cobranRBU,ingrxRBUSoc,iAE,iSP,
tiempo,nombrePh,pobl,anyosVidaMax)
        #grabará resultados de iteracion N, que serán leidos por la N+1 and so on
        datosReprPoblPib[i]=[pobl,nacidos,muertos,pibNuevoSoc]

    else:
        print('Tratamiento general del año o iteración número(Sociedad  EXTINGUIDA):  ', i)
        break


  if pobl >= 50:
    ph_salidas.tratamientoDatos(nombrePh,
datosReprPoblPib,poblCont,poblacionBegin)  # Todas las salidas agrupadas
    ph_seleDist.main()  #comparacion distribuciones Ingresos y Gastos. Algoritmo de
Amat

  sinfo()
  #session_info.show() #Creo NO funciona con spyder
  miConexion.close()


  print('llamadas aproximadas a
ph_herramientas.insertBD',ph_herramientas.insertBD.llamadas)

  print('llamadas para asignar intencionalidad',asignaIntencionalidad.llamadas)
  if  pobl >= 100:
    print('La Sociedad se EXTINGUIÓ')
  print('HISTORICO Dif ingresos gastos en Organizacion, iAE e iSP----->',historico)
  print('Contabilidad de población',poblCont)
  print ("Esta es la última instrucción")
```

```
# -*- coding: utf-8 -*-
"""
Created on 24/6/24
```

ERED debe ser el algoritmo por el que tomamos nuestras decisiones: hacemos esto y dejamos de hacer aquello...

Son decisiones individuales que deben de tener en cuenta :

los deseos individuales, la situación familiar, amistades, conocidos...y económica.- Intencionalidad y palp

```
@author: invat
"""

import random
import mysql.connector
import ph_herramientas
#Con los imports activos salia error debido a la circularidad
#Opte por eliminar estas importaciones y poner el código -duplicandolo-

#Duplicado
miDicActividades={1:'Comer',2:'Pareja',3:'Hijos',4:'Descanso',5:'Sexo',6:'Salud',7:'Amistad',
8:'Estudio',\
        9:'Preparar oficio',10:'Trabajo',11:'Vestirte',12:'Casa',13:'Coche',\
        14:'Milit pol-sindi',15:'Deporte',16:'Adicciones', 17:'Escribir leer',18:'Asac recreati',\
        19:'Satis Curiosid',20:'Jugar Azar',21:'Familia',22:'Manualidades',23:'Militancia
ONG',24:' Arte Ciencia',\
        25:'Viajar Emigrar', 26:'Seguir la moda',27:'Ejercer poder',28:'Investigar',\
        29:'Emprender',30:'Deporte riesgo',31:'Lider público',32:'Espiritualidad',33:'Estudios
medios',34:'Estudios superiores'}

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
   user="invat",password="jquintas49",auth_plugin="mysql_native_password")

def recuperarTablasitulaboposi():
   #se cargan los 32 primeros registros de la tabla actividades
```

```
    sqlselect= "Select * from tercerbd.situlaboposi "
    regActi = ph_herramientas.selectBDNoWhere(sqlselect)
    regis = [list(a) for a in regActi]
    return  regis



#FIN duplicado


def
ERealEDeseo(dni,edad,nombrePh,salarioMinimo,cobranRBU,rbuCapita,plusP,tiempo,pob
l,diccDecisionGeneral,matrizL,objeGene):
    #ph_ER_ED.llamadas += 1
    #ER-ED trabaja con su code y con sus  funciones internas
    #paso de variables y funcion del return: normal
    #recuerda: las variables de la función externa pueden ser consultadas en las internas
    #en funciones internas pueden modificar variables externas si usas:'nonlocal'

    #Actividades por edad
    def actividadesMenor18():

      #en miDicPh lo que está escrito se mantiene, está en RAM
      #en iterGrabaLocal esta en MySQL, se ha de grabar cada iteración
      numActi = [15,17,19,20,26]

      if 11 <= edad <= 15 :
          #deja miDicPh , añadiendo  de 1 a 3 voluntarias no Maslow
        for i in range (1,3):
          ii=0
          azar0=random.randint(0,4) # 0 a 4 ,inclusive
          #listas , tuplas...comienzan con CERO
          #15 depporte, 19 curiosidad, 20 jugar, 17 escribir leer,26 seguir moda
```

```
            ii = numActi[azar0]


            if  ((regActividades[ii-1][10] in miAC) or (regActividades[ii-1][11] in miAC)):
               cargarActiDat1245(dni,ii)
        elif  edad >15 :
            decisionUnidades(diccDecisionGeneral,'Deporte')


        return


    def actividadesMayor17Menor65():
        #la 1,4,6,7,8,9,11 YA están escritas en el diccionario
        #el trabajo Se incluye APARTE
        #decisiones: es decir no están puestos de serie al nacer
        #decidirSobre podía ser muyyyyy largo
        #A los que en reproducir se les ha puesto un idFiscalFamilia se les pone en actividad
Familia. Manu militari


        if objeGene.idFiscalFamilia != 0:
           try:
              cargarActiDat1245(dni,21)
           except KeyError:
              print('Ya estaba dada de alta la actividad')


        decidirSobre = {2:'Pareja',3:'Hijos', 12:'Casa',\
                  13:'Coche',15:'Deporte' }
        dictPh = objeGene.miDicPh
        #extrae los que estan en decisirSobre y no estan en miDicPh
        dict3 = set(decidirSobre) - (set(dictPh))
        dict4 = {}


        for item in dict3:
```

```
        valor = decidirSobre[item]
        dict4[item] = valor
        decisionUnidades(diccDecisionGeneral,valor)
    #Ahora de los que están en decidirSobre y tambien estan en miDicPh: se envian a
control
    dict3 = set(decidirSobre) & set(dictPh)
    for item in dict3:
        valor = decidirSobre[item]
        dict4[item] = valor
        controlUnidades(item)


    if edad ==25 or edad ==35 or edad == 45 or edad == 55:
        #va a intentar añadir alguna actividad, tres veces: maximo 9 actividades
        for i in range (1,5): #vueltas:1,2,3,4
            azar0=random.randint(14,34) #apuede salir el 14 e intermedios hasta el 34,
inclusive.-


            for ii in range (azar0,35):  #llega a 31- del 0 al 31 van 32
                if (regActividades[ii-1][10] in miAC) or (regActividades[ii-1][11] in miAC):
                    if ii not in objeGene.miDicPh:
                        i = ii #num actividad
                        cargarActiDat1245(dni,i)
                    continue #para salirse del for
    return


  def actividadesMayor64():
    if edad>=65 :
        if 2 not in objeGene.miDicPh:
            decisionUnidades(diccDecisionGeneral,'Pareja')
        if 15 not in objeGene.miDicPh:
            decisionUnidades(diccDecisionGeneral,'Deporte')
        if 25 not in objeGene.miDicPh:
```

```
            decisionUnidades(diccDecisionGeneral,'Viajar Emigrar')
          decisionDistribuyeTuTiempo()
      return

   #herramientas de ayuda y control actividades
   def cargarActiDat1245(objDni,numActi):
      #DAR DE ALTA UNA ACTIVIDAD -numActi- EN  miDicPh() del ph(objDni)
      objeGene=nombrePh.get(objDni)
      i = numActi
      #actvidades que tienen 0 en  coste de tiempo
      regActividades[i-1][5] =  random.randint(8,25) if regActividades[i-1][5] == 0 else
regActividades[i-1][5]
      objeGene.miDicPh[i] = ['X',0,0,0,0,0,0,0,0,0]
      objeGene.miDicPh[i][0]=regActividades[i-1][1]  #nombre Actividad
      objeGene.miDicPh[i][1]=regActividades[i-1][4]  #coste um
      objeGene.miDicPh[i][4]=regActividades[i-1][5]  #coste tiempo
      objeGene.miDicPh[i][7]=regActividades[i-1][2]  #riesgo actividad

      return

   def actividadesxArquetipo(arqueti):
      #nonlocal miDicPh y nonlocal iterGrabaLocal, no necesario
      #Aunque viene todas las iteraciones, solo se asigna UNA vez

      diccioArquet={'amigo':[7,16],'cuidador':[23] ,'explorador':[25] ,'heroe':[15,30] ,\
            'inocente':[26] ,'rebelde':[14] ,'sabio': [19],'gobernante':[27,31 ],\
            'creador':[17,24,28] ,'bufon':[18],'mago':[8,29],'amante':[5]}

      sumaExtra = {'amigo':[5],'cuidador':[7] ,'explorador':[7] ,'heroe':[8] ,'amante':[2],\
            'inocente':[5] ,'rebelde':[5] ,'sabio': [10],'gobernante':[5],\
            'creador':[8] ,'bufon':[2],'mago':[7] }
```

```
    arquetipo = arqueti

    if edad == 10 or edad == 18 :
      try:
        for item in diccioArquet[arquetipo]:
          i = item #si hay dos o 3 items, solo se toma el primero
          cargarActiDat1245(dni,i)
          objeGene.miDicPh[i][3] += sumaExtra[arquetipo][0] #indice éxito Deporte
      except KeyError:
        pass
    return

  def  actualizaContadores():
    #en objeGene está el ph CONCRETO en cada iteración
    #añade un año de practica a las Actividades activas que no sean 2,5
    #Indice de éxito de cada Actividad (antes trae el indice atractivo personal)

    excepciones=[2,5] #pareja y sexo: tienen azar.
    estadosInactivos=['X','N']
    atracPerso={1:0,2:0,3:0,4:0,5:0,6:0,7:0, 8:0,9:0,10:0,11:0,\
      12:0,13:0,14:0,15:0,16:0, 17:0,18:0,19:0,20:0,\
      21:0,22:0,23:0,24:0,25:0,26:0,27:0,28:0,29:0,30:0,31:0,32:0}
      #en este diccionario debe meterse indice atractivoPersonal de cada ph
    sql1='select actividad, atractivo from atractivopersonal where dni=%s'
    valor1=(dni,)
    registro1=ph_herramientas.selectBD(sql1,valor1) #da LISTA de TUPLAS
    #pasar tuplas a diccionario
    for i in range(1, 35): # uno mas: son 34 significativas
      atracPerso[i] = registro1[i-1][1] #
    for i in objeGene.miDicPh.keys(): #itera actividades activas
```

```
        if i not in excepciones:
          if  objeGene.miDicPh[i][0] not in estadosInactivos:
            objeGene.miDicPh[i][2] += 1
            numerador = int(objeGene.miDicPh[i][2])   #va2
            try:
              # indiceExito de cada Actividad (menos las tres excluidas)
              if numerador >= objeGene.miDicPh[i][4]:
                objeGene.miDicPh[i][4] = numerador #cociente será la unidad; intenta evitar efecto de Actividades con va2 "raro"
              objeGene.miDicPh[i][3] = round(numerador/objeGene.miDicPh[i][4],2)*(atracPerso[i])
            except ZeroDivisionError:
              print("No se ha podido realizar la división pues va4 es nulo")
            objeGene.miDicPh[i][6] += 1 #para índice eficacia
            numerad = int(objeGene.miDicPh[i][6]) #va5 max deseado, va6 logrado indice eficacia: va6/va5 * atracPerso
            try:
              # indiceEficacia de cada Actividad (menos las tres excluidas)
              if numerad >= objeGene.miDicPh[i][5]:
                objeGene.miDicPh[i][5] = numerad #cociente será la unidad; intenta evitar efecto de Actividades con va5 = 0
              objeGene.miDicPh[i][9] = round(numerad/objeGene.miDicPh[i][5],2)*(atracPerso[i])
            except ZeroDivisionError:
              print("No se ha podido realizar la división pues va5 es nulo")
      return


    def  experienciasActividades():
      #en objeGene está el ph CONCRETO
      #Van añadirse actividades TIPO: Optativas
      #excepciones=[1,2,3,4,5,6,7,8,9,10,11,12,13,21,] #Son TIPO = básicas o normales
      #actividadesPosiblesUnidades =[2,3,5,7,9,12,13,14,15,16,17,18,19,20,22,23,24,25,26,27,28,29,30,31,32] #25 actividades
```

```
    posiblesActividades=[14,15,16,17,18,19,20,22,23,24,25,26,27,28,29,30,31,32,33,34]
    estadosInactivos=['X','N']
    # los estadosInactivos son ['X','N']
    #elimino actividad con poco indiceExito de la actividad
    nombrePhClavesCongeladas = []
    nombrePhClavesCongeladas = list(objeGene.miDicPh)
    for i in nombrePhClavesCongeladas :
      if objeGene.miDicPh[i][0] not in estadosInactivos:
        if i  in posiblesActividades and ( objeGene.miDicPh[i][3] < 0.3 or
objeGene.miDicPh[i][9] < 0.3):  #indiceExito o indiceEficacia bajo
          print('HE QUITADO',objeGene.miDicPh[i][0])
          objeGene.miDicPh[i][0] = 'N' #hiciste esta actividad
          ii = i
          #eliminaste una Actividad, intentamos asignarte otra por tres veces
          #todas las incluidas como posibles estan en tipo: Optativas
          dentro=True
          vez=0
          while dentro and vez < 4:
            azar0=random.randint(14,31)
            vez +=1
            if (azar0 in posiblesActividades) and (ii != azar0): # asegura que la actividad
quitada se vuelva a poner, creo
              i = azar0 #num actividad
              cargarActiDat1245(dni,i)
              dentro = False
              print('HE AÑADIDO', objeGene.miDicPh[i][0])
    return


  def decisionUnidades(diccDecisionGeneral,deciSobre):
    #deciSobre debe ser del tipo: 'pareja', 'hijos'...
```

```
    #X: No se está ejecutando, N: se ejecutó ,pero ahora no.- Si tiene nombre actividad: se está ejecutando

    #si asigna es menor: se queda como estuviera

    #decisionGeneral [actividad, azarMayorq,azarUnidad: num de unidades max a que se aspira]

    #actividadesPosiblesUnidades =[2,3,9,12,13,14,15,16,18,20,22,23,24,26,28,29,30,31,32]

    #Pde cada actividad posible tomamos su nombre, su número,el numero que debe superarse, num máximo unidades, la duración antes de ser obsoleto

    #se toma de la tabla actividades y se mete en el diccionario: decisionGeneral

    #decisionGeneral = {'Pareja':[2,6,10],'Hijos':[3,1,10],'Sexo':[5,3,2],'Casas':[12,4,4],'Coches':[13,1,3],\

     #             'Familia':[21,6,1],'Viajar':[25,1,1]}

    decisionGeneral = diccDecisionGeneral


    actividad = decisionGeneral[deciSobre][0]

    azarMayorq = decisionGeneral[deciSobre][1]

    azarUnidad = decisionGeneral[deciSobre][2]

    obsolescencia = decisionGeneral[deciSobre][3]

    asigna=random.randint(1,9)

    if asigna > azarMayorq:

      i = actividad #num actividad

      cargarActiDat1245(dni,i)

      objeGene.miDicPh[actividad][0]=miDicActividades[actividad]

      if azarUnidad != 1:

        azar = random.randint(1,azarUnidad)

        if actividad == 3 or actividad == 12 or actividad == 13:

          objeGene.miDicPh[actividad][5]=azar  #va5)

        else:

          objeGene.miDicPh[actividad][4]=azar  #va4)

    return


  def decisionDistribuyeTuTiempo():
```

```
    #Indicar que es una forma de proceder. Puede ampliarse
    azar=random.randint(0,5)
    if azar>2:
      i = 22 #num actividad    #manuealidades
      cargarActiDat1245(dni,i)
      i = 18 #num actividad    #asoc recreativas
      cargarActiDat1245(dni,i)
    return

  def controlUnidades(actividad):
    #hijos,casas,coches,ochomiles...Unidades.- V(-9.1)
    #Si cumplen objetivo, escribe N; si aún no lo cumple= suma una Unidad
    i = actividad
    deseadas = objeGene.miDicPh[i][5]
    habidas = objeGene.miDicPh[i][6]
    diferencia = deseadas-habidas
    if diferencia > 0 :
      objeGene.miDicPh[i][6] += 1 #vañade una unidad
    else:
      objeGene.miDicPh[i][0] ='N' # 'N' : ya han tenido las Unidades que deseaban
    return

  def controlCambioEmpresa():
    #no me caliento la cabeza
    #miDicPh[10][6] ahí debe estar el num de años que lleva en la empresa
    cambia=random.randint(1,9)
    miAntiguedad = int(objeGene.miDicPh[10][6])
    antiguedadMediaEmpresa=10
    if antiguedadMediaEmpresa > miAntiguedad :
      objeGene.miDicPh[10][6] = miAntiguedad + 1
      #Sigue en la empresa
```

```
    elif antiguedadMediaEmpresa <= miAntiguedad  and cambia <6:
      objeGene.miDicPh[10][6] = miAntiguedad + 1
       #Sigue en la empresa
    elif antiguedadMediaEmpresa <= miAntiguedad  and cambia >=6:
      #cambia de empresa
      baja_Empresa()
      altaEmpl_Empr()
    return


  def independizarse():
    #los hijos pueden formar otra familia
    sql="UPDATE organiza_miembros SET esBaja = 'S' where id_nacer=%s"
    valor=(dni,)
    ph_herramientas.updateUno(sql,valor)
    return


  #temas de TRABAJO
  def asignaTrabajo(edad,riesTole,dni,rbuCapita,plusP,matrizLabo):
    #NO paso objeGene pues es función interna
    #segun antiguedad cambia trabajo y asigna cuantias según tabla empleos y
situaciones
    def rutina1(PlusP,azar,rbuCap):
      azarCuan = azar
      PlusP = PlusP
      rbuCapita = rbuCap
      objeGene.climLabo = 'Regul' #a modificar
      objeGene.antiguedad = 1 if  (objeGene.antiguedad == 0 or objeGene.antiguedad ==
9)  else objeGene.antiguedad + 1
      if objeGene.antiguedad == 1 or objeGene.antiguedad == 9 : #Si tiene 1: instrucción
anterior era cero
        azarProf = random.randint(7,206)
        #en tabla va de 8 a 207, pero en lista va de 0-206: matriz[6][0]=7 (S-dificil)
```

```
        #Si es S-dificil alto salario
        if objeGene.situlaboposi == 7 :
            objeGene.ingrxTrabB = int(matriz[7][2])
        elif objeGene.situlaboposi != 7 :
            azarProf = random.randint(7,206)
            if  (objeGene.antiguedad == 1 or objeGene.antiguedad == 9):
                objeGene.ingrxTrabB =  int(matriz[azarProf][2]/azarCuan)
                objeGene.situlaboposi = matriz[azarProf][0]
            elif 2 <= objeGene.antiguedad < 9:
                objeGene.ingrxTrabB = objeGene.ingrxTrabB
        objeGene.plusProductividad = PlusP + (PlusP/random.randint(8,10)) -
(PlusP/random.randint(5,10)) \
            if objeGene.plusProductividad < objeGene.ingrxTrabB else
objeGene.ingrxTrabB/10
        objeGene.ingrxTrab = objeGene.ingrxTrabB + objeGene.plusProductividad
        objeGene.ingrxRBU = rbuCapita
        objeGene.ingrTotal = objeGene.ingrxTrab + objeGene.ingrxRBU
        return
    def rutina2():
        objeGene.ingrxTrab = objeGene.ingrxTrabB + objeGene.plusProductividad
        objeGene.ingrxRBU = rbuCapita
        objeGene.ingrTotal = objeGene.ingrxTrab + objeGene.ingrxRBU
        return


    #MAIN asignaTrabajo()
    edad = edad
    rbuCapita = rbuCapita if edad >= 14 else (rbuCapita/3)
    riesgo=riesTole
    riesgoBajo=(0,1,2,3)
    riesgoAlto=(4,5)
    PlusP = plusP
    matriz = matrizLabo
```

```
azar=random.randint(1,9) #1 y 9 incluidos
if 1 <= edad < 18:
   objeGene.ingrxTrabB = 0
   objeGene.plusProductividad = 0
   rutina2()
   objeGene.situlaboposi = 1 if edad < 14 else 2

elif  18 <= edad < 65 and riesgo in riesgoBajo :
   if 18 <= edad < 25:
      azarCuan = random.randint(5,6)
      rutina1(PlusP,azarCuan,rbuCapita )

   elif 25 <= edad <32:
      azarCuan = random.randint(4,5)
      rutina1(PlusP,azarCuan,rbuCapita )

   elif 32 <= edad <40:
      azarCuan = random.randint(3,5)
      rutina1(PlusP,azarCuan,rbuCapita )

   elif 40 <= edad <50:
      azarCuan = random.randint(2,4)
      rutina1(PlusP,azarCuan,rbuCapita )

   elif 50<= edad < 65:
      azarCuan = random.randint(1,3)
      rutina1(PlusP,azarCuan,rbuCapita )

elif   18 <= edad < 65 and riesgo in riesgoAlto :
   if  azar <7:
      azarCuan = random.randint(1,4)
```

```
            rutina1(PlusP,azarCuan,rbuCapita )
        elif 7 == azar == 8 :
            #Casos raro:enfermedad crónica, accidente laboral...
            azarCuan = random.randint(4,5)
            objeGene.ingrxTrabB = int(matriz[5][2]/azarCuan)
            objeGene.plusProductividad = 0
            rutina2()
            objeGene.situlaboposi = 5
            objeGene.antiguedad = 0 #infancia y jubilados no cuenta la antiguedad
        elif azar ==9 and objeGene.intencionalidad=='acabemos':
            objeGene.ingrxTrabB = 0
            objeGene.plusProductividad = 0
            rutina2()
            objeGene.situlaboposi = 4 #situación que impide trabajar
            objeGene.antiguedad = 0 #infancia, oscuros y jubilados no cuenta la antiguedad
    elif edad>=65:  #debes crear los IMPUESTOS, no todos pueden tener la contributiva
        #le toca por edad, pero ¿y si es NO contributiva?
        azarCuan = random.randint(1,3)
        objeGene.ingrxTrabB = int(matriz[5][2]/azarCuan)
        objeGene.plusProductividad = 0
        rutina2()
        objeGene.situlaboposi = 5
        objeGene.antiguedad = 0 #infancia y jubilados no cuenta la antiguedad

    return

def asignaTrabajoSDificil():
    #edad,riesTole y otras externas las pilla por eso mismo: son externas y puede leerlas
    #Creo que objeGene es conocido cada este dni
    trabajoAsignado=''
    actividad=''
```

```
    azar=random.randint(0,13) #ambos inclusive
    actividadSdificil=(5,10,14,15,17,23,24,27,28,29,30,31,32,34)

tablaACSdificil=('creador','cuidador','sentimental','heroe','rebelde','gobernante','mago','apasionado','flematico','sanguineo','amante')
    #colerico no está...hay muchos....Estos son los arquetipos que pueden dar lugas a S-dificil
    ruleta=azar #elige la actividad S-dificil que se va a chequear: solamente una

    i = actividadSdificil[ruleta]

    check=regActividades[i-1][10]

    #si check esta en las dos listas
    if objeGene.situlaboposi == 7 and 0 < objeGene.antiguedad < 8:
      pass
    elif objeGene.situlaboposi == 7 and 0 < objeGene.antiguedad == 8:
      objeGene.situlaboposi = 3 #se va al paro
      objeGene.antiguedad = 1
      return
    #elif ((check in tablaACSdificil) and (check in miAC) and (riesTole >= regActividades[i-1][2] or objeGene.potencial == '3-Malto')) :
    elif (check in tablaACSdificil and  objeGene.potencial == '3-Malto') :
      trabajoAsignado = 'S-dificil'
      actividad = regActividades[i-1][1]
      objeGene.situlaboposi = 7  # es la clave de S-Dificil en la tabla situlaboposi
      objeGene.antiguedad = 1
    else:
      pass
    return trabajoAsignado, i,actividad

  def altaEmpl_Empr():
```

```
    #Por un doble random mete al trabajador en una empresa SI no está activo en alguna, es decir esBaja ='S' o es la PRIMERA
    sql = "Select count(*) from organiza_miembros where id_nacer = %s and esBaja = 'N'"
    valor = (dni,)
    existeEmpr = ph_herramientas.selectBD(sql,valor) #count debería ser cero ó uno
    if existeEmpr[0][0] == 0 : #no tiene empresa activa; si >>0, entonces tiene empresa: control
      dentro=True
      while dentro:
        tipo = ['industria','servicios','campo','enseñanza','sanidad',\
          'software','seguridad','investigacion','finanzas','administración']
        azar = random.randint(0,9)
        tipo_Empresa = tipo[azar]
        sql="SELECT dniFiscal from organizaciones where tipo=%s"
        valor=(tipo_Empresa,)
        empresas_posibles = ph_herramientas.selectBD(sql,valor)
        limite = len(empresas_posibles)
        if limite >=1:
          dentro = False
      azar_empresa = random.randint(0,limite)
      idFiscal_Empr = empresas_posibles[azar_empresa-1] [0]
      sql="INSERT INTO organiza_miembros (idFiscal,id_nacer,tipo) VALUES (%s,%s,%s)"
      valor = (idFiscal_Empr, dni,tipo_Empresa)
      ph_herramientas.insertUno(sql,valor)
      objeGene.idFiscalEmpresa=idFiscal_Empr #ya es distinto de cero; empresa actual
    return


  def baja_Empresa():
    idBaja = objeGene.idFiscalEmpresa
    idDni =  objeGene.dni
    sql1 = "UPDATE organiza_miembros SET esBaja = 'S'  WHERE idFiscal=%s and id_nacer= %s and esBaja = 'N'"
```

```
    valor1=(idBaja,idDni)
    ph_herramientas.updateUno(sql1,valor1)
    return

  #Economia  Individual

  def formaGasto():
    #acumula todos los gastos de todas las actividades del año
    gastoAnyo= 0
    convenciones=('X','N') #he quitado S-dificil
    for ii in objeGene.miDicPh:
      if objeGene.miDicPh[ii][0] not in convenciones:
        gastoAnyo=gastoAnyo+objeGene.miDicPh[ii][1]
    gastoAnyo=int(gastoAnyo)
    return gastoAnyo

  #(MAIN de ER_ED)

  #Se llama por CADA ph seleccionado, desde edad General
  dni = dni
  edad = edad
  plusP = plusP
  tiempo = tiempo
  #decisionGeneral = diccDecisionGeneral
  miAC = []
  arquetipo = ''
  regActivi = ph_herramientas.recuperarTablaActividades()  #no tiene sentido recuperarla
1000 veces
  regActividades = [list(a) for a in regActivi]
  matrizLabo = matrizL
  objeGene = objeGene
```

```
    #grabo en el ph de POO la edad
    objeGene.iteracion = tiempo  # tiempo es la iteracion
    dni = objeGene.dni
    edad = objeGene.edad
    #los dos arquetipos y los dos caracteres
    miAC.append(objeGene.miDicPhBasi['arquetipo1'])
    miAC.append(objeGene.miDicPhBasi['arquetipo2'])
    miAC.append(objeGene.miDicPhBasi['caracter1'])
    miAC.append(objeGene.miDicPhBasi['caracter2'])

    riesTole = objeGene.riesgoTolerancia
    #fallecido=objeGene.riesgoTolerancia
    arquetipo = miAC[0] #le pasa arquetipo1 al actividadesxArquetipo




    if 1 <= edad < 18:
        actividadesxArquetipo(arquetipo) #todos los años para que sume a lo
actualizacionContadores
        actividadesMenor18()
    elif 18 <= edad < 65:
        altaEmpl_Empr()  if objeGene.idFiscalEmpresa == 0 else controlCambioEmpresa()
        actividadesxArquetipo(arquetipo)
        #mira los indices de exito de algunas actividades y deja los mayores, elimina los <5 e
intenta asignar
        experienciasActividades()
        actividadesMayor17Menor65()
    elif edad>=65:
        experienciasActividades()
        actividadesMayor64()
    if 18 <= edad < 65 and objeGene.potencial == '3-Malto' and objeGene.situlaboposi != 7:
```

```
      asignaTrabajoSDificil() #se asigna por actividades; una vía diferente al trabajo. Si se le asigna: en def rutina1() se contempla

   asignaTrabajo(edad,riesTole,dni,rbuCapita,plusP,matrizLabo) #asigna tb ciudadania y jubilacion


   #objeGene concreto: actualizo contadores miDicPh (va3):en Actividades activas

   actualizaContadores()


   #todo es relativo al individuo, al ph() en curso

   #ingresoRBU = ingresoxCiudadania() #deberia estar fuera de este modulo, como una asignación segun el año anterior

   #ingrxTrabB,ingrxTrab,ingresoPlusP = ingresoxTrabajo(trabajo)

   ingrTotal = objeGene.ingrTotal  #

   gastoAnyo = formaGasto()

   objeGene.palp = objeGene.palp + int(ingrTotal)-int(gastoAnyo)  #NO Considera ACTIVOS...es el AHORRO ACUMULADO

   objeGene.gastoAnyo = gastoAnyo # caractwer anual


   return

#FIN DE ER_ED
```

```
# -*- coding: utf-8 -*-
"""
Created on Sábado 24/06/2024
ph_pantallasES.py
HOSS (Human Organizations Simulation Start) -9
José Quintás Alonso
@author: invat
"""
import tkinter  as tk
from tkinter import messagebox
from tkinter import Label, Frame, Entry, Button,Text,INSERT
from playsound import playsound
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import  FigureCanvasTkAgg
import matplotlib.pyplot as plt
import os
import webbrowser
from PIL import Image,ImageTk


datosIniciales=[]
datosSalida=[]

def preludioTraviata():
   #utilizando Tkinter Preludio de la aplicación, esplicar, acceder a Internet
   raiz=tk.Tk() #creo una variable
   raiz.geometry("800x500")  #ancho y alto
   raiz.config(bg='yellow')
   raiz.title('Simulación de una Sociedad de ph(). Versión -7')

   #Frame
```

```
  miframe=Frame() #hecho el Frame, es trasparente
  miframe.pack() #empaquetar
  miframe.config(bg="red")
  miframe.propagate(0)
  miframe.config(width="200",height="100")



  def playMp1():
    playsound('C:/Users/invat/MisDocumentos/A_ProyectoPython/Proyecto Python ph-7/2.mp3')


  def playMp4():
    #aquí irá la dirección de Onuglobal
    webbrowser.open('https://onuglobal.files.wordpress.com/2023/10/hoss16102023-1.mp4')


  def playMp3():
    playsound('https://onuglobal.files.wordpress.com/2023/10/hoss.mp3')


  def linkBotonGauss():
    webbrowser.open('https://www.geogebra.org/m/Wbb6Sk5C')


  def linkBotonGamma():
    webbrowser.open('https://www.geogebra.org/m/yCZgR3dh')


  def funcionBotonSS():
    raiz.destroy()



  img = Image.open('C:/Users/invat/MisDocumentos/A_ProyectoPython/Proyecto Python ph-7/altavoz.jpg')
  img = img.resize((50, 50))
```

```
    img = ImageTk.PhotoImage(img)


    playMp1_button = Button( miframe,image=img, text="Audio
explicativo",command=playMp1)
    playMp1_button.pack()
    playMp1_button.pack()
    playMp1_text = tk.Label (miframe,text="Audio explicativo V: -7")
    playMp1_text.pack()
    playMp4_button = Button(raiz, text="Ver post y video descriptivo de HOSS V:-10",
command=playMp4)
    playMp4_button.pack(pady=5)
    playMp3_button = Button(raiz, text="Agradecimientos", command=playMp3)
    playMp3_button.pack(pady=5)
    botonMostrar=Button(raiz,text="Click para ir a Internet: distribución
Gauss",command=linkBotonGauss)
    botonMostrar.pack()
    botonURL=Button(raiz,text="Click para ir a Internet: función
Gamma",command=linkBotonGamma)
    botonURL.pack()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()
    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción


    return



def capturaDatosIniciales():
    #Datos iniciales pedidos al usuario
    def verificar_numeros():
        numero1 = int(entry_numero1.get())
        numero2 = int(entry_numero2.get())
```

```
    numero3 = int(entry_numero3.get())


    numero7 = int(entry_numero7.get())
    numero8 = int(entry_numero8.get())
    numero9 = int(entry_numero9.get())
    numero10 = int(entry_numero10.get())
    numero11 = float(entry_numero11.get())


    if 25 <= numero1 <= 200 and 25 <= numero2 <= 2000 and 10<= numero3 <= 20 \
      and 60<= numero7 <=85   and (numero8 == 0 or numero8 == 1) and 500<= numero9
<=3001 and \
        10 <= numero10 <= 50  and 0.001 <= numero11 <= 1000 :
      messagebox.showinfo("Validación exitosa", "¡Las entradas son válidos!")
      btn_salir["state"] = tk.NORMAL
    else:
      messagebox.showerror("Error de validación", "¡Las entradas no cumplen con los
requisitos!")


  def salir():
    #Pongo los Entryes en la lista datosIniciales
    datosIniciales.append(entry_numero1.get()) #posicion cero
    datosIniciales.append(entry_numero2.get())
    datosIniciales.append(entry_numero3.get())


    datosIniciales.append(entry_numero7.get()) #posicion 6
    datosIniciales.append(entry_numero8.get()) #posicion 7
    datosIniciales.append(entry_numero9.get()) #posicion 8
    datosIniciales.append(entry_numero10.get()) #posicion 9
    datosIniciales.append(entry_numero11.get()) #posicion 10
    ventana.destroy() #salir


  # Crear la ventana principal
```

```
    ventana = tk.Tk()
    ventana.title("Validación de números")
    ventana.geometry("800x800")

    label_numero0 = tk.Label(ventana, text = "Introducir números, validar, introducir Datos y Salir")
    label_numero0.pack()
    label_numero00 = tk.Label(ventana, text = "Los valores de 'sociedad' deberian de obtenerse del desarrollo tecnologico, organizativo y del capital disponible")
    label_numero00.pack()
    # Etiqueta y campo de entrada para el primer número
    label_numero1 = tk.Label(ventana, text = "Dos Datos necesarios para ejecutar la aplicación",)
    label_numero1.pack()
    label_numero1 =tk.Label(ventana, text="Número de años simulación (25-200) :")
    label_numero1.pack()
    entry_numero1 = tk.Entry(ventana)
    entry_numero1.pack()

    # Etiqueta y campo de entrada para el segundo número
    label_numero2 = tk.Label(ventana, text="Población inicial (25-2000) :")
    label_numero2.pack()
    entry_numero2 = tk.Entry(ventana)
    entry_numero2.pack()

    # Etiqueta y campo de entrada para el tercer número
    label_numero3 = tk.Label(ventana, text = "DATOS INICIALES" )
    label_numero3.pack()
    label_numero3 = tk.Label(ventana, text="Años escolarización (10-20) -sociedad:")
    label_numero3.pack()
    entry_numero3 = tk.Entry(ventana)
    entry_numero3.pack()
```

```
    # Etiqueta y campo de entrada para el cuarto número



    # Etiqueta y campo de entrada para el quinto número



    # Etiqueta y campo de entrada para el sexto número



    # Etiqueta y campo de entrada para el septimo número
    label_numero7 = tk.Label(ventana, text="Años medios de Vida (60-85) -sociedad:")
    label_numero7.pack()
    entry_numero7 = tk.Entry(ventana)
    entry_numero7.pack()
    # Etiqueta y campo de entrada para el octavo número
    label_numero8 = tk.Label(ventana, text="Reciben RBU: Nadie (0), Todos(1)-sociedad:")
    label_numero8.pack()
    entry_numero8 = tk.Entry(ventana)
    entry_numero8.pack()

    # Etiqueta y campo de entrada para el noveno número
    label_numero9 = tk.Label(ventana, text="Salario mínimo  (500-3001) -sociedad:")
    label_numero9.pack()
    entry_numero9 = tk.Entry(ventana)
    entry_numero9.pack()

    # Etiqueta y campo de entrada para el décimo número
    label_numero10 = tk.Label(ventana, text="Impuesto Actividades Económicas (10-50) -
sociedad:") #iAE
    label_numero10.pack()
```

```
    entry_numero10 = tk.Entry(ventana)
    entry_numero10.pack()

    # Etiqueta y campo de entrada para el undécimo número
    label_numero11 = tk.Label(ventana, text="Masa monetaria sector Privado/Masa monetaria sector Sector Público igual a: 0,001-1000) -sociedad:") #iSP
    label_numero11.pack()
    entry_numero11 = tk.Entry(ventana)
    entry_numero11.pack()

    # Botón para verificar los números
    boton_verificar = tk.Button(ventana, text="Verificar", command=verificar_numeros)
    boton_verificar.pack()

    btn_salir = tk.Button(ventana, text="Salir", command=salir, state=tk.DISABLED)
    btn_salir.pack()

    ventana.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción
    return datosIniciales




def capturaDatosSalida():
    #utilizando Tkinter, tomo dos datos (año de inicio y población inicial)
    #Se los paso a los siguientes procedimientos
    #podrian ampliarse los campos que se piden, habría más parametros de DISEÑO
    raiz=tk.Tk() #creo una variable
    raiz.geometry("700x400")
```

```
raiz.config(bg='yellow')
raiz.title('Simulación Sociedad. Versión -1')
miVariable=tk.StringVar() #los parentesis, fundamentales
#Frame
miframe=Frame() #hecho el Frame, es trasparente
miframe.pack() #empaquetar
miframe.config(bg="red")
#miframe.config(width="1300",height="700")
miframe.propagate(0)
miframe.config(width="650",height="350")

instFrame=Frame() #Frame de instrucciones
instFrame.pack() #empaquetar
instFrame.config(bg="blue")
#miframe.config(width="1300",height="700")
instFrame.propagate(0)
instFrame.config(width="650",height="350")
#witges text y button, nuestro objetivo
miLabelText=Label(instFrame, text="Consideraciones")
miLabelText.grid(row=0,column=0)
miEntryText=Text(instFrame,width=60,height=3)
miEntryText.grid(row=1,column=0)
miEntryText.insert(INSERT,"Se ejecutan TODAS las opciones" )


miLabelA=Label(miframe, text=" Elija un 'ph' válido para conocer su estado a través de
la impresora")
miLabelA.grid(row=1,column=0,sticky="e",padx=15,pady=15)
miEntryPh=Entry(miframe,textvariable=miVariable)
miEntryPh.grid(row=1,column=1)
```

```
  miLabelB=Label(miframe, text="Elija un 'ph' válido para conocer su último estado de
Actividades")
  miLabelB.grid(row=2,column=0,sticky="e",padx=15,pady=15)
  miEntryHijo=Entry(miframe)
  miEntryHijo.grid(row=2,column=1)

  miLabelC=Label(miframe, text="Para un año concreto, ¿cual es el palp de los vivos?")
  miLabelC.grid(row=3,column=0,sticky="e",padx=15,pady=15)
  miEntryPalp=Entry(miframe)
  miEntryPalp.grid(row=3,column=1)

  miLabelD=Label(miframe, text="Para un 'ph', ¿cual es su espacio de fases?")
  miLabelD.grid(row=4,column=0,sticky="e",padx=15,pady=15)
  miEntryEspaFase=Entry(miframe)
  miEntryEspaFase.grid(row=4,column=1)

  def funcionBotonMS():
    #Pongo los Entryes en la lista datosIniciales
    messagebox.showinfo("Estado de:", miEntryPh.get() +">>Actividades de:>>"\
        +miEntryHijo.get()+">> Palp del año:>>"+miEntryPalp.get())
    datosSalida.append(miEntryPh.get())
    datosSalida.append(miEntryHijo.get())
    datosSalida.append(miEntryPalp.get())
    datosSalida.append(miEntryEspaFase.get())
  def funcionBotonSS():
    raiz.destroy()

  botonMostrar=Button(raiz,text="Introducir Datos",command=funcionBotonMS)
  botonMostrar.pack()
  botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
  botonSalir.pack()
```

```
    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción
    return  datosSalida

def representarPobl(representar,poblB):
    #Datos
    poblacionBegin = poblB
    datosDic=representar
    x,Y1,Y2,Y3,Y4=[],[],[],[],[]

    for i in datosDic:
        #x and y must have same first dimension,
        x.append(i) #en i estan las keys, que son enteros
        Y1.append(datosDic[i][0])
        Y2.append(datosDic[i][1])
        Y3.append(datosDic[i][2])
        Y4.append(datosDic[i][3])

    eje_x=list(x)
    Y1=list(Y1)
    Y2=list(Y2)
    Y3=list(Y3)
    Y4=list(Y4)
    #crea un objeto fig de la clase Figure y de él se crean 4 instancias diferentes
    fig=plt.figure()
    #primer dibujo-ax1-, de 2 fila(2) y dos columnas(2), colocado el primero(1)
    ax1=fig.add_subplot(2,2,1)
    ax1.plot(eje_x,Y1)
    plt.ylabel('Población')
    #segundo dibujo-ax2-, de dos fila(2) y dos columnas(2), colocado el segundo(2)
```

```
  ax2=fig.add_subplot(2,2,2)
  ax2.plot(eje_x,Y2)
  plt.ylabel('Nacidos')
  # tercer dibujo-ax2-, de dos fila(2) y dos columnas(2), colocado el tercero(3)
  ax3=fig.add_subplot(2,2,3)
  ax3.plot(eje_x,Y3)
  plt.ylabel('Fallecidos')
  #4º dibujo
  ax4=fig.add_subplot(2,2,4)
  ax4.plot(eje_x,Y4)
  plt.ylabel('Fallecidos x desastres')
  #mostrar
  plt.show
  #imprime fichero
  poblacion,nacidos,muertos,muertosAP=0,0,0,0
  #tomo datos de la aplicación ph()
  fichero=open('Variaciones_poblacion.txt','w')
  fichero.write('Procedimiento monitorizar')
  fichero.write('\n'+'Información guardada en file: Variaciones_poblacion.txt')

fichero.write('\n'+format('Año',"^5")+format('Población',"^10")+format('Nacidos',"^15")+for
mat('Muertos',"^15")\
          +format('Muertos AP',"^8"))
  #convertir  formato
  for ele in x :   #x comienza en 1
    if ele == 1:

tira=('\n'+format(0,"<5")+"*"+format(poblacionBegin,"<10")+"*"+format(0,"<15")+"*"+forma
t(0,"<15")+"*"\
          +format(0,"<7"))
       fichero.write(tira)
    tiem=ele
```

```
      poblacion=Y1[ele-1]
      nacidos=Y2[ele-1]
      muertos=Y3[ele-1]
      muertosAP=Y4[ele-1]

tira=('\n'+format(tiem,"<5")+"*"+format(poblacion,"<10")+"*"+format(nacidos,"<15")+"*"+f
ormat(muertos,"<15")+"*"\
          +format(muertosAP,"<7"))
      fichero.write(tira)
   fichero.close
   os.startfile("Variaciones_poblacion.txt","print")
   return

def representarIDH(representar):
   #Datos
   datosDic=representar
   x,Y1=[],[]

   for i in datosDic:
      #x and y must have same first dimension,
      x.append(i) #en i estan las keys, que son enteros
      Y1.append(datosDic[i] )



   eje_x3=list(x)
   Y1=list(Y1)

   #crea un objeto fig de la clase Figure
   fig=plt.figure()
   #primer dibujo-ax1-, de 2 fila(2) y dos columnas(2), colocado el primero(1)
   ax1=fig.add_subplot(1,1,1)
   ax1.plot(eje_x3,Y1)
```

```
    plt.ylabel('IDH')

    plt.show

def presentaDatosSalidaDiccionario(frase):
    objeto=frase
    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -1')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Diccionario de Activiodades del ph() solicitado")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
    miEntryText.insert(INSERT,objeto )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonSS():
        raiz.destroy()
```

```
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaDatosSalida(objetoPh,frase1,frase2):
    #utilizando Tkinter, tomo dos datos (año de inicio y población inicial)
    #Se los paso a los siguientes procedimientos
    #podrian ampliarse los campos que se piden, habría más parametros de DISEÑO
    getdatos=objetoPh   #getDatos() del objeto dniPh1
    objeto1=frase1     #diccionario de actividades del objeto dniPh2
    objeto2=frase2      #tupla con los palp resultado de un select

    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -10')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Actividades del ph solicitado")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
```

```
    miEntryText.insert(INSERT,objeto1 )
    miEntryText.insert(INSERT,'++++++++++++++PALP del año
solicitado++++++++++++++++++++++++++++++')
    miEntryText.insert(INSERT,objeto2 )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonMS():
        #Pongo los Entryes en la lista datosIniciales
        messagebox.showinfo("Estado del Ph solicitado:", getdatos)

    def funcionBotonSS():
        raiz.destroy()

    botonMostrar=Button(raiz,text="Estado del Ph solicitado:",command=funcionBotonMS)
    botonMostrar.pack()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaDatosSalidaHijos(frase):
    objeto=frase
    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -1')
```

```
    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Parejas con descendientes. Puede hacer scroll")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
    miEntryText.insert(INSERT,objeto )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonSS():
        raiz.destroy()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaDatosSalidaVHijos(frase):
    objeto=frase
    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
```

```
    raiz.title('Simulación Sociedad. Versión -1')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Varones con descendientes. Puede hacer scroll")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
    miEntryText.insert(INSERT,objeto )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonSS():
        raiz.destroy()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaDatosSalidaMHijos(frase):
    objeto=frase
    raiz=tk.Tk() #creo una variable
```

```
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -1')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Mujeres con descendientes. Puede hacer scroll")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
    miEntryText.insert(INSERT,objeto )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonSS():
        raiz.destroy()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaDatosSalidaSDificil(frase):
```

```
objeto=frase
raiz=tk.Tk() #creo una variable
raiz.geometry("800x500")  #ancho y alto
raiz.config(bg='yellow')
raiz.title('Simulación Sociedad. Versión -1')

#Frame
miframe=Frame() #hecho el Frame, es trasparente
miframe.pack() #empaquetar
miframe.config(bg="red")
miframe.propagate(0)
miframe.config(width="800",height="400")

miLabelA=Label(miframe, text="ph con trabajo S-dificil")
miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

miEntryText=Text(miframe,width=60,height=20)
miEntryText.grid(row=2,column=0)
miEntryText.insert(INSERT,objeto )

miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
miScrollVertical.grid(row=2,column=3,sticky="nsew")
miEntryText.config(yscrollcommand=miScrollVertical.set)

def funcionBotonSS():
   raiz.destroy()
botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
botonSalir.pack()

raiz.mainloop() #está en un bucle ejecutandose
#debe ser la ultima instrucción
```

```
def presentaDatosSalidaFallecidos(frase):
    objeto=frase
    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -1')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Relación de ph Fallecidos")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miEntryText=Text(miframe,width=60,height=20)
    miEntryText.grid(row=2,column=0)
    miEntryText.insert(INSERT,objeto )

    miScrollVertical=tk.Scrollbar(miframe,command=miEntryText.yview)
    miScrollVertical.grid(row=2,column=3,sticky="nsew")
    miEntryText.config(yscrollcommand=miScrollVertical.set)

    def funcionBotonSS():
        raiz.destroy()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()
```

```
    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def presentaPantallaMuda():

    raiz=tk.Tk() #creo una variable
    raiz.geometry("800x500")  #ancho y alto
    raiz.config(bg='yellow')
    raiz.title('Simulación Sociedad. Versión -1')

    #Frame
    miframe=Frame() #hecho el Frame, es trasparente
    miframe.pack() #empaquetar
    miframe.config(bg="red")
    miframe.propagate(0)
    miframe.config(width="800",height="400")

    miLabelA=Label(miframe, text="Relación no exhaustiva de gráficos e informaciones
siguientes")
    miLabelA.grid(row=1,column=0,sticky="w",padx=15,pady=15)

    miLabelA=Label(miframe, text="1-Población, nacimientos, fallecimientos,Pib")
    miLabelA.grid(row=2,column=0,sticky="w",padx=15,pady=15)
    miLabelA=Label(miframe, text="2-Índice de Gini")
    miLabelA.grid(row=3,column=0,sticky="w",padx=15,pady=15)
    miLabelA=Label(miframe, text="3-Índice IDH")
    miLabelA.grid(row=4,column=0,sticky="w",padx=15,pady=15)
    miLabelA=Label(miframe, text="4-Ajustes de distribuciones de probailidad de ingresos
anuales")
    miLabelA.grid(row=5,column=0,sticky="w",padx=15,pady=15)

    def funcionBotonSS():
```

```
        raiz.destroy()
    botonSalir=Button(raiz,text="Salir",command=funcionBotonSS)
    botonSalir.pack()

    raiz.mainloop() #está en un bucle ejecutandose
    #debe ser la ultima instrucción

def graficoBarras(frase):
    #En frase se le debe pasar un diccionario
    valor=frase
    class App(tk.Tk):
        def __init__(self, valor):
            super().__init__()

            self.title('Demo de Tkinter & Matplotlib ')

            # prepare data
            abscisas = valor.keys()
            ordenadas = valor.values()
            # create a figure
            figure = Figure(figsize=(12, 6), dpi=100)
            # create FigureCanvasTkAgg object
            figure_canvas = FigureCanvasTkAgg(figure, self)
            # create axes
            axes = figure.add_subplot()

            # create the barchart
            axes.bar(abscisas,ordenadas)
            axes.set_title('Paso de valor: un diccionario')
            axes.set_ylabel('Ordenadas: valor población')
            axes.set_xlabel('Abscisas: tiempo')
```

```
        figure_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
    app = App(valor)
    app.mainloop()

def graficoBarrasPIB(frase):

    #En frase se le debe pasar un diccionario
    valor=frase
    class App(tk.Tk):
        def __init__(self, valor):

            super().__init__()

            self.title('Demo de Tkinter & Matplotlib ')
            # prepare data
            abscisas = valor.keys()
            ordenadas = valor.values()
            # create a figure
            figure = Figure(figsize=(12, 6), dpi=100)
            # create FigureCanvasTkAgg object
            figure_canvas = FigureCanvasTkAgg(figure, self)
            # create axes
            axes = figure.add_subplot()

            # create the barchart
            axes.bar(abscisas,ordenadas)
            axes.set_title('Paso de valor: un diccionario')
            axes.set_ylabel('Ordenadas: PIB anual')
            axes.set_xlabel('Abscisas: tiempo')
            figure_canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
```

```
app = App(valor)
app.mainloop()
```

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 25 18:11:45 2024
ph_herramientas.py contiene aquellas funciones que son llamadas desde varios modulos
@author: invat
"""
import mysql.connector

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
   user="invat",password="jquintas49",auth_plugin="mysql_native_password")

def selectBDNoWhere(sql1):
   #hace unselect generico de una tabla
   miCursor=miConexion.cursor()
   sql= sql1
   miCursor.execute(sql)
   registro=miCursor.fetchall()
   miConexion.commit()
   miCursor.close()
   return registro

def selectBD(sql1,valor1):
   #hace un select de una tabla e incluye la clausula WHERE
   miCursor=miConexion.cursor()
   sql= sql1
   valor=valor1
   miCursor.execute(sql,valor)
   registro=miCursor.fetchall()
   miConexion.commit()
   miCursor.close()
   return registro
```

```
def insertBD(sql1,valor1):
    #insertBD.llamadas += 1

    #Crea, inserta VARIOS -many- registros en tabla
    #tambien proporciona la ultima clave AUTOMATICA añadida (tema Hijos, descendencia)
    miCursor=miConexion.cursor()
    sql= sql1
    valor=valor1
    miCursor.executemany(sql,valor)
    miCursor.execute(" select last_insert_id()")
    idDesc=miCursor.fetchall()
    idDescendiente=idDesc[0][0]
    miConexion.commit()
    miCursor.close()
    return idDescendiente

def insertUno(sql1,valor1):
    #insertBD.llamadas += 1
    #inserta UN registro
    miCursor=miConexion.cursor()
    sql= sql1
    valor=valor1
    miCursor.execute(sql,valor)
    miConexion.commit()
    miCursor.close()
    return

def updateBD(sql2,valor2):
    #el update NO precisa un Select previo
    miCursor=miConexion.cursor()
```

```
    sqlUpdate=sql2
    valorUpdate=valor2
    miCursor.executemany(sqlUpdate,valorUpdate)
    miConexion.commit()
    miCursor.close()
    return

def updateUno(sql2,valor2):
    #el update NO precisa un Select previo
    miCursor=miConexion.cursor()
    sqlUpdate=sql2
    valorUpdate=valor2
    miCursor.execute(sqlUpdate,valorUpdate)
    miConexion.commit()
    miCursor.close()
    return

def recuperarTablaActividades():
    #se cargan los 32 primeros registros de la tabla actividades
    sqlselect= "Select * from tercerbd.actividades "
    regActi = selectBDNoWhere(sqlselect)
    regis = [list(a) for a in regActi]

    return  regis

def recuperarRegistroSociedad(iteracion):
    #recupera los valores de Sociedad correspondientes al AÑO ACTUAL GLOBAL
    #considerar que la edad de cada ph NO es el AÑO ACTUAL GLOBAL
    sqlselect= "Select * from sociedad where tiempo=%s"
    valorselect= (iteracion,) #para ser tupla un valor
    registro=selectBD(sqlselect,valorselect)
```

```
    regis=registro[0]
    return regis
```

```
# -*- coding: utf-8 -*-
"""
Created on Sun Aug 11 08:21:20 2024
ph_salidas
@author: invat
"""

from collections import Counter
from io import open
import matplotlib.pyplot as plt

import mysql.connector
import numpy as np
import os
import pandas as pd
import ph_herramientas
import ph_pantallasES
from scipy.stats import shapiro
from scipy.stats import gamma
from scipy.stats import erlang
import scipy.special as sps

import warnings
import random
from math import log

warnings.filterwarnings('ignore')

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
   user="invat",password="jquintas49",auth_plugin="mysql_native_password")
```

```
def extraeDatosTotales():
   #No incluye RBU y excluye a menores de 18
   fraseIA = []
   sql2='SELECT ingrTotal FROM tercerbd.iteracionecon  where (tiempo > 0 and edad >= 18
) '
   fraseIA = ph_herramientas.selectBDNoWhere(sql2)
   listTotal =   [ a[0] for a in fraseIA]
   #print('listTotal',listTotal)
   return listTotal


def recuperarRegistroSociedadPIB():
   #recupera los valores del campo pib de Sociedad de tods los años
   dic =  {}
   sqlselect= "Select tiempo, pib from sociedad"
   registro=ph_herramientas.selectBDNoWhere(sqlselect)
   for a in registro:
      dic[a[0]] = int(a[1])
   return dic


def phTrabajoSDificil(nombrePh):
   #explora los ph() vivos (los muertos han sido borrados!) que tienen trabajo= S-dificil
   #No asigna, es para representar
   listaSDificil=[]
   for dni in nombrePh:
      objeGene=nombrePh.get(dni)
      #if (objeGene is not None):
      if objeGene.situlaboposi  == 7:
         listaSDificil.append(dni)
   return listaSDificil


def dibuEspaFase(etiquetas, valores):
```

```
    #ChatGPT
    # Creamos el gráfico
    plt.figure(figsize=(8, 5))
    plt.plot(etiquetas, valores, marker='o')
    # Añadimos títulos y etiquetas
    plt.title('Espacio de Fases de un ph(dni)')
    plt.xlabel('Intencionalidad')
    plt.ylabel('Palp')
    plt.grid()
    # Mostramos el gráfico
    plt.show()
    return


def histoVa3Va9():
    #dibuja histogramas de Va3 y Va9
    def extraeVa3Va9():
        #sale un perfecto diccionario de listas de ingresos mayores que cero y en edad de
trabajar
        sql2='SELECT sumava3, sumava9 FROM tercerbd.iteracionecon  where tiempo > 0 '
        fraseVa3Va9=ph_herramientas.selectBDNoWhere(sql2)
        return fraseVa3Va9


    sumaVa3 = []
    sumaVa9 = []
    reg = extraeVa3Va9()
    for item in reg:
        sumaVa3.append(item[0])
        sumaVa9.append(item[1])


    plt.hist(sumaVa3, 50, density=True)
    plt.title("Histograma  Va3")
```

```
    plt.show()


    plt.hist(sumaVa9, 50, density=True, color = 'r')
    plt.title("Histograma  Va9")
    plt.show()
    return


def esGaussiana():


    def extraeDatosG():
        #sale uns lista con TODOS los datos  ingrxTrab; no se distingue por iteración
        fraseIA = []
        lista = []
        sql2='SELECT ingrxTrab FROM tercerbd.iteracionecon  where (tiempo > 0 and (18 <= edad <65)  and ingrxTrab > 0) order by ingrxTrab '
        fraseIA=ph_herramientas.selectBDNoWhere(sql2)
        #frase IA es una lista de tuplas: iteracion,ingresos anual (ojo por trabajo) de esa persona
        for i in fraseIA:
            lista.append(i[0])
        return lista


    def extraeDatosAnual():
        #frase IA es una lista de tuplas: iteracion,ingresos anual (ojo por trabajo: ingrxTrab) de esa persona
        #forma un diccionario de listas. A cada lista se le aplica el test de Shapiro
        fraseIA = []
        lista = []
        lKey = []
        lValue = []
        dic = {}
```

```
    #sale un perfecto diccionario de listas de ingresos mayores que cero y en edad de
trabajar
    sql2='SELECT tiempo, ingrxtrab FROM tercerbd.iteracionecon  where tiempo > 0  and
ingrxTrab > 0 order by tiempo, ingrxTrab '
    fraseIA=ph_herramientas.selectBDNoWhere(sql2)
    for a in fraseIA:
      lKey.append(a[0])
      lValue.append(a[1])
    #tenemos la lista de tuplas como dos listas
    iterA = iter(lKey)
    siguienteA = next(iterA)
    iterDato = iter(lValue)
    siguienteDato = next(iterDato)
    lista = []
    for item in lKey:
      try:
        siguienteA = next(iterA)

        lista.append(siguienteDato)
        if item != siguienteA:

          dic[item] = lista
          lista = []
        else:

          pass
        siguienteDato = next(iterDato)
      except  StopIteration:
        lista.append(siguienteDato)
        dic[item] = lista

    return dic
```

```
  def ingresosEsGaussiana(valores):
    #Diccionario a lista
    ssN=0
    nnN=0
    valores = valores
    # Shapiro-Wilk
    stat, p = shapiro(valores)
    # Interpretación
    alpha = 0.05
    if p > alpha:
      ssN +=1
    else:
      nnN +=1
    return ssN,nnN

  def esGaussianaGT():
    #Gran Total: todos los ingrxTrab de todos los ph() en todas las iteraciones
    lista = []
    #datos x años
    valores = extraeDatosG()
    #veamos si ingresos sigue una distribución gaussiana
    r1 = sorted(valores)
    for a in r1:
      #ingresos.-Divido adaptar datos a escala
      lista.append(round(a/10_000,2))
    #test de normalidad de  Sapiro
    ssN,nnN=ingresosEsGaussiana(lista)


print('================================================================
======')
```

```
      print('La muestra parece Gaussiana o Normal, en el siguiente núm de ocasiones:
',ssN)
      print('La muestra NO parece Gaussiana o Normal:  ', nnN,' veces')
      return lista

   def esGaussianaAnual(iteracion,valores):
      #Anual: todos los ingrxTrab de todos los ph() en cada iteracion
      lista = []
      #datos x años
      iteracion = iteracion
      valores = valores
      #veamos si ingresos sigue una distribución gaussiana
      r1 = sorted(valores)
      for a in r1:
         #ingresos.-Divido adaptar datos a escala
         lista.append(round(a/10_000,2))
      #test de normalidad de  Sapiro
      ssN,nnN=ingresosEsGaussiana(lista)

      print('=======================Iteración:
',iteracion,'=========================')
      print('La muestra parece Gaussiana o Normal, en el siguiente núm de ocasiones:
',ssN)
      print('La muestra NO parece Gaussiana o Normal:  ', nnN,' veces')
      return lista



   #MAIN esGaussiana
   print('Se consideran todos los ingrxTrab de todos los ph() en todas las iteraciones: Gran
Total')
   lista = esGaussianaGT()
```

```
  plt.hist(lista, 50, density=True)
  plt.title("Histograma  ingrxTrab: Gran Total")
  plt.show()
  print('Se genera un gráfico por cada iteración: todos los ingrxTrab de todos los ph()')
  dic = extraeDatosAnual()
  print(dic.keys())
  for item in dic:
    lista = esGaussianaAnual(item, dic[item])
    plt.hist(lista, 50, density=True)
    plt.title("Histograma  ingrxTrab por iteracion: ")
    plt.show()
  return


def dibujoGammasObtenidas():
  #tomado de Gamma-vueltas.py
  print(' Va a dibujarse las distribuciones Gamma que corresponden a los valores')
  print (' de Alfa y Beta calculados y almacenados en la tabla Sociedad')
  print('En el punto anterior, el histograma se calcula por random coherente con Alfas y
Betas')
  print('Se toman los valores de x generados y se ajusta una Gamma, obteniendose Alfas y
Betas')
  print('El proceso parece coherente')
  print('No ocurre lo mismo cuando se obtienen los valores de x de los ingresos generados
por HOSS (V.-7)')



  def dibujoGammas4Casos():


    def extraeDatos():#ingresos anuales de cada ph()
      #sale un perfecto diccionario de listas de ingresos mayores que cero y en edad de
trabajar
      sql2='SELECT tiempo, ingrxTrab FROM tercerbd.iteracionecon  where (tiempo > 0
and (18 <= edad <65) ) and ingrxTrab > 0 order by tiempo, ingrxTrab '
```

```
    fraseIA=ph_herramientas.selectBDNoWhere(sql2)
    dic=listaTuplAListEspecial(fraseIA)
    return dic

def listaTuplAListEspecial(lista0):
    #convierto la lista de tuplas en diccionario de listas de valores mayor que cero
    lV4 = []
    dic = {}
    clave = 1
    reg=lista0
    regis = [list(a) for a in reg]
    regis.append(['*','*'])
    for elemento in regis:
      if elemento[0] == clave:
        elemM0 = (float(elemento[1])/1_000)
        #print(elemM0)
        if elemM0 >0:
          lV4.append (elemM0)
      else:
        dic[clave] = lV4
        clave +=1
        lV4 = []
    return dic

#MAIN

print(' Va a dibujarse las distribuciones Gamma que corresponden a los valores')
print (' de Alfa y Beta calculados con Datos Iniciales y almacenados en la tabla Sociedad')

alfas = []
```

```
    betas = []
    shape1 = []
    scale1 = []
    s = []
    anyo = 0
    #extrae alfas y betas de Datos Iniciales
    sql1='Select alfa, idh FROM sociedad'
    regis=ph_herramientas.selectBDNoWhere(sql1)
    for a in regis:
      alfas.append(a[0])
      betas.append(round(float(a[1]),2))
    alfas.remove(0)
    betas.remove(0)

    itera = iter(betas)  # creamos el iterador

    for alfa in alfas:
      anyo +=1
      beta = next(itera)
      #para dibujar Gamma: random su histograma
      serie = np.random.gamma(alfa, beta,1000)   #1000 valores random

      #dibujo
      count, bins, ignored = plt.hist(serie, 50, density=True) #histoigrama
      y = bins**(alfa-1) * ((np.exp(-bins/beta))/ (sps.gamma(alfa) * beta**alfa))   #formulka
distribución Gamma Excel
      plt.title('1º-Histograma año: '+ str(anyo) + '.Ingresos random. Alfa/beta calculadas
Datos Iniciales')
      plt.plot(bins, y, linewidth=2, color='r')  # distribución Gamma
      plt.show()
    #Siguiente Grafico; alfas, betas de Datos iniciales y serie datos de ingrxTrab en s
procedentes de dicc
```

```
    dicc = extraeDatos()
    for item in dicc:
        s.append(dicc[item]) #s: ingrxTrab


    #creamos el iterador y el siguiente elemento
    itera1 = iter(s)
    iteraBeta = iter(betas)
    anyo = 0
    for alfa in alfas:
        anyo +=1
        beta = next(iteraBeta)
        s = next(itera1) #Lista con los ingresos anuales de cada ph()
        #s debe see una lista
        #dibuja histograma de ingresos y gamma de la calculadas (no ajustadas) segun
datos iniciales
        count, bins, ignored = plt.hist(s, 50, density=True)
        y = bins**(alfa-1) * ((np.exp(-bins/beta))/  (sps.gamma(alfa) * beta**alfa)) # la formula
de la distribución Gamma/Excel
        plt.title("2º-Histograma año: "+ str(anyo) + ".Ingresos reales. Alfa/beta calculadas
datos iniciales")
        plt.plot(bins, y, linewidth=2, color='r')
        plt.show()


    #Tomemos alfas y betas obtenidas a partir de (.fit) de los ingresos: ingrxTrab
    anyo = 0
    borraDicc = []
    serie1 = pd.Series(dicc)
    for item in serie1:
        anyo +=1
        #con la misma s generada: obtiene alfa y beta ajustadas (.fit)
        serie = pd.Series(item)
        print('pd.item >>>>>>>>>>>>>>>>',serie)
```

```
    media = serie.mean()
    minimo = serie.min()
    maximo = serie.max()
    std = serie.std()

    # Imprimir los estadísticos básicos
    print("Estadísticos básicos:")
    print("Media:", media)
    print("Mínimo:", minimo)
    print("Máximo:", maximo)
    print("Desviación estándar:", std)
    #params = gamma.fit(serie) ; params es tupla: [0] es shape y [2] es scale
    shape2, loc2, scale2 = gamma.fit(serie)  # deberia alfa==shape2 y beta==scale2
    if shape2 > 1:
      shape1.append(round(shape2,2))
      scale1.append(round(scale2,2))
    else:
      #los valores de shape1 <= 0 dan un ajuste malo, creo
      borraDicc.append(anyo)
print(f'Valores del AJUSTE  Alfa  {shape1} ---- Beta  {scale1}' )
for item in borraDicc:
    print('Años eliminados: ', item)
    del dicc[item]

iteraKey = iter(dicc)
# se repite el proceso de dibujo con las nuevas Alfa y Beta
itera = iter(scale1)
for alfa in shape1:

    sig = next(itera)
```

```
        beta = sig
        s = np.random.gamma(alfa, beta,1000)
        diccKey = next(iteraKey)
        #valores de x: random adecuado a Alfa y Beta del AJUSTE
        #una lista de 1000 valores float
        count, bins, ignored = plt.hist(s, 50, density=True)
        y = bins**(alfa-1) * ((np.exp(-bins/beta))/ (sps.gamma(alfa) * beta**alfa))
        plt.title("3º-Histograma  ingresos random y alfa/beta del ajuste gamma.fit de ingrxTrab. Año: " + str(diccKey) )
        plt.plot(bins, y, linewidth=2, color='r')
        plt.show()
    #usando las alfa y betas procedentes de (.fit) se usan datos de sserie1 (ingrxTrab) en vez de random
    serie2 = pd.Series(dicc)
    itera2 = iter(serie2)
    iteraBeta = iter(scale1)
    for alfa in shape1:
        beta = next(iteraBeta)

        try:
            s = next(itera2)
        except:
            StopIteration
        #s debe see una lista
        #dibuja histograma de ingresos y gamma de la calculadas (no ajustadas) segun datos iniciales
        count, bins, ignored = plt.hist(s, 50, density=True)
        y = bins**(alfa-1) * ((np.exp(-bins/beta))/  (sps.gamma(alfa) * beta**alfa)) # la formula de la distribución Gamma/Excel
        plt.title("4º-Histograma  ingresos reales y alfa/beta del ajuste gamma.fit de ingrxTrab")
        plt.plot(bins, y, linewidth=2, color='r')
        plt.show()
```

```
        #time.sleep(1) #pausa de un segundo


    #Erlang; todos los ingrxTrab independientemente del año
    ingresos = s
    #histograma


    #Crear el histograma:
    plt.hist(ingresos, bins=15, density=True, alpha=0.6, color='g', label='Histograma de
Ingresos')


    # Estimar los parámetros de la distribución de Erlang
    k, loc, scale = erlang.fit(ingresos, floc=0)  # Ajustamos loc a 0


    # Crear un rango de valores para el gráfico
    x = np.linspace(min(ingresos), max(ingresos), 100)
    pdf = erlang.pdf(x, k, loc=loc, scale=scale)


    # Graficar la distribución de Erlang ajustada
    plt.plot(x, pdf, 'r-', lw=2, label='Ajuste Erlang')


    #Personalizar y mostrar el gráfico:
    plt.title('Histograma de Ingresos y Ajuste de Erlang')
    plt.xlabel('Ingresos')
    plt.ylabel('Densidad')
    plt.legend()
    plt.grid()
    plt.show()


    return


  dibujoGammas4Casos()
```

```
    return


def imprimirSociedad():
    #Imprime parte de  tabla Sociedad
    pib,ingresos,idh,gini,alfa=0.0,0.0,0.0,0.0,0.0
    #tomo datos de la aplicación ph()
    sql1='Select tiempo,poblacion,pib,ingrTotal,idh,gini,alfa FROM sociedad'
    regis=ph_herramientas.selectBDNoWhere(sql1)
    r1=[]
    fichero=open('pibIngIDHGini.txt','w')
    fichero.write('Procedimiento imprimirSociedad')
    fichero.write('\n'+'Información guardada en file: pibIngIDHGini.txt')

fichero.write('\n'+format('Año','^5')+format('Población','^10')+format('PIB','^15')+format('I
ngresos','^15')\
            +format('Gini(G)','^8')+format('Alfa(G*20) ','^10')+format('IDH','^8')+format('Beta
','^8'))
    #convertir  formato
    for a in regis:
        r1.append(a[0])
        r1.append(a[1])
        r1.append(a[2])
        r1.append(a[3])
        r1.append(a[4])
        r1.append(a[5])
        r1.append(a[6])
        tiem=a[0]
        poblacion= a[1]
        if int(poblacion) < 20:
            break
        pib=a[2]
```

```
        ingresos=a[3]
        idh=a[4]
        beta=idh #IDH está cerca de 1; igualdad implicaria trabajar con Gamma
estandarizada
        gini=a[5]
        alfa=a[6]

tira='\n'+format(tiem,"<5")+"*"+format(poblacion,"<10")+"*"+format(pib,"<15")+"*"+format
(ingresos,"<15")+"*"\
            +format(gini,"<7")+"*"+format(alfa,"<7")+"*"+format(idh,"<7")+"*"+format(beta,"<7")


        fichero.write(tira)
    fichero.close
    os.startfile("pibIngIDHGini.txt","print")
    return regis


def listaTuplasAListasSE(lista0):
    #convierto la lista de tuplas en lista de listas
    reg=lista0
    regis = [list(a) for a in reg]
    return regis
def codificar(lista0):
    #cada elemento de lista2  es un str
    lista1=lista0
    lista2=[]
    lista2 = [str(item)  for item in lista1]
    return lista2


def parejaHijos():
    #recorrer la tabla nacer y ver cuantos hijos tiene cada pareja que tiene al menos, un hijo
    reg=[]
    sql1='Select idPadre,idMadre from nacer where idPadre != -1 and idMadre != -1'
```

```
    reg=ph_herramientas.selectBDNoWhere(sql1)
    listaListas=listaTuplasAListasSE(reg)
    lista=codificar(listaListas)
    ocurrencia=Counter(lista)
    #Ocurrencia: es diccionario la lista [padre,madre] es clave y el valor es num hijos
    return ocurrencia

def varonesHijos():
     #recorrer la tabla nacer y ver cuantos hijos tiene cada varón
     reg=[]
     ocurrencia={}
     sql1='Select idPadre from nacer where idPadre!= -1 and idMadre!= -1'
     reg=ph_herramientas.selectBDNoWhere(sql1)
     #convierto la lista de tuplas en lista de listas
     listaListas=listaTuplasAListasSE(reg)
     lista=codificar(listaListas)
     ocurrencia=Counter(lista)
     return ocurrencia

def mujeresHijos():
    #recorrer la tabla nacer y ver cuantos hijos tiene cada mujer
    reg=[]
    ocurrencia={}
    sql1='Select idMadre from nacer where idPadre!= -1 and idMadre!= -1'
    reg=ph_herramientas.selectBDNoWhere(sql1)
    #convierto la lista de tuplas en lista de listas
    listaListas=listaTuplasAListasSE(reg)
    lista=codificar(listaListas)
    ocurrencia=Counter(lista)
    return ocurrencia
```

```
def imprimirPH(ph):

    #los datos de un ph() tomados de MySQL
    #tabla nacer

    fichero=open('imprimirPH.txt','w')
    fichero.write('Procedimiento imprimirPH.py')
    fichero.write('\n'+'Información guardada en file: imprimirPH.txt' + '\n')
    tira = '\n'+'>>>>>>>>>>>>>>>>>>>>>>>>>>>Tabla
nacer>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'+ '\n'

fichero.write('\n'+'*DNI*Edad*IniAbs*Sexo*Falle*Intenc*IdFam*IdP*IdE*RieT*IdP*IdM*RT
ole**CoI**Resili*Poten*AtraPers*'+'\n')

    tira=''
    sql1='Select * FROM nacer where id=%s'
    valor1=(ph,)
    registro=ph_herramientas.selectBD(sql1,valor1)
    regis = registro[0]
    for item in regis:
        tira = tira +str(item) +'*'

    fichero.write(tira)
    tira = ''
    #Linea de separación
    tira =
'\n'+'>>>>>>>>>>>>>>>>>>>>>>>>>>>Iteración>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
+ '\n'
    fichero.write(tira)

    #tabla iteracion
    tira=''
```

```
   fichero.write('\n'+'*DNI*Tiempo*Actividad**Edad va0*
va1*va2*va3*va4*va5*va6*va7*va8*va9*'+'\n')



   sql1='Select * FROM iteracion where dni=%s and edad > 18 order by tiempo,actividad'
   valor1=(ph,)
   regis=ph_herramientas.selectBD(sql1,valor1)
   for item in regis:
      tira = str(item)
      tira='\n'+tira
      fichero.write(tira)
      tira=''


   #Linea de separación
   tira =
'\n'+'>>>>>>>>>>>>>>>>>>>>>>Familia>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'+
'\n'
   fichero.write(tira)
   #tabla organiza_miembros - Familia
   tira=''
   sql1="SELECT * FROM tercerbd.organiza_miembrosfami where  id_nacer = %s;"
   valor1=(ph,)
   regis=ph_herramientas.selectBD(sql1,valor1)
   for item in regis:
      tira = str(item)
      tira='\n'+tira
      fichero.write(tira)
      tira=''
   #Linea de separación
   tira =
'\n'+'>>>>>>>>>>>>>>>>>>>>>>Empresa>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>'
+ '\n'
   fichero.write(tira)
```

```
    #tabla organiza_miembros - Organizaciones (sea  software, industria...)
    tira = ''
    sql2="SELECT * FROM tercerbd.organiza_miembros where  id_nacer=%s ;"
    valor2=(ph,)
    regis=ph_herramientas.selectBD(sql2,valor2)
    for i in regis:
        tira = str(i)
        tira = '\n' + tira
        fichero.write(tira)
        tira=''

    fichero.close
    #La siguiente linea imprime por impresora: ojo que son varias A4!!
    os.startfile("imprimirPH.txt","print")

    return

def monitorizar(poblCont,poblacionBegin):

    def moniCI():
        #APLICACION al 'Histograma Coeficiente Intelectual población'
        pobl=[]
        sql1='SELECT coefiInte FROM tercerbd.nacer;'
        pobl = ph_herramientas.selectBDNoWhere(sql1)
        numeros =  sorted([float(a[0]) for a in pobl])
        # creamos un histograma y pintamos
        plt.hist(numeros, bins=10, density=True)
        plt.xlabel("Valores CI")
        plt.ylabel("Frecuencia")
        plt.title("Histograma Coeficiente Intelectual población")
        plt.show()
```

```
    return
  def moniResi():
    #APLICACION al 'Histograma Resiliencia población'
    pobl=[]
    sql1='SELECT resiliencia FROM tercerbd.nacer;'
    pobl = ph_herramientas.selectBDNoWhere(sql1)
    numeros =  sorted([float(a[0]) for a in pobl])
    # creamos un histograma y pintamos
    plt.hist(numeros, bins=10, density=True)
    plt.xlabel("Valores resiliencia")
    plt.ylabel("Frecuencia")
    plt.title("Histograma  Resiliencia población")
    plt.show()
    return

  def moniPoten():
    #Aplicacion a bins literales
    'Histograma Potencial población'
    #grupos=sorted(['0-Bajo','1-Medio','2-Altos','3-Malto'])
    #print(grupos)
    sql1='SELECT potencial FROM tercerbd.nacer;'
    pobl = ph_herramientas.selectBDNoWhere(sql1)
    numeros =  ([(a[0]) for a in pobl])
    plt.hist(numeros, bins=4, density=True)
    # Configurar el histograma
    plt.xlabel('Potencial: Grupos')
    plt.ylabel('Valores')
    plt.title('Histograma Potencial población')
    # Mostrar histograma
    plt.show()
    return
```

```
    def moniPobl():
        ph_pantallasES.representarPobl(poblCont,poblacionBegin)
        print(poblCont)
        return
    moniCI()
    moniResi()
    moniPoten()
    moniPobl()
    return

def volcadoPh(nombrePh):
    #comprueba escritura en todos los vivos  ph() de POO de seis datos economicos
    for i in nombrePh:
        objeGene = nombrePh.get(i)
        print(objeGene.palp) #en realidad los ahorros
        print(objeGene.gastoAnyo)
        print(objeGene.ingrxTrabB)
        print(objeGene.plusProductividad)
        print(objeGene.ingrxTrab)
        print(objeGene.ingrxRBU)
        print('>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>')
    return

# Función  Llamada desde ph_principal
def tratamientoDatos(nombrePh, datosReprPoblPib,poblCont,poblacionBegin):
        #Utilizado Ingresos Anuales para ver si es Gaussiana y para graficos Gamma
        #extraeDatos,
listaTuplAListEspecial,ingresosAnualesEsGaussiana,ingAnualHistoGamma,
        # y para acabar: MAIN de Ingresos Anuales, en Tratamiento de datos
        #Por cada iteracion un grafico de Ingresos Anuales (IA)
```

```
  nombrePh= nombrePh
  datosReprPoblPib = datosReprPoblPib
  poblCont = poblCont
  def extraeDatos():
    fraseIA = []
    #sale un perfecto diccionario de listas de ingresos mayores que cero y en edad de
trabajar
    sql2='SELECT tiempo, ingrxTrab FROM tercerbd.iteracionecon  where (tiempo > 0 and
(18 <= edad <65) ) and ingrxTrab > 0 order by tiempo, ingrxTrab '
    fraseIA=ph_herramientas.selectBDNoWhere(sql2)
    dic=listaTuplAListEspecial(fraseIA)
    return dic

  def listaTuplAListEspecial(lista0):
    #convierto la lista de tuplas en diccionario de listas de valores mayor que cero
    lV4 = []
    dic = {}
    clave = 1
    reg=lista0
    regis = [list(a) for a in reg]
    regis.append(['*','*'])
    for elemento in regis:
      if elemento[0] == clave:
        elemM0 = float(elemento[1])/10_000
        if elemM0 >0:
          lV4.append (elemM0)
      else:
        dic[clave] = lV4
        clave +=1
        lV4 = []
    return dic
```

```
  def diccionarioUltimo(dniPh2):
    sql = 'SELECT max(tiempo) FROM tercerbd.iteracionecon where dni = %s;'
    valor = (dniPh2,)
    frase=ph_herramientas.selectBD(sql,valor)
    frase = frase[0][0]

    sql1='Select * from iteracionecon where dni=%s and tiempo=%s'
    valor1=(dniPh2,frase)
    frase1=ph_herramientas.selectBD(sql1,valor1)
    return frase1

  def diccEspaFase(dniPh3):
    sql1='Select intenciona,palp from iteracionecon where dni=%s order by tiempo'
    valor1=(dniPh3,)
    frase3=ph_herramientas.selectBD(sql1,valor1)

    return frase3

  def palpVivosAnyo(anyo):
    sql2="Select sum(palp) from iteracionecon where tiempo = %s  "
    valor2=(tiempoSolicitado,) #ha de ser una tupla
    frase2=ph_herramientas.selectBD(sql2,valor2)

    frase2 = (frase2[0][0])
    return frase2

  def ingresosAnualesEsGaussiana(valores):
    #Diccionario a lista
    ssN=0
    nnN=0
    valores = valores
```

```
    # Shapiro-Wilk
    stat, p = shapiro(valores)
    # Interpretación
    alpha = 0.05
    if p > alpha:
      #print ('La muestra parece Gaussiana o Normal (no se rechaza la hipótesis nula H0)')
      ssN +=1
    else:
      #print ('La muestra NO parece Gaussiana o Normal(se rechaza la hipótesis nula H0')
      nnN +=1
    return ssN,nnN

  def HAlfaBetaFichero():
    #Por cada iteracion un grafico de Ingresos Anuales (IA)
    dic = {}
    i = 1
    #datos ingrxTrab x años
    dic = extraeDatos() #extrae ingrxTrab por cada iteracion
    #claveMax = valorMax[0][0]

    # Creamos un archivo PDF con PdfPages
    #os.chdir('C:/Users/invat/MisDocumentos/A_ProyectoPython/Proyecto Python ph-7')

    regSociedad = imprimirSociedad()
    print('regSociedad--',regSociedad)
    garbage = open('garbage.txt','w')
    fichero = open ('HIngrAnyo.txt','w')
    fichero.write('\n'+'Información guardada en file: HIngrAnyo.txt')
    fichero.write('\n'+'Iteración--Alfa,beta de Ajuste(fit)--Alfa,beta usando Datos Iniciales')

    for i in dic:
```

```
tira = ''
lista = []
shape = 0.00
scale = 0.00
listaValues = (dic[i])
listaValues = sorted(listaValues)
r1 = listaValues
for a in r1:
   #ingresos.-Divido adaptar datos a escala
   if a > 0:
      #lista.append(round(float(a)/10_000,2))
      listaBis = round(log(a),2)
      lista.append(listaBis)
# Convertir el diccionario en una serie de pandas
try:
   serie = pd.Series(lista,dtype=float )
except ValueError:
   continue
#ChatGPT
# 2. Ajustar una distribución gamma a los datos

try:
   shape, loc, scale = gamma.fit(serie)  # floc=0---Forzar loc a 0 para un mejor ajuste
except  ValueError :
  continue
#para el fichero de texto
alfa = str(round(shape,2))
beta = str(round(scale,2))

if regSociedad[i][6] != None:
```

```
            alfa2 = str(round(regSociedad[i][6],2))
            beta2 = str(round(regSociedad[i][4],2))
        else:
            alfa2 = 'AAA'
            beta2 = 'BBB'
        alfa22 = str(round((regSociedad[i][6]/20),2) )




        tira='\n'+format(str(i),"<5")+" ** "+format(alfa,"<5")+" ** "+format(beta,"<5")+"
********* "+format(alfa2,"<5")+"("\
              +format(alfa22,"<5") +") **"+format(beta2,"<5")


        print('TIRA--------',tira)
        fichero.write ( tira )
        i += 1



    fichero.close()
    os.startfile('HIngrAnyo.txt','print')
    #os.remove('HIngrAnyoAlfaBeta.txt')


    return




#MAIN tratamientoDatos()

#petición de datos para hacer algunas Salidas. Utilizo el mismo mecanismo en todas
```

```
    datosSalida=ph_pantallasES.capturaDatosSalida()
    #el INT  asegura de pasar en el get un entero
    dniPh1=int(datosSalida[0]) #Elija un 'ph' válido para conocer su estado
    dniPh2=int(datosSalida[1]) #lija un 'ph' válido para conocer su último estado de
Actividades
    tiempoSolicitado=int(datosSalida[2]) #Para un año concreto, ¿cual es el palp de los
vivos?"
    dniPh3 = int(datosSalida[3])
    #Respuesta segunda pregunta
    objeto=nombrePh.get(dniPh1)
    if objeto != None:
        objetoPh=objeto.getDatos()
        imprimirPH(dniPh1)
    else:
        objetoPh='Ha fallecido'

    #obtenemos el ULTIMO diccionario del ph solicitado:dniPh2
    frase1 = diccionarioUltimo(dniPh2)
    ph_pantallasES.presentaDatosSalidaDiccionario(frase1)
    if objeto != None:
        print('Actividades de ph() '+str(dniPh2),objeto.miDicPh)

    #obtenemos el palp anual de los vivos
    frase2 = palpVivosAnyo(tiempoSolicitado)
    ph_pantallasES.presentaDatosSalida(objetoPh,frase1, frase2)

    #espacio de fases del dniPh3
    for i in range(1,11):
        azar = random.randint(1,dniPh3)
        etiquetas = []
        valores = []
        dniPh3 = dniPh3 if i == 1 else  azar
```

```
    frase3 = diccEspaFase(dniPh3) #frase3 es un diccionario del tipo intenciona:palp
    for item in frase3:
        etiquetas.append ( (item[0]))
        valores.append ( (item[1]))
    dibuEspaFase(etiquetas,valores)


#otras recuperaciones NO solicitadas.
#forma de operar: def calcula la frase y pantalla la imprime
#num hijos por la misma pareja
frase=parejaHijos()
ph_pantallasES.presentaDatosSalidaHijos(frase)
#num hijos por varón
frasev=varonesHijos()
ph_pantallasES.presentaDatosSalidaVHijos(frasev)
#num hijos por mujer
frasem=mujeresHijos()
ph_pantallasES.presentaDatosSalidaMHijos(frasem)
#quienes han logrado trabajo S-dificil
fraseSDificil=phTrabajoSDificil(nombrePh)
ph_pantallasES.presentaDatosSalidaSDificil(fraseSDificil)
#relacion de fallecidos
sql1="SELECT id,edad FROM tercerbd.nacer where fallecido = 'S' order by id"
record=ph_herramientas.selectBDNoWhere(sql1)
ph_pantallasES.presentaDatosSalidaFallecidos(record)
ph_pantallasES.presentaPantallaMuda()
#otras informaciones NO solicitadas
#preparo para hacer grafico barras poblacion solamente
datos={}
for a in datosReprPoblPib:
    datos[a]=datosReprPoblPib[a][0]
ph_pantallasES.graficoBarras(datos)
```

```
    #preparo para hacer grafico barras PIB solamente
    diccionario = {}
    diccionario = recuperarRegistroSociedadPIB()
    print(diccionario)
    ph_pantallasES.graficoBarrasPIB(diccionario)
    #Monitorizar CI, Resiliencia...de la población
    monitorizar(poblCont,poblacionBegin)
    #representar IDH
    representar={}
    sql1='SELECT tiempo, idh FROM tercerbd.sociedad order by tiempo'
    fraseIDH=ph_herramientas.selectBDNoWhere(sql1)
    fraseIDH1=listaTuplasAListasSE(fraseIDH)
    for i in fraseIDH1:
        representar[i[0]]=i[1]
    ph_pantallasES.representarIDH(representar)
    #dibuja histogramas de Va3 y Va9
    histoVa3Va9()
    # mira si los ingresos anuales o totales es gaussiana
    esGaussiana()

    ##imprime  de tabla sociedad, relación del alfas beta, x datos iniciales, y alfas,beta por
ajuste de ingresos
    dibujoGammasObtenidas()
    #'Iteración--Alfa,beta de Ajuste(fit)--Alfa,beta usando Datos Iniciales'
    HAlfaBetaFichero()
    #comprueba escritura en todos los VIVOS  ph() de POO de seis datos economicos
    volcadoPh(nombrePh)
    return
```

```
# -*- coding: utf-8 -*-
"""
Created on 24/6/24
ph-seleDist
Tomado del artículo 'Ajuste y selección de distribuciones con Python' de Joaquín Amat Rodrigo
Realizo minimas y escasas adaptaciones para cargar y preparar mis datos
@author: invat
"""

import matplotlib.pyplot as plt
import mysql.connector
import numpy as np
from scipy import stats
import ph_herramientas
#from ph_herramientas import selectBDNoWhere
import pandas as pd
import inspect
import warnings

warnings.filterwarnings('ignore')

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
   user="invat",password="jquintas49",auth_plugin="mysql_native_password")

global resultados



def datosIngrGast():
   #sql1 = SELECT tiempo, sum(va4), sum(va5) FROM tercerbd.iteracionecon group by tiempo
   #sql1='Select ingrxTrab,gastoanyo FROM tercerbd.iteracionecon where tiempo > 0'
```

```
    sql1 = 'SELECT ingrxTrab, gastoanyo FROM tercerbd.iteracionecon  where (tiempo > 0
and (18 <= edad <65) ) and ingrxTrab > 0 order by tiempo, ingrxTrab '
    regis = ph_herramientas.selectBDNoWhere(sql1)
    del regis[0]
    r1 = []
    r2 = []
    for a in regis:
        #ingresos / gastos.-Divido adaptar datos a escala
        r1.append(round(float(a[0])/10_000,2))
        #gastos.- Divido
        r2.append(round(float(a[1])/10_000,2))
    return r1,r2


def seleccionar_distribuciones(familia='realplus', verbose=True):
    '''
    Esta función selecciona un subconjunto de las distribuciones disponibles
    en scipy.stats

    Parameters
    ----------
    familia : {'realall', 'realline', 'realplus', 'real0to1', 'discreta'}
        realall: distribuciones de la familia `realline` + `realplus`
        realline: distribuciones continuas en el dominio (-inf, +inf)
        realplus: distribuciones continuas en el dominio [0, +inf)
        real0to1: distribuciones continuas en el dominio [0,1]
        discreta: distribuciones discretas

    verbose : bool
        Si se muestra información de las distribuciones seleccionadas
        (the default `True`).
```

```
  Returns
  -------
  distribuciones: list
     listado con las distribuciones (los objetos) seleccionados.

  Raises
  ------
  Exception
     Si `familia` es distinto de 'realall', 'realline', 'realplus', 'real0to1',
     o 'discreta'.

  Notes
  -----
     Las distribuciones levy_stable y vonmises han sido excluidas por el momento.

  '''

  distribuciones = [getattr(stats,d) for d in dir(stats) \
              if isinstance(getattr(stats,d), (stats.rv_continuous, stats.rv_discrete))]

  exclusiones = ['levy_stable', 'vonmises', 'studentized_range','chi2']
  distribuciones = [dist for dist in distribuciones if dist.name not in exclusiones]
  #Delimito desde cero
  dominios = {
     'realall' : [-np.inf, np.inf],
     'realline': [np.inf,np.inf],
     'realplus': [0, np.inf],
     'real0to1': [0, 1],
     'discreta': [None, None],
  }
```

```
  distribucion = []
  tipo = []
  dominio_inf = []
  dominio_sup = []

  for dist in distribuciones:
    distribucion.append(dist.name)
    tipo.append(np.where(isinstance(dist, stats.rv_continuous), 'continua', 'discreta'))
    dominio_inf.append(dist.a)
    dominio_sup.append(dist.b)

  info_distribuciones = pd.DataFrame({
                  'distribucion': distribucion,
                  'tipo': tipo,
                  'dominio_inf': dominio_inf,
                  'dominio_sup': dominio_sup
                 })

  info_distribuciones = info_distribuciones \
                  .sort_values(by=['dominio_inf', 'dominio_sup'])\
                  .reset_index(drop=True)

  if familia in ['realall', 'realline', 'realplus', 'real0to1']:
    info_distribuciones = info_distribuciones[info_distribuciones['tipo']=='continua']
    condicion = (info_distribuciones['dominio_inf'] == dominios[familia][0]) & \
            (info_distribuciones['dominio_sup'] == dominios[familia][1])
    info_distribuciones = info_distribuciones[condicion].reset_index(drop=True)

  if familia in ['discreta']:
    info_distribuciones = info_distribuciones[info_distribuciones['tipo']=='discreta']
```

```
    seleccion = [dist for dist in distribuciones \
                if dist.name in info_distribuciones['distribucion'].values]



    if verbose:
        print("-----------------------------------------------------")
        print("      Distribuciones seleccionadas               ")
        print("-----------------------------------------------------")
        with pd.option_context('display.max_rows', None, 'display.max_columns', None):
            print(info_distribuciones)

    return seleccion



def comparar_distribuciones(texto,x, familia='realplus', ordenar='aic', verbose=True):
    '''
    Esta función selecciona y ajusta un subconjunto de las distribuciones
    disponibles en scipy.stats. Para cada distribución calcula los valores de
    Log Likelihood, AIC y BIC.

    Parameters
    ----------
    x : array_like
        datos con los que ajustar la distribución.

    familia : {'realall', 'realline', 'realplus', 'real0to1', 'discreta'}
        realall: distribuciones de la familia `realline` + `realplus`
        realline: distribuciones continuas en el dominio (-inf, +inf)
        realplus: distribuciones continuas en el dominio [0, +inf)
        real0to1: distribuciones continuas en el dominio [0,1]
        discreta: distribuciones discretas
```

```
    ordenar : {'aic', 'bic'}
        criterio de ordenación de mejor a peor ajuste.

    verbose : bool
        Si se muestra información de las distribuciones seleccionadas
        (the default `True`).

    Returns
    -------
    resultados: data.frame
        distribucion: nombre de la distribución.
        log_likelihood: logaritmo del likelihood del ajuste.
        aic: métrica AIC.
        bic: métrica BIC.
        n_parametros: número de parámetros de la distribución de la distribución.
        parametros: parámetros del tras el ajuste

    Raises
    ------
    Exception
        Si `familia` es distinto de 'realall', 'realline', 'realplus', 'real0to1',
        o 'discreta'.

    Notes
    -----

    '''

    distribuciones = seleccionar_distribuciones(familia=familia, verbose=verbose)
    distribucion_ = []
```

```
log_likelihood_= []
aic_ = []
bic_ = []
n_parametros_ = []
parametros_ = []

for i, distribucion in enumerate(distribuciones):

  print(f"{i+1}/{len(distribuciones)}  Ajustando distribución: {distribucion.name}")
  try:
    parametros = distribucion.fit(data=x)
    nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                if not p=='x'] + ["loc","scale"]
    parametros_dict = dict(zip(nombre_parametros, parametros))
    log_likelihood = distribucion.logpdf(x, *parametros).sum()
    aic = -2 * log_likelihood + 2 * len(parametros)
    bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

    distribucion_.append(distribucion.name)
    log_likelihood_.append(log_likelihood)
    aic_.append(aic)
    bic_.append(bic)
    n_parametros_.append(len(parametros))
    parametros_.append(parametros_dict)

    resultados = pd.DataFrame({
            'distribucion': distribucion_,
            'log_likelihood': log_likelihood_,
            'aic': aic_,
            'bic': bic_,
            'n_parametros': n_parametros_,
```

```
                'parametros': parametros_,

            })

        resultados = resultados.sort_values(by=ordenar).reset_index(drop=True)
        print ('resultados---',resultados)
      except Exception as e:
        print(f"Error al tratar de ajustar la distribución {distribucion.name}")
        print(e)
        print("")

  return resultados

def plot_distribucion(texto, x, nombre_distribucion, ax=None):
  '''
  Esta función superpone la curva de densidad de una distribución con el
  histograma de los datos.

  Parameters
  ----------
  x : array_like
    datos con los que ajustar la distribución.

  nombre_distribuciones : str
    nombre de una de las distribuciones disponibles en `scipy.stats`.

  Returns
  -------
  resultados: matplotlib.ax
    gráfico creado
```

```
    Raises
    ------

    Notes
    -----
    '''

    distribucion = getattr(stats, nombre_distribucion)

    parametros = distribucion.fit(data=x)

    nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                         if not p=='x'] + ["loc","scale"]
    parametros_dict = dict(zip(nombre_parametros, parametros))

    log_likelihood = distribucion.logpdf(x, *parametros).sum()

    aic = -2 * log_likelihood + 2 * len(parametros)
    bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

    x_hat = np.linspace(min(x), max(x), num=1000)
    y_hat = distribucion.pdf(x_hat, *parametros)

    if ax is None:
       fig, ax = plt.subplots(figsize=(7,4))

    ax.plot(x_hat, y_hat, linewidth=2, label=distribucion.name)
    ax.hist(x=x, density=True, bins=10, color="#3182bd", alpha=0.5);
    ax.plot(x, np.full_like(x, -0.01), '|k', markeredgewidth=1)
    ax.set_title('Ajuste distribución '+texto)
    ax.set_xlabel('x')
```

```
    ax.set_ylabel('Densidad de probabilidad')
    ax.legend();

    print('----------------------')
    print('Resultados del ajuste')
    print('----------------------')
    print(f"Distribución:  {distribucion.name}")
    print(f"Dominio:      {[distribucion.a, distribucion.b]}")
    print(f"Parámetros:    {parametros_dict}")
    print(f"Log likelihood: {log_likelihood}")
    print(f"AIC:          {aic}")
    print(f"BIC:          {bic}")

    return ax


def plot_multiple_distribuciones(texto,x, nombre_distribuciones, ax=None):
    '''
    Esta función superpone las curvas de densidad de varias distribuciones
    con el histograma de los datos.

    Parameters
    ----------
    x : array_like
        datos con los que ajustar la distribución.

    nombre_distribuciones : list
        lista con nombres de distribuciones disponibles en `scipy.stats`.

    Returns
    -------
```

```
    resultados: matplotlib.ax
        gráfico creado

    Raises
    ------

    Notes
    -----
    '''

    if ax is None:
        fig, ax = plt.subplots(figsize=(7,4))

    ax.hist(x=x, density=True, bins=10, color="#3182bd", alpha=0.5)
    ax.plot(x, np.full_like(x, -0.01), '|k', markeredgewidth=1)
    ax.set_title('Ajuste distribuciones al histograma de line 332'+texto+' anuales totales')
    ax.set_xlabel('x')
    ax.set_ylabel('Densidad de probabilidad')

    for nombre in nombre_distribuciones:

        distribucion = getattr(stats, nombre)

        parametros = distribucion.fit(data=x)

        nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                             if not p=='x'] + ["loc","scale"]
        parametros_dict = dict(zip(nombre_parametros, parametros))

        log_likelihood = distribucion.logpdf(x, *parametros).sum()
```

```
        aic = -2 * log_likelihood + 2 * len(parametros)
        bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

        x_hat = np.linspace(min(x), max(x), num=1000)
        y_hat = distribucion.pdf(x_hat, *parametros)
        ax.plot(x_hat, y_hat, linewidth=2, label=distribucion.name)

    ax.legend();

    return ax

def ajuste(serie,texto):
    serie = serie
    texto = texto
    # Ajuste y comparación de distribuciones
    #
=========================================================================
======
    resultados = comparar_distribuciones(texto,
            x=serie.to_numpy(),
            familia='realall',
            ordenar='aic',
            verbose=False
            )
    resultados

    fig, ax = plt.subplots(figsize=(8,5))


    plot_distribucion(texto,
        x=serie.to_numpy(),
        nombre_distribucion=resultados['distribucion'][0],
```

```
        ax=ax
        );

    fig, ax = plt.subplots(figsize=(8,5))

    plot_multiple_distribuciones(texto,
        x=serie.to_numpy(),
        nombre_distribuciones=resultados['distribucion'][:4],
        ax=ax
        );
    return

#MAIN CÓDIGO
def main():
    ingr = []
    gast = []
    ingr,gast = datosIngrGast()

    for item in range(1,3):

        comodin = []
        texto = ''
        serie = []
        comodin = ingr if item ==1 else gast
        texto = 'Ingresos' if item ==1 else 'Gastos'
        #estadisticos básicos
        serie = pd.Series(comodin)
        ajuste(serie, texto)
    return
```

```
# -*- coding: utf-8 -*-
"""
Created on 24/6/24
ph-seleDist
Tomado del artículo 'Ajuste y selección de distribuciones con Python' de Joaquín Amat Rodrigo
Realizo minimas y escasas adaptaciones para cargar y preparar mis datos
@author: invat
"""

import matplotlib.pyplot as plt
import mysql.connector
import numpy as np
from scipy import stats
import ph_herramientas
#from ph_herramientas import selectBDNoWhere
import pandas as pd
import inspect
import warnings

warnings.filterwarnings('ignore')

miConexion=mysql.connector.connect(host="localhost",database="tercerbd", \
   user="invat",password="jquintas49",auth_plugin="mysql_native_password")

global resultados



def datosIngrGast():
   #sql1 = SELECT tiempo, sum(va4), sum(va5) FROM tercerbd.iteracionecon group by tiempo
   #sql1='Select ingrxTrab,gastoanyo FROM tercerbd.iteracionecon where tiempo > 0'
```

```
    sql1 = 'SELECT ingrxTrab, gastoanyo FROM tercerbd.iteracionecon  where (tiempo > 0
and (18 <= edad <65) ) and ingrxTrab > 0 order by tiempo, ingrxTrab '
    regis = ph_herramientas.selectBDNoWhere(sql1)
    del regis[0]
    r1 = []
    r2 = []
    for a in regis:
        #ingresos / gastos.-Divido adaptar datos a escala
        r1.append(round(float(a[0])/10_000,2))
        #gastos.- Divido
        r2.append(round(float(a[1])/10_000,2))
    return r1,r2


def seleccionar_distribuciones(familia='realplus', verbose=True):
    '''
    Esta función selecciona un subconjunto de las distribuciones disponibles
    en scipy.stats

    Parameters
    ----------
    familia : {'realall', 'realline', 'realplus', 'real0to1', 'discreta'}
        realall: distribuciones de la familia `realline` + `realplus`
        realline: distribuciones continuas en el dominio (-inf, +inf)
        realplus: distribuciones continuas en el dominio [0, +inf)
        real0to1: distribuciones continuas en el dominio [0,1]
        discreta: distribuciones discretas

    verbose : bool
        Si se muestra información de las distribuciones seleccionadas
        (the default `True`).
```

```
  Returns
  -------
  distribuciones: list
     listado con las distribuciones (los objetos) seleccionados.

  Raises
  ------
  Exception
     Si `familia` es distinto de 'realall', 'realline', 'realplus', 'real0to1',
     o 'discreta'.

  Notes
  -----
     Las distribuciones levy_stable y vonmises han sido excluidas por el momento.

  '''

  distribuciones = [getattr(stats,d) for d in dir(stats) \
              if isinstance(getattr(stats,d), (stats.rv_continuous, stats.rv_discrete))]

  exclusiones = ['levy_stable', 'vonmises', 'studentized_range','chi2']
  distribuciones = [dist for dist in distribuciones if dist.name not in exclusiones]
  #Delimito desde cero
  dominios = {
     'realall' : [-np.inf, np.inf],
     'realline': [np.inf,np.inf],
     'realplus': [0, np.inf],
     'real0to1': [0, 1],
     'discreta': [None, None],
  }
```

```
  distribucion = []
  tipo = []
  dominio_inf = []
  dominio_sup = []

  for dist in distribuciones:
    distribucion.append(dist.name)
    tipo.append(np.where(isinstance(dist, stats.rv_continuous), 'continua', 'discreta'))
    dominio_inf.append(dist.a)
    dominio_sup.append(dist.b)

  info_distribuciones = pd.DataFrame({
                  'distribucion': distribucion,
                  'tipo': tipo,
                  'dominio_inf': dominio_inf,
                  'dominio_sup': dominio_sup
                 })

  info_distribuciones = info_distribuciones \
                  .sort_values(by=['dominio_inf', 'dominio_sup'])\
                  .reset_index(drop=True)

  if familia in ['realall', 'realline', 'realplus', 'real0to1']:
    info_distribuciones = info_distribuciones[info_distribuciones['tipo']=='continua']
    condicion = (info_distribuciones['dominio_inf'] == dominios[familia][0]) & \
            (info_distribuciones['dominio_sup'] == dominios[familia][1])
    info_distribuciones = info_distribuciones[condicion].reset_index(drop=True)

  if familia in ['discreta']:
    info_distribuciones = info_distribuciones[info_distribuciones['tipo']=='discreta']
```

```
    seleccion = [dist for dist in distribuciones \
                if dist.name in info_distribuciones['distribucion'].values]



    if verbose:
        print("-----------------------------------------------------")
        print("      Distribuciones seleccionadas               ")
        print("-----------------------------------------------------")
        with pd.option_context('display.max_rows', None, 'display.max_columns', None):
            print(info_distribuciones)

    return seleccion



def comparar_distribuciones(texto,x, familia='realplus', ordenar='aic', verbose=True):
    '''
    Esta función selecciona y ajusta un subconjunto de las distribuciones
    disponibles en scipy.stats. Para cada distribución calcula los valores de
    Log Likelihood, AIC y BIC.

    Parameters
    ----------
    x : array_like
        datos con los que ajustar la distribución.

    familia : {'realall', 'realline', 'realplus', 'real0to1', 'discreta'}
        realall: distribuciones de la familia `realline` + `realplus`
        realline: distribuciones continuas en el dominio (-inf, +inf)
        realplus: distribuciones continuas en el dominio [0, +inf)
        real0to1: distribuciones continuas en el dominio [0,1]
        discreta: distribuciones discretas
```

```
    ordenar : {'aic', 'bic'}
        criterio de ordenación de mejor a peor ajuste.

    verbose : bool
        Si se muestra información de las distribuciones seleccionadas
        (the default `True`).

    Returns
    -------
    resultados: data.frame
        distribucion: nombre de la distribución.
        log_likelihood: logaritmo del likelihood del ajuste.
        aic: métrica AIC.
        bic: métrica BIC.
        n_parametros: número de parámetros de la distribución de la distribución.
        parametros: parámetros del tras el ajuste

    Raises
    ------
    Exception
        Si `familia` es distinto de 'realall', 'realline', 'realplus', 'real0to1',
        o 'discreta'.

    Notes
    -----

    '''

    distribuciones = seleccionar_distribuciones(familia=familia, verbose=verbose)
    distribucion_ = []
```

```
log_likelihood_= []
aic_ = []
bic_ = []
n_parametros_ = []
parametros_ = []

for i, distribucion in enumerate(distribuciones):

  print(f"{i+1}/{len(distribuciones)}  Ajustando distribución: {distribucion.name}")
  try:
    parametros = distribucion.fit(data=x)
    nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                if not p=='x'] + ["loc","scale"]
    parametros_dict = dict(zip(nombre_parametros, parametros))
    log_likelihood = distribucion.logpdf(x, *parametros).sum()
    aic = -2 * log_likelihood + 2 * len(parametros)
    bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

    distribucion_.append(distribucion.name)
    log_likelihood_.append(log_likelihood)
    aic_.append(aic)
    bic_.append(bic)
    n_parametros_.append(len(parametros))
    parametros_.append(parametros_dict)

    resultados = pd.DataFrame({
            'distribucion': distribucion_,
            'log_likelihood': log_likelihood_,
            'aic': aic_,
            'bic': bic_,
            'n_parametros': n_parametros_,
```

```
                'parametros': parametros_,

            })

        resultados = resultados.sort_values(by=ordenar).reset_index(drop=True)
        print ('resultados---',resultados)
      except Exception as e:
        print(f"Error al tratar de ajustar la distribución {distribucion.name}")
        print(e)
        print("")

  return resultados

def plot_distribucion(texto, x, nombre_distribucion, ax=None):
  '''
  Esta función superpone la curva de densidad de una distribución con el
  histograma de los datos.

  Parameters
  ----------
  x : array_like
    datos con los que ajustar la distribución.

  nombre_distribuciones : str
    nombre de una de las distribuciones disponibles en `scipy.stats`.

  Returns
  -------
  resultados: matplotlib.ax
    gráfico creado
```

```
    Raises
    ------

    Notes
    -----
    '''

    distribucion = getattr(stats, nombre_distribucion)

    parametros = distribucion.fit(data=x)

    nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                         if not p=='x'] + ["loc","scale"]
    parametros_dict = dict(zip(nombre_parametros, parametros))

    log_likelihood = distribucion.logpdf(x, *parametros).sum()

    aic = -2 * log_likelihood + 2 * len(parametros)
    bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

    x_hat = np.linspace(min(x), max(x), num=1000)
    y_hat = distribucion.pdf(x_hat, *parametros)

    if ax is None:
       fig, ax = plt.subplots(figsize=(7,4))

    ax.plot(x_hat, y_hat, linewidth=2, label=distribucion.name)
    ax.hist(x=x, density=True, bins=10, color="#3182bd", alpha=0.5);
    ax.plot(x, np.full_like(x, -0.01), '|k', markeredgewidth=1)
    ax.set_title('Ajuste distribución '+texto)
    ax.set_xlabel('x')
```

```
    ax.set_ylabel('Densidad de probabilidad')
    ax.legend();

    print('----------------------')
    print('Resultados del ajuste')
    print('----------------------')
    print(f"Distribución:  {distribucion.name}")
    print(f"Dominio:      {[distribucion.a, distribucion.b]}")
    print(f"Parámetros:    {parametros_dict}")
    print(f"Log likelihood: {log_likelihood}")
    print(f"AIC:          {aic}")
    print(f"BIC:          {bic}")

    return ax


def plot_multiple_distribuciones(texto,x, nombre_distribuciones, ax=None):
    '''
    Esta función superpone las curvas de densidad de varias distribuciones
    con el histograma de los datos.

    Parameters
    ----------
    x : array_like
      datos con los que ajustar la distribución.

    nombre_distribuciones : list
      lista con nombres de distribuciones disponibles en `scipy.stats`.

    Returns
    -------
```

```
    resultados: matplotlib.ax
        gráfico creado

    Raises
    ------

    Notes
    -----
    '''

    if ax is None:
        fig, ax = plt.subplots(figsize=(7,4))

    ax.hist(x=x, density=True, bins=10, color="#3182bd", alpha=0.5)
    ax.plot(x, np.full_like(x, -0.01), '|k', markeredgewidth=1)
    ax.set_title('Ajuste distribuciones al histograma de line 332'+texto+' anuales totales')
    ax.set_xlabel('x')
    ax.set_ylabel('Densidad de probabilidad')

    for nombre in nombre_distribuciones:

        distribucion = getattr(stats, nombre)

        parametros = distribucion.fit(data=x)

        nombre_parametros = [p for p in inspect.signature(distribucion._pdf).parameters \
                             if not p=='x'] + ["loc","scale"]
        parametros_dict = dict(zip(nombre_parametros, parametros))

        log_likelihood = distribucion.logpdf(x, *parametros).sum()
```

```
        aic = -2 * log_likelihood + 2 * len(parametros)
        bic = -2 * log_likelihood + np.log(x.shape[0]) * len(parametros)

        x_hat = np.linspace(min(x), max(x), num=1000)
        y_hat = distribucion.pdf(x_hat, *parametros)
        ax.plot(x_hat, y_hat, linewidth=2, label=distribucion.name)

    ax.legend();

    return ax

def ajuste(serie,texto):
    serie = serie
    texto = texto
    # Ajuste y comparación de distribuciones
    #
=========================================================================
======
    resultados = comparar_distribuciones(texto,
            x=serie.to_numpy(),
            familia='realall',
            ordenar='aic',
            verbose=False
            )
    resultados

    fig, ax = plt.subplots(figsize=(8,5))


    plot_distribucion(texto,
        x=serie.to_numpy(),
        nombre_distribucion=resultados['distribucion'][0],
```

```
        ax=ax
        );

    fig, ax = plt.subplots(figsize=(8,5))

    plot_multiple_distribuciones(texto,
        x=serie.to_numpy(),
        nombre_distribuciones=resultados['distribucion'][:4],
        ax=ax
        );
    return

#MAIN CÓDIGO
def main():
    ingr = []
    gast = []
    ingr,gast = datosIngrGast()

    for item in range(1,3):

        comodin = []
        texto = ''
        serie = []
        comodin = ingr if item ==1 else gast
        texto = 'Ingresos' if item ==1 else 'Gastos'
        #estadisticos básicos
        serie = pd.Series(comodin)
        ajuste(serie, texto)
    return
```