Este documento se cita como

Garcia-Sabater, Jose P. (2021)
Programación Matemática en Python con PULP
RIUNET Repositorio UPV
http://hdl.handle.net/10251/XXXX

Contenido

TUTORIAL PARA COMENZAR A PROGAMAR EN PYTHON

Disclaimer: El autor no sabe nada de programación, ni de Python, ni en general de informática.

Solo pretende que sus alumnos le pierdan miedo a una herramienta que solo requiere horas para conseguir a hacer sus pinitos.

Dudo mucho que mis alumnos dediquen su vida a programar. Pero dudo menos que mis alumnos tienen que decidir si quieren que de su trabajo mande un ordenador o que su trabajo sea mandar a ordenadores.

Mis alumnos no saben de Python ni de otros lenguajes modernos.

Mis alumnos ya dieron programación en primero de grado. Así que saben lo que son las variables y las funciones, y los bucles y todo eso. Y además mis alumnos son gente a la que no les molesta explorar y equivocarse, vamos que les gusta aprender.

Si no eres como mis alumnos, este manual te va a parecer un caos y/o una pérdida de tiempo. No sigas leyendo, no vale la pena.

Incorporar sobre arcos y demás en Pulp (correo enviado a Anny)

Incorporar en Pulp comentarios sobre microgaps y cosas así

INTRODUCCIÓN

Para hacer este manual he visitado muchas páginas de internet y he visto muchos vídeos de youtube. Algunos son muy buenos, otros no tanto. Pero sobre todo me gustan estos:

- https://www.youtube.com/watch?v=rfscVS0vtbw
- https://www.youtube.com/watch?v=_uQrJ0TkZlc

El primero dura 4 horas y el segundo 6. El primero lo hice casi entero. El segundo lo puse a doble velocidad.

Pero sobre todo se aprende preguntando la comunidad de stackoverflow y otros como ellos están ahí para ayudarte y responden todas las preguntas (es casi imposible hacer una pregunta que no haya sido ya contestada)… aunque claro las preguntas hay que hacerlas en inglés.

INSTALACIÓN

Cada uno sabrá el ordenador que tiene así que cada uno tendrá que resolver sus problemillas de instalación.

Python es un lenguaje y hay que instalarlo en el ordenador. Yo instalé al última versión r la última versión de Python 3 (cuando lo escbirí 3.8.5) python.org https://www.python.org/downloads/release/python-385/ pero supongo que eso irá cambiando.

Seguí estas instrucciones: Como instalar Python en Windows 10 (https://matthewhorne.me/how-to-install-python-and-pip-on-windows-10/). Como todos los instaladores pide autorizaciones y demás.

Muy importante (por lo menos lo fue para mí) es que hagas una instalación personalizada para todos los usuarios del PC en la carpeta raíz. La primera vez no lo hice así y se puso a crear carpetas con nombres raros, largos y llenos de blancos y al final es un infierno progrmar.

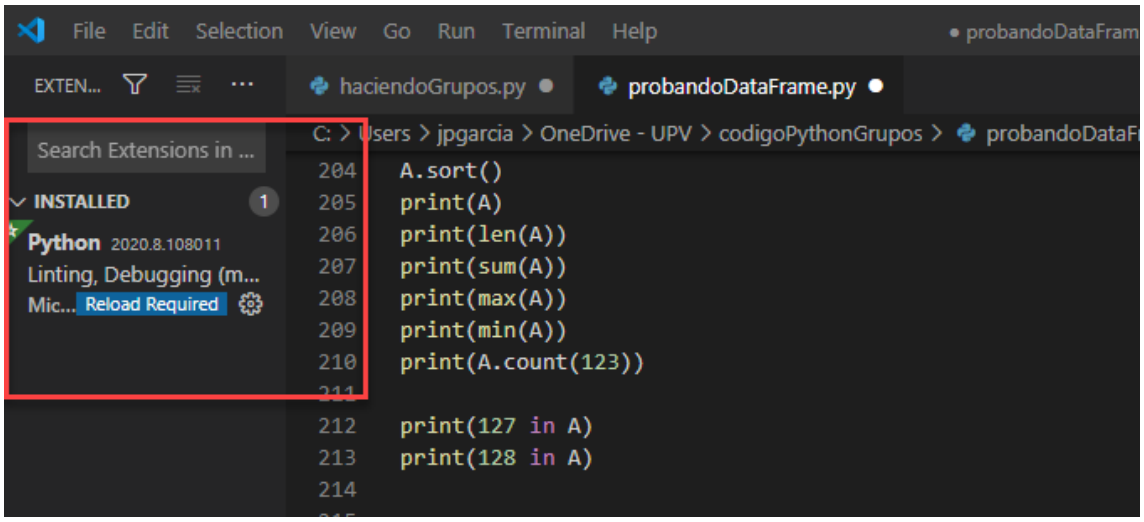
Para poder programar hace falta un editor. Yo instalé en mi ordenador *Geany* (tienes en el anexo como lo instalé). Mucha gente recomienda también *PyCharm* Pero ahora estoy trabajando con ***visual studio code*** y ese el que te recomiendo https://code.visualstudio.com/.

Si te lo descargas te dirá que puedes instalarte soporte para Python (harás bien en instalártelo) aunque eso solo sirve para que el fichero Python se vea mejor (un fichero Python es un fichero con la extensión .py)

Y cuando ya están instalados el VSCode y el Python se han de “conectar”. Porque el editor no sabe que tienes instalado el Python. Aquí lo explican bastante bien

https://www.youtube.com/watch?v=QtWry1YNLks

Se trata de ir a la ventana de extensiones y decirle que active el Python (que está allí) o por lo menos en mi caso estaba ;-)



Ya que estás instalando cosas aprovecha e instálate el PIP también

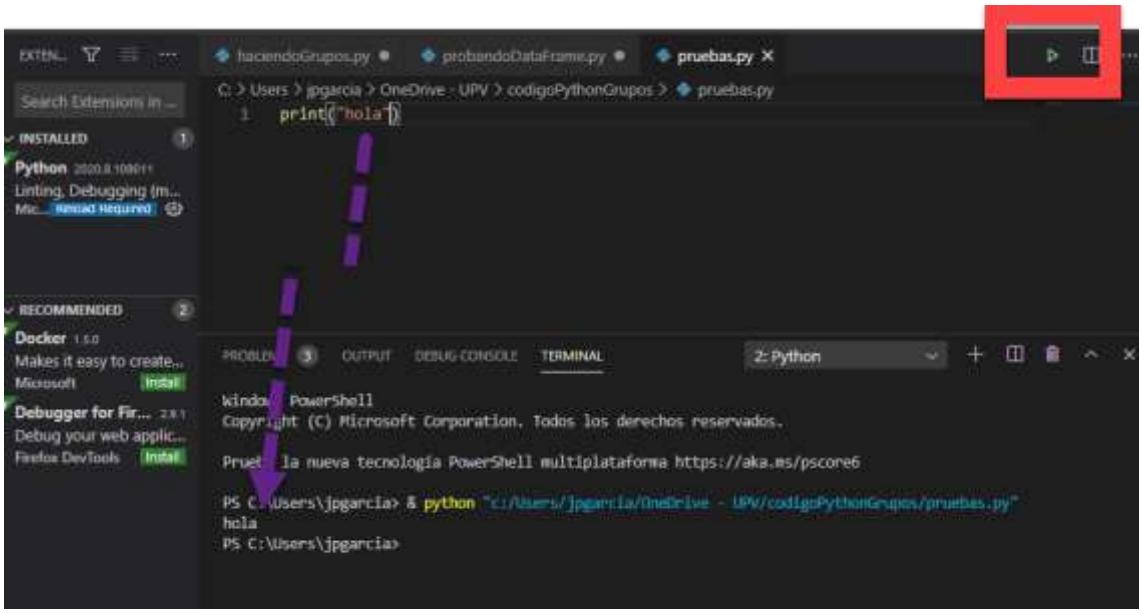
https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/

Ya te enterarás para que sirve cuando hable de paquetes

COMENZANDO

La pantalla tiene dos partes principales la zona donde escribes código y la consoloa.

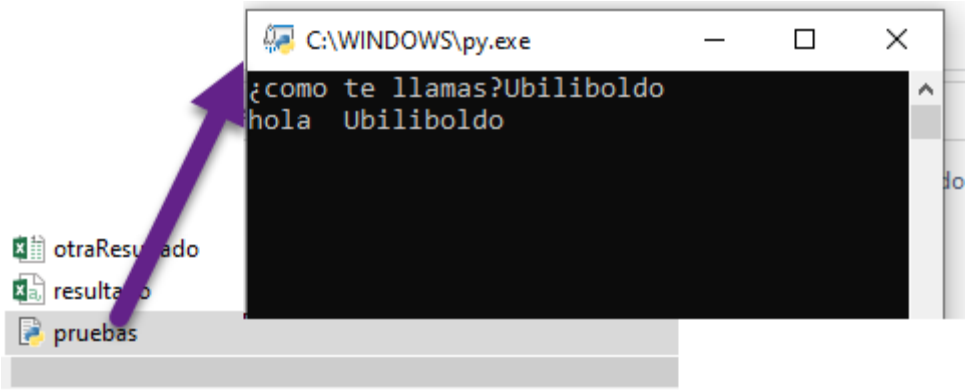
Si creas un fichero (con la extensión .py) en la que solo hagas print("hola") en la consola y le das al play (el botón verde) pues escribe hola



Pero es que es más. Si escribes en el fichero

```
1   a= input("¿como te llamas?")
2   print("hola ",a)
3   a=input()
```

Y grabas el fichero, y luego vas al explorador de Windows y buscas el fichero y lo ejecutas con doble click (si está asociado a Python)… Lo ejecuta también.



A estas alturas ya sabes que es una variable, y que es la función print() y la función input y que es la consola, y que se ponen paréntesis, y que los strings se ponen en naranja en el *VSC*

UN POCO DE TEORÍA

PAQUETES O LIBRERÍAS

Un paquete es una carpeta que tiene varios módulos. https://tutorial.recursospython.com/modulos-y-paquetes/#:~:text=Un%20paquete%20es%20una%20carpeta,carpeta%20con%20la%20siguiente%20estructura.&text=Debe%20contener%20siempre%20un%20archivo,no%20de%20una%20simple%20carpeta.

Por ejemplo, si quiero calcular números aleatorios puede crear mis propios algoritmos. Pero es más fácil que busque un paquete que ya tenga implementadas las funciones principales. https://docs.python.org/3/library/random.html

Instalar el paquete exige saber utilizar el PIP https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/ que no es más que el programa que instala paquetes.

Pero luego en cada uno de tus programas deberás importar el paquete o los módulos que te interesan. Esto es así para que los programas no estén llenos de paquetes inútiles (como las aplicaciones malas de móvil)

```
from tkinter.filedialog import askopenfilename
import csv
import pandas as pd
import numpy as np
```

Puedes importar todo el módulo o solo algún atributo, y puedes ponerle un alias (as) para facilitar su uso o solo para que no haya conflictos. https://likegeeks.com/es/modulos-de-python/

```python
import random

for i in range(30):
    print(random.randint(0,i), "es un número aleatorio entre",0," y ",i)
```

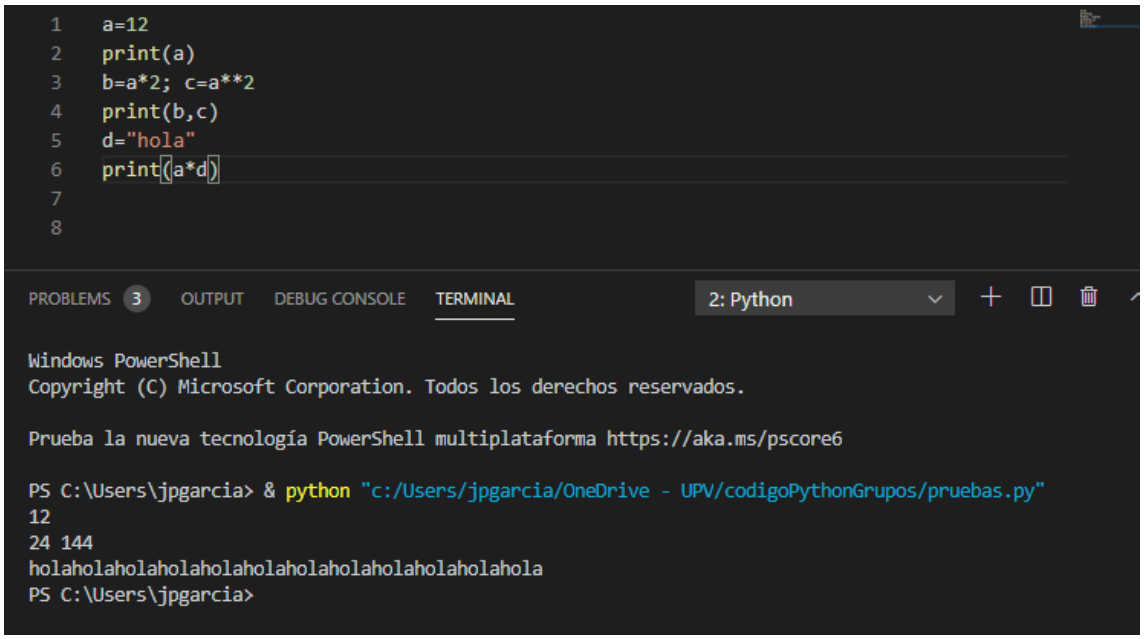
```
PROBLEMS 3  OUTPUT  DEBUG CONSOLE  TERMINAL  2: Python

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\jpgarcia> & python "c:/Users/jpgarcia/OneDrive - UPV/codigoPythonGrupos/pruebas.py"
0 es un número aleatorio entre 0  y  0
1 es un número aleatorio entre 0  y  1
0 es un número aleatorio entre 0  y  2
1 es un número aleatorio entre 0  y  3
0 es un número aleatorio entre 0  y  4
5 es un número aleatorio entre 0  y  5
5 es un número aleatorio entre 0  y  6
2 es un número aleatorio entre 0  y  7
2 es un número aleatorio entre 0  y  8
5 es un número aleatorio entre 0  y  9
0 es un número aleatorio entre 0  y  10
7 es un número aleatorio entre 0  y  11
8 es un número aleatorio entre 0  y  12
3 es un número aleatorio entre 0  y  13
11 es un número aleatorio entre 0  y  14
15 es un número aleatorio entre 0  y  15
7 es un número aleatorio entre 0  y  16
10 es un número aleatorio entre 0  y  17
14 es un número aleatorio entre 0  y  18
15 es un número aleatorio entre 0  y  19
8 es un número aleatorio entre 0  y  20
8 es un número aleatorio entre 0  y  21
19 es un número aleatorio entre 0  y  22
8 es un número aleatorio entre 0  y  23
20 es un número aleatorio entre 0  y  24
17 es un número aleatorio entre 0  y  25
12 es un número aleatorio entre 0  y  26
21 es un número aleatorio entre 0  y  27
15 es un número aleatorio entre 0  y  28
15 es un número aleatorio entre 0  y  29
PS C:\Users\jpgarcia>
```

VARIABLES

Una variable es un sitio donde guardamos algún tipo de dato https://www.tutorialpython.com/variables-en-python/#:~:text=Tipos%20de%20variables%20en%20python,de%20uno%20u%20otro%20tipo

Los datos pueden ser booleanos, caracteres, cadenas de caracteres –strings-, números, (enteros, float…)

Si quiero transformar un entero en un string se utiliza la función str(). Si quiero transformar un entero en un string utilizo la función int().



Las variables son (en general) locales de la función en la que están desarrollándose. Si queremos que sean globales y están en una función hay que denominarlas **global.** Pero aún estás lejos de saber eso para qué sirve.

LISTAS

Lo más característico del Python (desde mi punto de vista de extraño a todo esto) son las listas.

Una lista es una estructura de datos formada por una secuencia ordenada de objetos. https://devcode.la/tutoriales/listas-python/ Un objeto puede ser datos (cualquier tipo de datos) o incluso funciones. En realidad las listas son un tipo de algo más profundo (iterators, iterables… de los que no tengo mucha idea https://es.stackoverflow.com/questions/84591/diferencia-entre-objetos-iterable-iterator-y-secuencias-en-python-3). Seguro que es muy útil pero no lo entiendo.

Sobre las listas se pueden hacer muchas cosas. Una cosa importante a saber es que el primer elemento es el cero, y el segundo es el 1. Y algunas cosas que se pueden hacer con las listas son las siguientes:

Crear	A=[1,1,2,3,5,8] print(A)	[1,1,2,3,5,8]
Acceder	A[1]=123	[1,123,2,3,5,8]

	print(A)	
Recorrer	for elem in A: print(A[i])	1 123 2 3 5 8
Elegir subconjuntos (slicing)	print(A[1:3]) print(A[:3]) print(A[3:])	[123,2,3] [1,123,2] [3,5,8]
insertar	A.insert(2,"hola") print(A)	[1,123,"hola",2,3,5,8]
Insertar al final	A.append("adios") print(A)	[1,123,"hola",2,3,5,8,"adios"]
Extender con otra lista	A.extend([120,127])	[1,123,"hola",2,3,5,8,"adios",120,127]
Identificar en qué posición están	indice=A.index("hola") A[indice]=123 print(A)	[1,123,124,2,3,5,8,"adios",120,127]
Eliminar un elemento	indice=A.index("adios") A.pop(índice) print(A)	[1,123,123,2,3,5,8,120,127]
ordenar	A.sort() print(A)	[1,2,3,5,8,120,123,123,127] "solo funciona si todo son numeros o todo son strings
invertir	A.reverse() print(A)	[127,123,123,120,8,5,3,2,1]

Tambien podemos aplicar funciones como len,sum, min,max,count

longitud	print(len(A))	7
sumar	print(sum(A))	512

Máximo	print(max(A))	127
Mínimo	print(min(A))	1
Contar	print(count(123))	2
Buscar dentro	print(127 in A) print(128 in A)	True False

Y hay muchos sitios en internet donde ponen muchas maneras de ejecutar más funciones, solo hay que preguntarle a google.

Algo que compensa destacar es que si una lista es A y hago B=A. Entonces A y B serán la misma lista. Esto es, si cambio un elemento de A entonces lo cambio también de B. Si lo que quiero es copiar A en B entonces debo hacer B=A.copy()

Y ¿qué ocurre cuando una lista es una lista de listas? ¿puedo multiplicar listas?

B=[[1,200],[3,4],[5,6,7]] print(B)	200
B=2*B print(B)	[[1, 2], [3, 4], [5, 6, 7], [1, 2], [3, 4], [5, 6, 7]]

Y por eso quizá compensa no dar por supuesto lo que se sabe de las listas.

Los más viejos utilizábamos vectores (arrays se llamaban) y estos estaban formados por datos que eran idénticos entre sí. Por eso es posible que nos sintamos más cómodos utilizando el paquete numpy y las funciones asociadas a numpy.array https://jakevdp.github.io/PythonDataScienceHandbook/02.02-the-basics-of-numpy-arrays.html

DICCIONARIOS

Los diccionarios son estructuras de datos que es muy parecido a una lista pero que está pensada para que podamos acceder a los datos a través de claves más que a través de posiciones. Son diccionarios a partir de la clave buscamos dentro.

D={12:'Hola', 'Cosa': 3, 125 : 25, "bonjour": "guten tag"}	
print(D[12], D['bonjour'])	Hola guten tag
print(D['Cosa'], D[125])	3 25

TUPLAS

Y aquí ya alcanzo el límite de mi conocimiento. Las tuplas son conjuntos ordenado e inmutables de elementos del mismo tipo. https://www.mclibre.org/consultar/python/lecciones/python-tupla.html#:~:text=Qu%C3%A9%20son%20las%20tuplas,par%C3%A9ntesis%20y%20separados%20por%20comas.

No sé para que sirven, ni porqué queremos que sean inmutables pero resulta que el paquete de programación matemática pulp las utiliza. Así que yo las utilizo

CONDICIONALES

Desde mi punto de vista programar consiste en leer y modificar valores de variables utilizando bucles y condicionales.

El clásico **Si (condición) Entonces Ejecuta** se escribe utilizando una de las particularidades más especiales de Python para mí: **la indentación (tabulación) como modo de estructurar el código**.

Es elegante y confuso a la vez.

Aunque más confuso es saber porque se puede indentar con espacios o con tabuladores (y a veces el compilador acepta ambas y otras veces da error). Muy de freakies todo (https://www.youtube.com/watch?time_continue=8&v=SsoOG6ZeyUI)

Volviendo a los condicionales. La estructura básica es

if (condicion1): -bloque1- elif condicion2: -bloque2- else bloque3

```
a=int(input("cuantos años tienes?"))

if a<22:
    print("¿tan joven? Pues no lo aparentas")
elif a<30:
    print("yo tengo un primo que es de tu edad")
else:
    print("¿de verdad? Pues no pareces ser tan viejo")
```

Y las condiciones (funciones que generan booleanos)

A>=B,A<B,	comparan
A==B y A!=B	Igual o distinto
A in B y not(B in A)	Uno dentro de otro

La función in funciona de manera diferente si es una lista o un diccionario

BUCLES

Y ahora a los loops. Para mostrar un loop con un for utilizo un modo de calcular los primeros 100 numeros de la serie de Fibonacci

```
A1=1; print(A1)
A2=1; print(A2)
suma=A1+A2
for i in range(10):
    print(suma)
    A1=A2
    A2=suma
    suma=A1+A2
```

Fíjate que para indicar lo que hay dentro del loop lo ha indentado (tabulado)

Y un loop condicional (un **while**), la misma serie pero parará cuando alcance un valor de 10.000 o más

```
A1=1; print(A1)
A2=1; print(A2)
Suma=A1+A2
while suma<10.000:
    print(Suma)
    A1=A2
    A2=suma
    suma=A1+A2
```

Y el **while** también. Siempre lo hace así (y siempre se me olvida poner los dos puntos)

DEFINICIÓN DE FUNCIONES

Las funciones son trozos de código reutilizables que se encarga de realizar una tarea.

https://devcode.la/tutoriales/funciones-en-python/

Las funciones reciben parámetros desde fuera (metidos entre párentesis) y devuelven (o no datos hacia afuera con la función return

Las variables que se utilizan son locales (que signfica que no afectaran a ninguna variable que sea externa)

```
def letraNIF(num):
    clave='TRWAGMYFPDXBNJZSQVHLCKE'
    letra=clave[num%23]
    NIF=str(num)+letra
    respuesta=[letra,NIF]
    return respuesta

print("la letra es:",letraNIF(19948657)[0])
print("el NIF entero es:",letraNIF(19948657)[1])
```

En el anterior ejemplo entra un número y sale una lista con dos strings. Se utiliza también la función módulo (%) que es el resto y la función str(), y se leen los elementos de una lista directamente desde la llamada de la función.

LEYENDO DATOS DE FUENTES EXTERNAS

FICHEROS DE TEXTO

La entrada de datos se puede hacer tecleándolos uno a uno con la función input. Pero eso es muy cansado.

Los podemos leer de ficheros. Lo recomendable ficheros csv. SI googleo “read files csv Python” lo primero que sale es https://realpython.com/python-csv/

Y lo explican bastante bien. Como para poder leer un fichero primero hay que tenerlo voy a hacer un código que escribe un fichero, luego lo lee, lo modifica y luego lo escribe de nuevo con otro formato.

```python
import csv
#primero escribo el fichero
with open('miFichero2020.csv',mode='w',newline="") as fichero1:
    escribidor=csv.writer(fichero1, delimiter=";")
    escribidor.writerow(["Jaime",12,"J"])
    escribidor.writerow(["Guille",10,"J"])
    escribidor.writerow(["Marta",8,"J"])

#luego leo el fichero
#para ello primero declaro las variables
nombre=[]
edad=[]
inicial=[]
with open('miFichero2020.csv') as fichero2:
    leedor=csv.reader(fichero2, delimiter=";")
    contador=0
    for linea in leedor:
        nombre.append(linea[0])
        edad.append(int(linea[1]))
        inicial.append(linea[2])


#en los que la edad sube 10 años

for i in range(len(edad)):
    edad[i]+=10

#Y ahora creo el fichero de 10 años despues
with open('miFichero2030.csv',mode='w',newline="") as fichero3:
    escribidor=csv.writer(fichero3, delimiter=";")
    for i in range(len(nombre)):
        escribidor.writerow([nombre[i],edad[i],inicial[i]])
```

FICHEROS EXCEL

Si leer de ficheros de texto es fácil. No lo es menos leer y escribir ficheros Excel. En el código que presento a continuación creo un fichero, le cambio el nombre a una hoja, altero algunos valores primero uno a uno y luego con un bucle. Luego lo guardo. Y luego lo leo sobre unas listas. Atención al while que permite leer mientras no se acabe

```
import openpyxl
from openpyxl import load_workbook

fichero='LibroPrueba.xlsx' #esto lo guardará en el directorio raiz
wb = openpyxl.Workbook()
wb.save(fichero)


wb=load_workbook('LibroPrueba.xlsx')
ws=wb['Sheet'] #la hoja que crea la llama sheet
ws.title='Hoja1' #le cambio el nombre
ws.cell(1,1).value=1
ws.cell(2,1).value=2
ws.cell(3,1).value=3
for i in range(2,4):
    for j in range(1,4):
        ws.cell(j,i).value=i*ws.cell(j,1).value
wb.save(fichero)

ws=wb['Hoja1']
fila=1; simple=[]; doble=[]; triple=[]

while not(ws.cell(fila,1).value is None):
    simple.append(ws.cell(fila,1).value)
    doble.append(ws.cell(fila,2).value)
    triple.append(ws.cell(fila,3).value)
    fila=fila+1

print(simple)
print(doble)
print(triple)
```

Lo primero es elegir el paquete que queremos utilizar

Yo utilizo la librería openpyxl https://realpython.com/openpyxl-excel-spreadsheets-python/ pero he utilizado también xlwt (aunque creo que esa ya está desfasada) y dicen en la web que xlxsxwriter es más rápida cuando tienes muchos datos https://xlsxwriter.readthedocs.io/ y creo que además está vinculada a pandas.

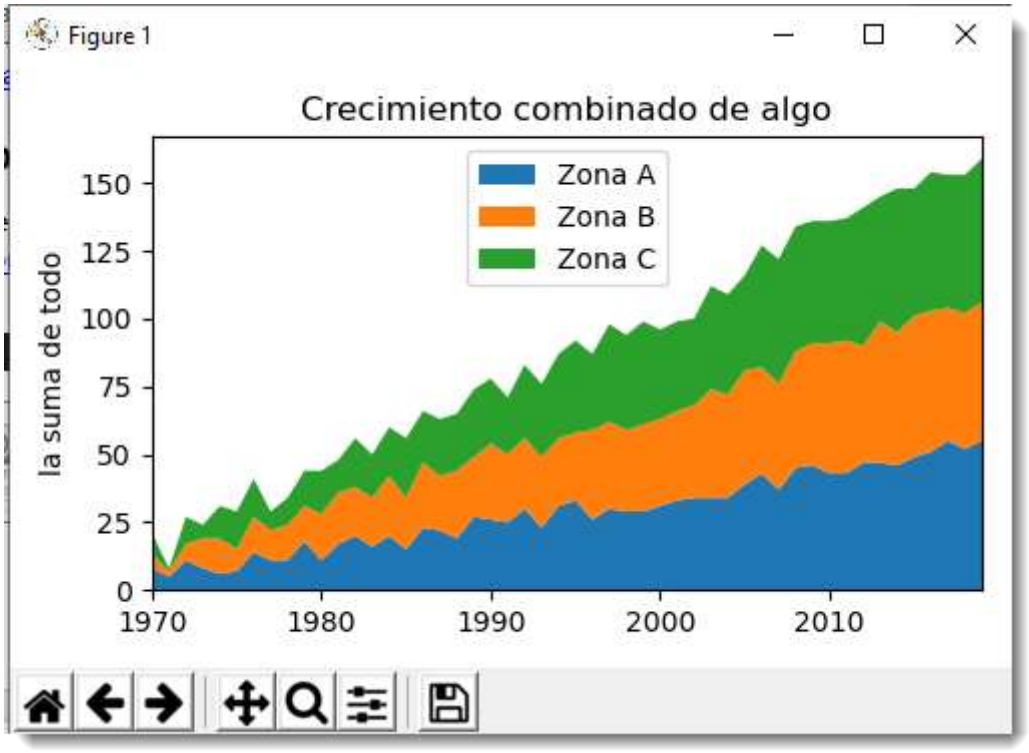
ENTREGANDO DATOS CON GRÁFICOS

Una de las maneras de sacar datos es utilizando la librería matplotlib y recomiendo https://realpython.com/python-matplotlib-guide/

```
import matplotlib.pyplot as plt
import numpy as np

rng = np.arange(50)
rnd = np.random.randint(0, 10, size=(3, rng.size))
```

```
yrs = 1970 + rng

fig, ax = plt.subplots(figsize=(5, 3))
ax.stackplot(yrs, rng + rnd, labels=['Zona A', 'Zona B', 'Zona C'])
ax.set_title('Crecimiento combinado de algo')
ax.legend(loc='upper center') #podría
ax.set_ylabel('la suma de todo')
ax.set_xlim(xmin=yrs[0], xmax=yrs[-1])
fig.tight_layout()

plt.show()
```

y lo que sale es esto



UTILIZANDO NUMPY

NumPy es una biblioteca que da soporte para crear vectores y matrices, y además aporta un gran número de funciones matemáticas.

De manera natural Python trabaja con listas muy muy versátiles, lo que es interesante. Demasiado versátiles para los que solemos trabajar con números que se expresan en forma de vectores (también llamadas arrays o matrices).

NumPy se encarga de esto y permite además hacer otras muchas cosas que tiene instalado y que nos puede servir. Dicen que numPy se utiliza con SciPy (pero no los he

utilizado al que) y con matPlotLib (la librería inmediatamente anterior). También dicen que numPy es como Matlab pero mejor y en abierto.

```
import numpy as np
a=np.array([1,2,3])
print(2*a)
b=np.array([1,2]) #crea un vector
print(b)
a=np.array([[1,2,3],[1,1,1]]) #crea una matriz
print(3*a) #la multiplica escalarmente
b=np.array([[1,1],[1,2],[2,3]]) #crea una amtriz
print(np.transpose(a)*b) #multiplica escalarmente la transpuesta de a por
 b
print(np.arange(1,11)) #crea un vector desde el 1 hasta el 10
print(np.rint(7/3)) #la parte entera del número decimal
print(np.sqrt(13)) #da la raiz cuadrada de 13
print(np.pi) #imprime el valor de pi según numpy
print(np.zeros(4,3,2)) #genera una matriz 3D de ceros de tamaño 4x3x2
print(np.sin(np.pi/2)) #el seno de pi/2
```

Desde luego numPy hace muchas más cosas pero lo mejor es buscar cada vez que necesitamos algo.

UTILIZANDO PANDAS PARA TRABAJAR CON DATOS

Intuyo que trabajar con datos es algo que hace habitualmente quien lee este documento. Y para trabajar con datos lo suyo es utilizar Pandas.

Pandas es una librería para manipular y analizar datos. Es una extensión de Numpy.

Según he llegado a entender (quizá mal) los datos se estructuran en dataframes. Un dataframe es una tabla que tiene filas, y en cada columna se guarda el dato correspondiente a la fila. Por ejemplo una tabla con los nombres, apellidos, teléfonos y emails de los alumnos de una clase. Pero también una tabla bidimensional que tenga en contenga los valores de una o más matrices, dedicando algunas columnas a los índices, y otras a los valores que ese conjunto de índices tienen.

Manejar un dataframe NO es manejar un array. Y por tanto la lógica array no aplica directamente.

Un data frame consta de columnas que contendrán datos para cada fila.

A continuación, vamos a crear un dataframe con datos aleatorios para jugar un poco

```
dates = pd.date_range('10/9/2020', periods=8)
df = pd.DataFrame(np.random.randn(8, 4),index=dates, columns=list("ABCD"))
print(df)
df.at['2020-10-11','B']=1111
df.iat[4,1]=2222
print(df) #esto imprime toda la tabla
print(df.B) #Esto imprime solo la columna 2
```

Y este es el resultado.

Con el date_range hemos creado un vector de 8 días empezando el 9 de octubre del 2020.

Ese vector se convertirá en el índice (el nombre de las filas) del dataframe.

A las columnas las vamos a llamar ABCD, y para eso utilizamos la función list que aplicada a un string, lo descompone en caracteres..

Y para llenarlo lo hacemos con datos aleatorios utilizando randn de numpy.

Y entonces lo imprimimos

	A	B	C	D
2020-10-09	-1.215641	0.718182	-1.646155	1.273744
2020-10-10	-1.663755	-0.996722	-1.006593	0.153628
2020-10-11	-0.431783	0.703972	-0.756641	0.337456
2020-10-12	-0.983890	0.245693	-0.567440	-0.357533
2020-10-13	-0.794146	-0.972377	1.336904	0.144589
2020-10-14	1.676720	-1.116907	-1.048679	0.498509
2020-10-15	-0.410881	0.126669	0.568239	-0.756097
2020-10-16	-0.923110	0.320227	0.520447	0.584835

Luego modificamos dos valores del dataframe utilizando dos funciones diferentes at e iat. At accede a las posiciones utilizando los nombres mientras que iat lo hace utilizando las posiciones. Si imprimimos la tabla con print(df) entonces vemos lo que ha cambiado.

	A	B	C	D
2020-10-09	-1.215641	0.718182	-1.646155	1.273744
2020-10-10	-1.663755	-0.996722	-1.006593	0.153628
2020-10-11	-0.431783	1111.000000	-0.756641	0.337456
2020-10-12	-0.983890	0.245693	-0.567440	-0.357533
2020-10-13	-0.794146	2222.000000	1.336904	0.144589
2020-10-14	1.676720	-1.116907	-1.048679	0.498509
2020-10-15	-0.410881	0.126669	0.568239	-0.756097
2020-10-16	-0.923110	0.320227	0.520447	0.584835

De hecho podríamos haber impreso solo la columna 'B' que es la que hemos cambiado con print(df.B)

```
2020-10-09       0.718182
2020-10-10      -0.996722
2020-10-11    1111.000000
2020-10-12       0.245693
2020-10-13    2222.000000
2020-10-14      -1.116907
2020-10-15       0.126669
2020-10-16       0.320227
```

Se podría haber cargado la tabla de otra manera

```
df=pd.DataFrame(
    [['Feb',12,20,30,2],
    ['Jun',12,20,28,2],
    ['Ago',12,22,32,6],
    ['Nov',15,2,26,7]],
    index=[0,1,2,3],
    columns=['mes','media','desv','maximo','minimo'])
print(df)
```

Y este es el resultado

```
   mes  media  desv  maximo  minimo
0  Feb     12    20      30       2
1  Jun     12    20      28       2
2  Aug     12    22      32       6
3  Nov     15     2      26       7
```

También se puede cargar el fichero leyéndolo directamente desde un csv

```
df2=pd.read_csv(nomFicher)
```

La tabla que yo he cargado se puede ver aquí

```
print(df2.values) #esto imprme los valores de la tabla
```

```
[['Ene' 12 20.3 30 2]
 ['Apr' 12 21.2 28 2]
 ['Jul' 12 21.6 32 3]
 ['Oct' 115 12.3 26 7]]
```

Como en general los dataframes son muy grandes a veces es interesante representar las primeras (o las últimas filas)

```
print(df2.head(2))
print(df2.tail(3))
```

Y evidentemente se puede tratar de entender las estructuras que tiene la tabla

```
print(df2.dtypes) #deveuelve los tipos de las columnas
print(df2.index) #devuelve la lísta de índices
```

```
print(df2.columns) #devuelve un objeto cuya primera parte es una lista
print(df2.columns[2]) #devuelve la columna en 3a posicion
print(len(df2.columns)) #devuelve el número de columnas
print(df2["col3"].size) #esto escribe el número de filas
print(df2.values) #esto imprme los valores de la tabla
```

Dado que pandas sirve para analizar datos era de esperar que existieran cosas como

```
print(df2.describe())
```

```
print(df2.sort_values('maximo',ascending=True))
```

Que hagan cosas como:

```
[5, datos estadísticos]
        media       desv     maximo    minimo
count    4.00   4.000000   4.000000  4.000000
mean    37.75  18.850000  29.000000  3.500000
std     51.50   4.400379   2.581989  2.380476
min     12.00  12.300000  26.000000  2.000000
25%     12.00  18.300000  27.500000  2.000000
50%     12.00  20.750000  29.000000  2.500000
75%     37.75  21.300000  30.500000  4.000000
max    115.00  21.600000  32.000000  7.000000
::::::::::::::::::::::::::::::::::::::::::::::::::
[6 ordenar por alguna columna]
   mes  media  desv  maximo  minimo
3  Oct    115  12.3      26       7
1  Apr     12  21.2      28       2
0  Ene     12  20.3      30       2
2  Jul     12  21.6      32       3
```

Se puede acceder a fragmentos de tabla. Por ejemplo (que ya se ha cometando) a la columna: **df2.media** da la columna media igual que lo hace **df2['media']**

Si hacemos **df2[['media','maximo']]** obtenderemos dos columnas. (OJO: Mucha atención al doble corchete.

Si queremos acceder a la segunda y tercera filas **df2[2:4]**

Los dataframe se pueden grabar como csv e incluso como xlsx

```
df2.to_csv(nomDir+"/resultado.csv")
df2.to_excel(nomDir+"/otraResultado.xlsx")
```

Convendría aprender a utilizar la función loc y la función iloc

```
print("solo las columnas media y maximo para todas las filas")
print(df2.loc[:,['media','maximo']]) #acabo de colocar loc (for location)
```

```
print("solo las columnas media y maximo para la fila 1")
print(df2.loc[1,['media','maximo']])
print("solo las columnas media y maximo para las fila 1 a 3")
print(df2.loc[1:3,['media','maximo']])
print("solo las columnas 2 y 4 para las fila 1 a 3")
print(df2.iloc[1:3,[2,4]]) #atencion que acabo de colocar iloc (for index
location)
```

También se puede aprender a filtrar datos

```
print("los que la media es mayor que 20")
print(df2[df2.media>20])
print("los que el mes es enero o julio")
print(df2[df2['mes'].isin(['Ene','Jul'])])
```

O a modificar datos

```
df2.loc[1,'media']=12000
print(df2)
df2.loc[:,'maximo']=df2.media
print(df2)
```

Se pueden añadir al final de la tabla más datos **append**

```
nuevo={'mes':'Dic','media':1,'desv':5,'maximo':6,'minimo':7}
df2=df2.append(nuevo,ignore_index=True)
print(df2)
nuevo={'mes':'Mar'}
df2=df2.append(nuevo,ignore_index=True)
print(df2)
```

Aunque si lo hacemos como en la segunda, va a generar celdas vacías que habrái que rellenar

```
print(df2.loc[df2.mes=='Mar'])
df2.loc[df2.mes=='Mar','maximo']=5
print(df2.loc[df2.mes=='Mar'])
df2.loc[df2.mes=='Mar','media']=8
print(df2.loc[df2.mes=='Mar'])
```

```
   mes  media  desv  maximo  minimo
5  Mar    NaN   NaN     NaN     NaN
   mes  media  desv  maximo  minimo
5  Mar    NaN   NaN     5.0     NaN
   mes  media  desv  maximo  minimo
5  Mar    8.0   NaN     5.0     NaN
```

Añadir permite también añadir directamente un dataframe a otro

```
df2=df2.append(df,sort=None,ignore_index=True)
```

Se pueden añadir columnas asignándoles valores o no

```
print("esto añade otra columna  sin valores")
df2['novaCol']=np.nan
print(df2)
print("esto añade otra columna con un valor resultado de un calculo")
df2['novaCol2']=(df2.media+2*df2.desv)
print(df2)
```

Y las columnas se pueden renombrar

```
encabezado("14 renombrar columnas")
df2.rename(columns={'novaCol':'prueba'}, inplace=True)
print(df2)
df2.columns=['col1','col2','col3','col4','col5','col6','col7']
print(df2)
```

Para leer un dataframe lo lógico es iterar sobre él y para eso existe un “algo” que se llama iterrows

```
for index, row in df2.iterrows(): #aprender a utilizar esto que itera el
indice y la línea a la vez
    print(index,row["col1"],row["col3"])
    if row["col1"]=="Ene":
        df2.loc[0,'col1']="Jan"
```

Los puristas dicen que se puede hacer más eficiente si haces itertuples, aunque yo la verdad es que no lo entiendo.

```
for row in df2.itertuples():
    print(row.Index, row.col1, row.col3)
```

Podemos investigar si un valor está en un dataframe

```
print(list(df2.col1))
existe= "Ene" in list(df2.col1)
print(existe)
```

```
existe= 22.1 in list(df2["col2"])
print(existe)
```

O si existen celdas que no están asignadas

```
print(df2["col3"].isnull()) #este da un verctor de trues y falsos
print("esto escribe verdadero si hay valore no nulos ",df2["col3"].isnull
().values.any()) #este da verdadero si hay un verdadero
print(np.isnan(df2.loc[5,"col3"])) #este da verdadero si hay un nan
print(np.isnan(df2.loc[6,"col3"])) #este da verdadero si hay un nan
```

Y en el colmo del desvarío puedes modificar valores de filas y columnas que han sido previamente filtradas

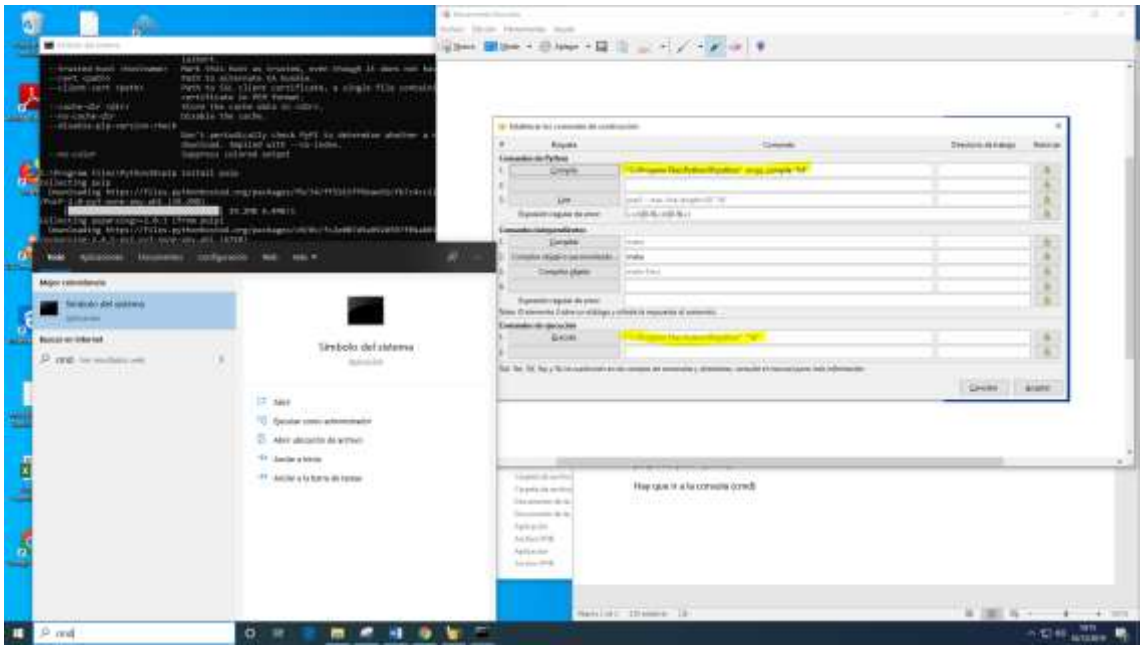
```
filtro1= df2.media==12
filtro2= df2.desv==20
print(df2.loc[(filtro1 & filtro2)])
df2.loc[(filtro1 & filtro2),'desv']=df2.media-df2.minimo+df2.maximo/2
df2.loc[(filtro1 & filtro2),'media']=df2.media/2+df2.minimo/2+df2.maximo/
2
print(df2.loc[(filtro1 & filtro2)])
```

LA LIBRERÍA (O PAQUETE) PULP

De entre las librerías que me utilizo destaca PULP. Es un paquete para el uso de herramientas de programación matemática (concretamente programación lineal es lo que yo conozco). https://coin-or.github.io/pulp/main/installing_pulp_at_home.html

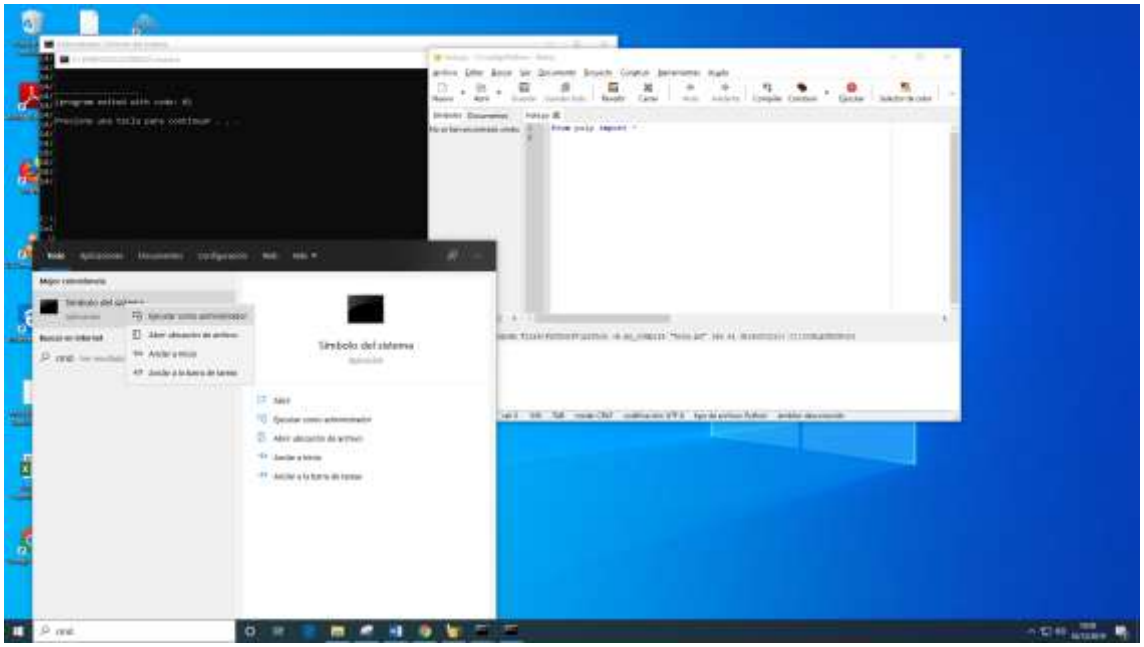
Para ello me he de descargar el paquete. Lo más fácil es utilizar pip (que es un instalador de paquetes para Python) que viene preinstalado.

Hay que abrir la consola (cmd) e ir al lugar donde está Python en mi caso en c:/program files/Python38 por la vía de usar comandos de msdos (lo cual me hace sentir joven). La ventana de comandos se ejecuta a través de cmd



Y entonces cuando estás en c:\Program Files\Python38 ejecutar **pip install pulp** como indican en https://pythonhosted.org/PuLP/main/installing_pulp_at_home.html

Importante: En mi caso he tenido que abrir el cmd como administrador con el botón derecho sobre "símbolo de sistema" porque si no no tenía permisos.



Y una vez instalado puedo comenzar a programar un modelo de programación matemática.

Este es el caso que se corresponde con el problema Jorge y Nuria de los apuntes de Programación Matemática. Se trata de generar las mesas de una boda en función de las características de las mesas y de las afinidades (o desafinidades) de los gruos entre sí.

Primero se importa el PULP y se cargan los datos.

Luego se crean las variables que van a ser necesarias

```
print("comienza a cargar el modelo")

prob  = LpProblem("AsignacionMesas", LpMaximize)
#la variable que dice si los de i se sientan en k
delta = LpVariable.dicts("delta",
            [(i,k) for i in agrupaciones for k in mesas],
            0,1, LpBinary)
#la variable donde se sienta cada agrupacion
x_var=  LpVariable.dicts("mesa",
            [(i) for i in agrupaciones],
            1,4, LpInteger)
#la variable que dice si dos agrupaciones están en la misma mesa
gamma=  LpVariable.dicts("gamma",
            [(i,j) for i in agrupaciones for j in agrupaciones],
            0,1, LpBinary)
beta=   LpVariable.dicts("beta",
            [(i,j) for i in agrupaciones for j in agrupaciones],
            0,1, LpBinary)
```

Luego se crea el modelo

```
# este es el objetivo
print("comienza a cargar el modelo")

prob += lpSum(Afinidad[i][j]*gamma[(i,j)] for i in agrupaciones
                                          for j in agrupaciones
                                          )

#esta es la restriccion de que no caben más en la mesa
for k in mesas:
        prob += lpSum(TamanyoAgrupacion[i]*(delta[(i,k)]) for i in agrupaciones

#esta es la restriccion en la que sólo se pone una vez en cada mesa
for i in agrupaciones:
        prob += lpSum((delta[(i,k)]) for k in mesas) == 1


#esta restriccion es para saber en que mesa están
for i in agrupaciones:
    prob += x_var[(i)] - lpSum(NumMesa[k]*delta[(i,k)] for k in mesas) == 0

#estas restricciones sirven para saber si dos están en la misma mesa
for i in agrupaciones:
    for j in agrupaciones:
        prob += ( x_var[(i)] - x_var[(j)] - numMesas*beta[(i,j)] ) <= 0

for i in agrupaciones:
    for j in agrupaciones:
        prob += ( x_var[(j)] - x_var[(i)] - numMesas*beta[(j,i)] ) <= 0

for i in agrupaciones:
    for j in agrupaciones:
        prob += beta[(i,j)] + beta[(j,i)] + gamma[(i,j)] == 1
```

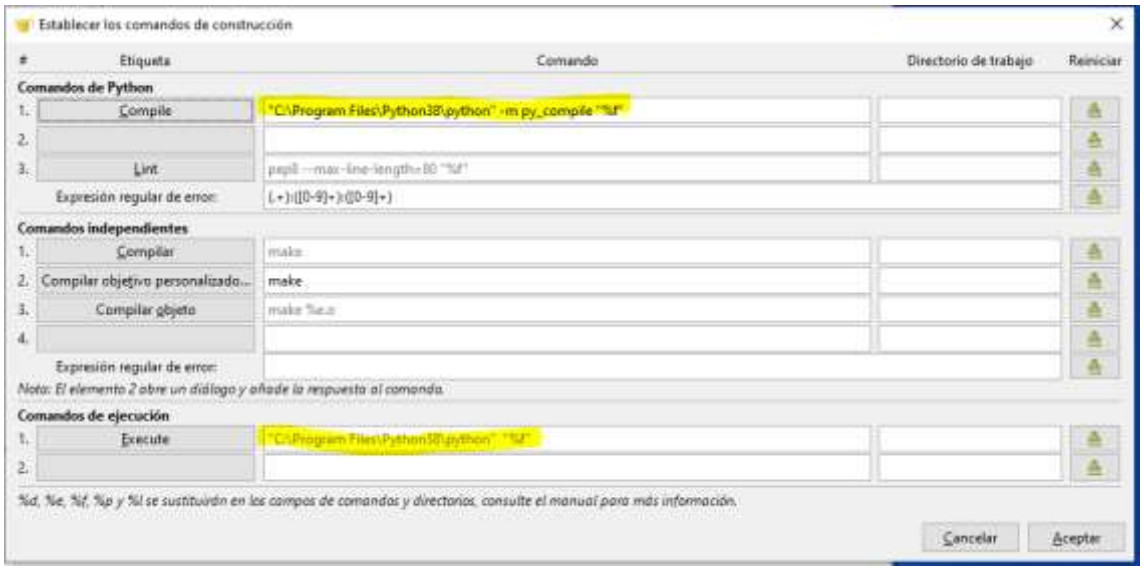
Y luego se resuelve y se presentan resultados. La variable status del pulp nos dirá si se ha resuelto al óptimo o si es infesible.

```
print('Inicio resolución')

prob.solve()

if prob.status=1:

    print(resuelto)
    for k in mesas:
        print("mesa", k)
        suma=0
        for i in agrupaciones:
            if x_var[i].varValue==NumMesa[k]:
                print("La agrupación ",i, "va en la mesa ",k)
                suma=suma+TamanyoAgrupacion[i]
        print("El numero de comensales de la mesa",k,"es"suma)


    for i in agrupaciones:
            print("la agrupacion",i,"va en la mesa", x_var[i].varValue)

    print()

    for k in mesas:
        for i in agrupaciones:
            if delta[(i,k)].varValue >= 1:
                print(i,k, delta[(i,k)].varValue)

    for i in agrupaciones:
        for j in agrupaciones:
                print(i,j, beta[(i,j)].varValue, beta [(j,i)].varValue, gamma[(i,j)].varValue)
else: print("no se ha podido resolver")
print('Y esto es todo amigos')
```

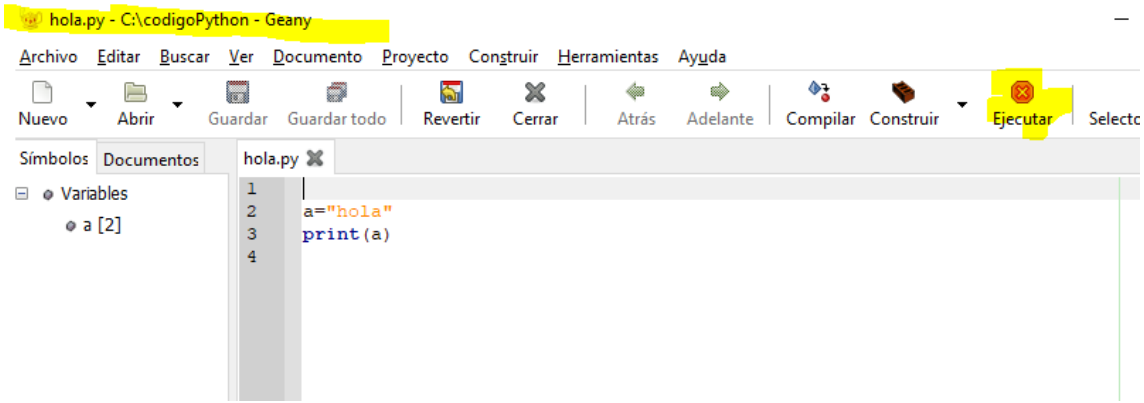
ANEXO 1: INSTALAR GEANY

https://www.geany.org/. Pero Geany no sabe donde está tu Python por eso hay que decirselohttp://introtopython.org/programming_environment_windows.html#installing_geany para ello hay que ir al compilar y modificar los comandos de construcción

Hay que poner la dirección de instalación Y luego al final se añade –m py_compiles “%f” en “compile y en execute sólo “%f”. Sinceramente no tengo ni idea de eso que significa.



Para comprobar que todo ha ido bien creo un trozo de código Python. Lo guardo en una carpeta que yo me creé en Python que es bastante sencilla (c:/codigoPython) . Si le das a ejecutar ha de funcionar. Y si no… pues no sé cómo ayudar.



ANEXO 2: INSTALANDO PAQUETES

El PIP es el paquete para instalar paquetes de Python.

Si no lo has hecho ya instálate el PIP

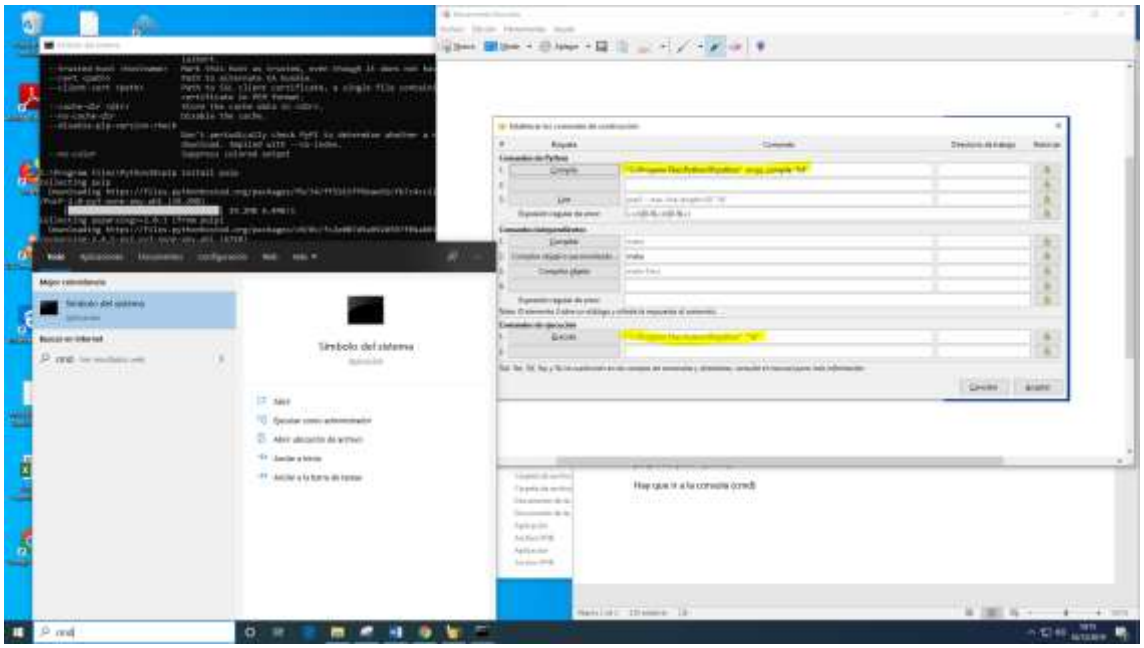
https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/

Los paquetes que creo que son interesantes son pulp, tkinter, mysql-connector, pandas, numpy

El menos utilizado de todos es PULP

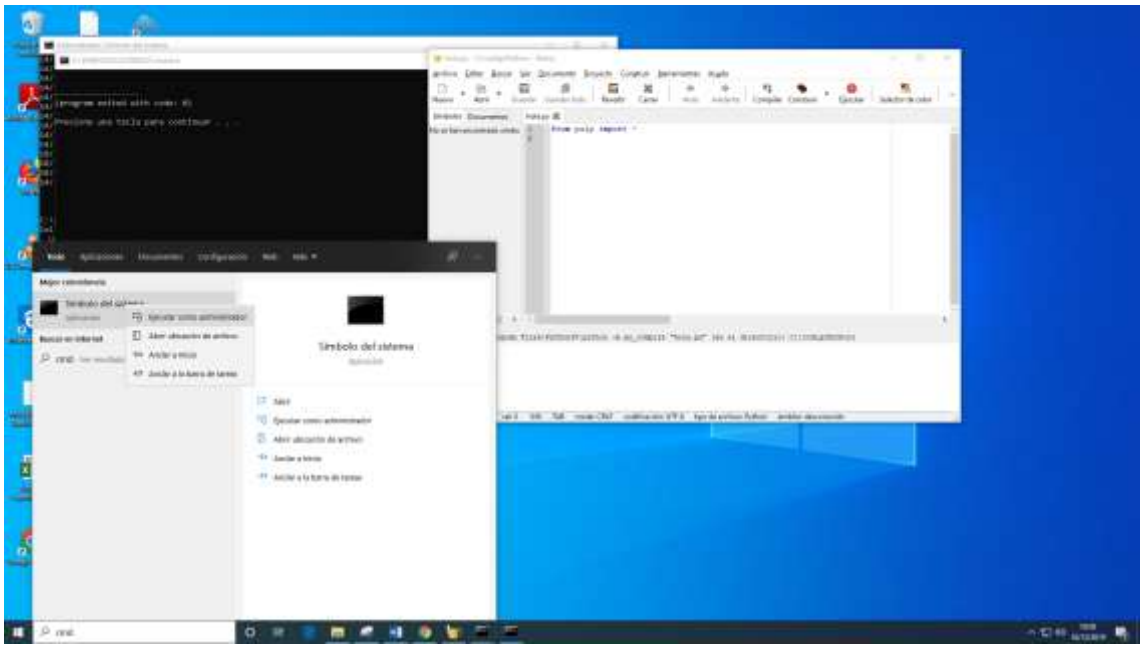
Para mí era importante que al hacerlo desde la consola (cmd) ejecutarlo como administrador.

Hay que abrir la consola (cmd) e ir al lugar donde está Python en mi caso en c:/program files/Python/Python 38 por la vía de usar comandos de msdos (lo cual me hace sentir joven).



Y entonces cuando estás en c:\Program Files\Python38 ejecutar **pip install pulp** como indican en https://pythonhosted.org/PuLP/main/installing_pulp_at_home.html

En mi caso he tenido que abrir el cmd como adminsitrador con el botón derecho sobre símbolo de sistema porque si no no tenía permisos



ANEXO 3: UN EJEMPLO PARA APRENDER PYTHON.

DESCRIBIR EL PROBLEMA:

Un intensificación de un Master con 6 asignaturas y 2 grupos en cada asignatura quiere hacer equipos de prácticas (número y tamaño diferentes) donde los alumnos, en la medida de lo posible los mismos alumnos no coincidan en diferentes grupos. Se podría tratar también de que fueran mixtas en género, o en procedencia de país o universidad.

El objetivo fundamental es que los alumnos interactúen entre ellos y generar una red lo más compacta posible.

Los alumnos están matriculados en más de una de las asignaturas de la intensificación. Las asignaturas tienen dos grupos. Cada asignatura tiene más de una sesión de prácticas (el mismo número para cada grupo).

Para algunas sesiones de prácticas los alumnos pueden ellos mismos hacer individualmente su equipo. Los profesores podrían forzar también algún emparejamiento en alguna sesión de prácticas. Los alumnos podrían justificar también la prohibición de algún tipo de emparejamiento por razones justificables.

MODELADO

Que nadie crea que he llegado hasta este modelado de tirón. He avanzado en espiral hasta dejarlo un poco compacto

Conjuntos:

Alumnos (i)

Asignaturas (j)

Grupos de Asignatura (g)

Sesiones de Asignatura (k)

Datos

Alumnos (email)

Asignaturas (código de Asignatura, nombre, número de grupos, numero de sesiones prácticas)

Grupos de Asignatura (código de Asignatura, código de Grupo, número de Alumnos)

Sesiones de Asignatura (código de Asignatura, código de Sesión, tamaño máximo de Equipo)

Alumnos por Grupo de Asignatura (código Asignatura, código Grupo, email)

Resultados

Alumnos por Equipo de Grupo de Asignatura (código Asignatura, código Grupo, código Sesión, código Equipo, Tamaño Grupo, email1, email2, email3…)

Origen de los datos:

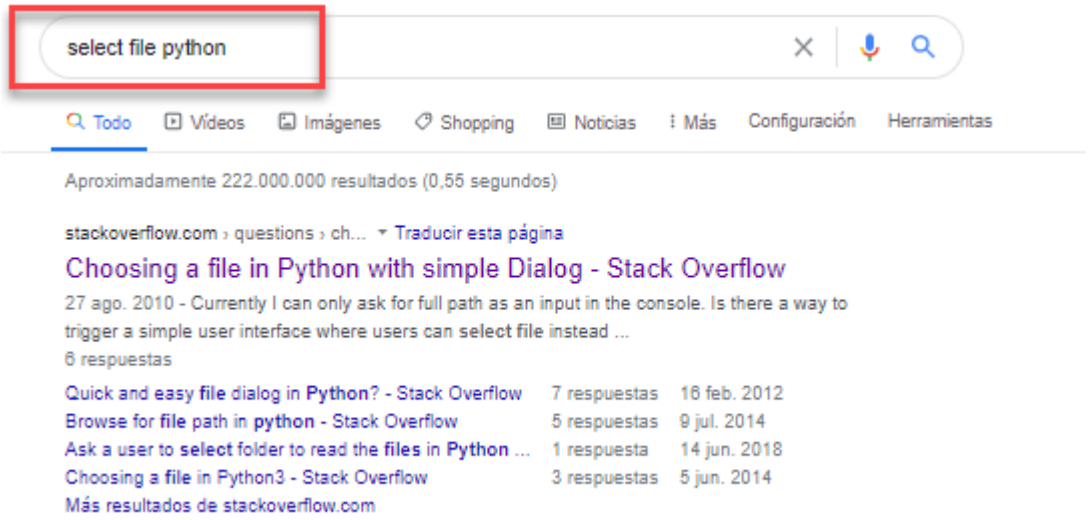
Tabla de Información solicitada a los responsables de las asignaturas con el número de sesiones que quiere tener y el tamaño máximo de equipo. La tengo en Excel.

Ficheros CSV para cada grupo de asignatura

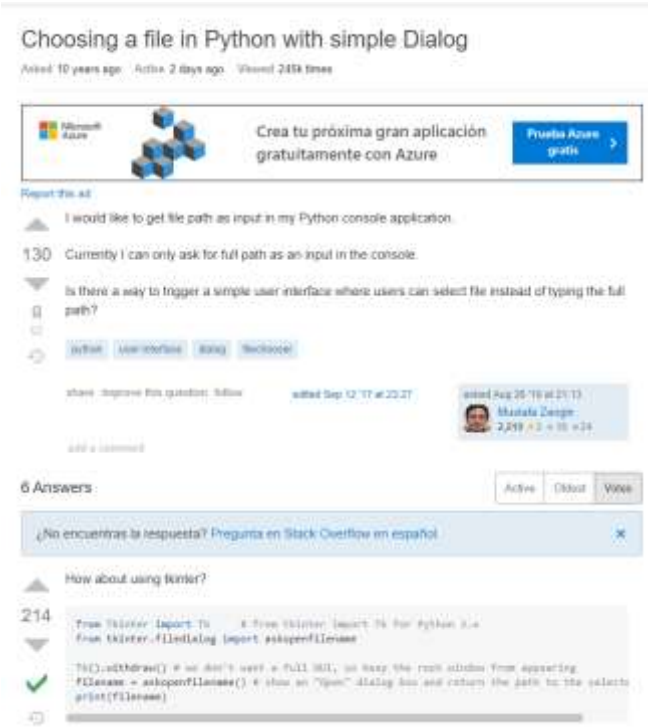
Vamos a ir construyendo el código poco a poco

Primero voy a cargar todas las asignaturas una a una seleccionándolas del directorio donde quiera que estén.

Para ello he buscado en el google “select file Python” (siempre pregunto en inglés) y el primer link me envía al stack overflow



Estos informáticos son la caña, te dice lo que has de hacer.

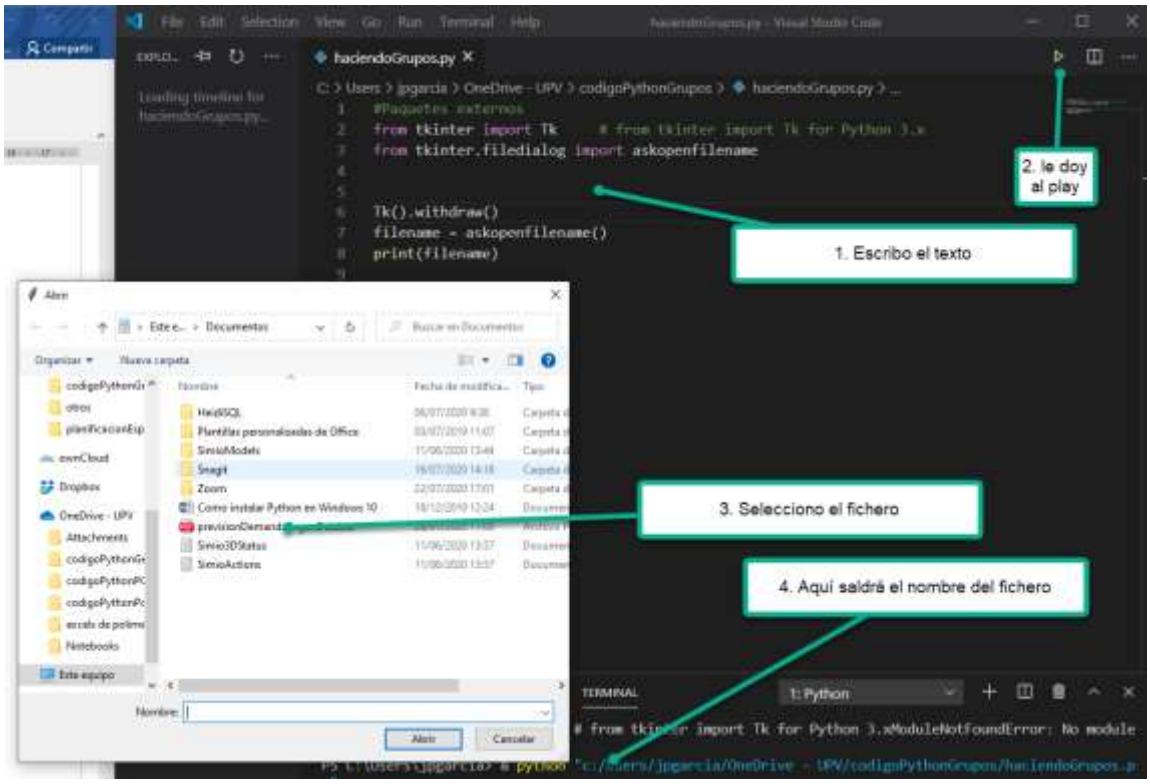


Es verdad que han dado por supuesto que yo sabía lo que era tkInter. Afortunadamente lo sabía (si no le hubiera preguntado a google). https://docs.python.org/3/library/tkinter.html

He tenido que hacer otra búsqueda para saber que tenía que instalar el paquete tk y como se instalaba. https://tutorial.recursospython.com/libreria-estandar/

Para eso he tenido que aprender que es pip. https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/

Total que he escrito lo que me han dicho que haga ejecuto y se abre una pantalla



Yo soy mayor y estudié programación con Pascal, así que no me gusta que haya cosas sueltas. Por eso creo una función. En Python se declaran con def y además el bloque que conforma la función se define por su tabulación (en Pascal era con begin y end pero aquí basta con indentar, ciertamente simplifica pero hay un problema estúpido con que hay quien indenta con tabuladores y quien lo hace con espacios, no he entendido muy bien la lógica pero es lo que hay)

Para que aprendamos un poco más he decidido abrir y leer el fichero que voy a seleccionar. Es un fichero csv que genera el poliformat automáticamente. Lo del csv (comma separated values) lo sé porque soy mayor, como usarlos en Python he ido a la web (google: leer y escribir ficheros csv en Python)

https://code.tutsplus.com/es/tutorials/how-to-read-and-write-csv-files-in-python--cms-29907

Un fichero csv es un fichero en el que los campos están separados por comas. Por algún motivo extraño Excel en español lo quiere separado por comas… De todos esos y otros misterios habrá que aprender un rato (y de algún misterio más)

Total que me he liado y me he puesto a programar

```
#Paquetes externos
from tkinter import Tk     # from tkinter import Tk for Python 3.x
from tkinter.filedialog import askopenfilename
import csv

#variables globales
apellidos=[]
nombre=[]
dni=[]
email=[]

def cargarAsignatura():
    Tk().withdraw()
    filename = askopenfilename()
    print(filename)
    numAlumnos=0
    with open(filename, newline='') as File:
        reader = csv.reader(File)
        for row in reader:
            print(row)
```

```
            apellidos.append(row[0])
            nombre.append(row[1])
            dni.append(row[2])
            email.append(row[3])
            numAlumnos+=1
    return numAlumnos

def sumarDNI(numDni):
    suma=0
    print(numDni)
    for cont in range(len(numDni)):
        if numDni[cont].isdigit():
            suma=suma+int(numDni[cont])
    return suma

N=cargarAsignatura()
print("el numero de alumnos es",N)
print("sus nombres son")
print(nombre)
print("sus apellidos son")
print(apellidos)
print("los digitos del dni suman")
suma2=0
for i in range(N):
    suma2+=sumarDNI(dni[i])
print(suma2)
```

Lo anterior se explica del siguiente modo:

1. Importo la librería tkInter y la del csv
2. Creo una lista de lsitas globales

3. Hago una función que carga una asignatura (elige el fichero) y luego asigna en las listas el valor de apellidos, nombres y demás. Esa función devuelve el número de alumnos
4. Defino otra función (se me ha ocurrido de repente) que suma los dígitos del dni. En principio era fácil, pero algunos de los compañeros tiene NIE, así que a veces sumaba X, y he tenido que preguntarle al google que "how to know if a carácter is a digit" y allí esta stackoverflow. (https://stackoverflow.com/questions/40097590/detect-whether-a-python-string-is-a-number-or-a-letter)
5. Entonces llamo a la función cargar asignatura (como va a devolver un número lo cargo en N)
6. Escribo diferentes cosas, y entonces es cuando se me va la pinza y trato de escribir cuanto suman los dígitos de los dnis

Y seguiría pero no tengo más tiempo y he de colgar el documento

ANEXO 4: OTRO PROBLEMA PARA PRACTICAR CON EL PULP

ENUNCIADO DEL PROBLEMA

Eres el coordinador de producción en un centro especial de empleo. A través dél, personas con discapacidad intelectual tienen acceso al trabajo a través de acuerdos con empresas. Las personas con DI (peones) desarrollan su actividad según el esquema de enclaves laborales. Un enclave laboral es un centro de trabajo ordinario donde un grupo de peones del Centro Especial de Empleo acude a trabajar haciendo el horario del centro de Trabajo supervisadas por un monitor. El monitor es también un trabajador que además sirve de enlace con la empresa.

Cada jueves, debes realizar el programa de trabajo para la siguiente semana. Dicho programa consiste en asignar los diferentes trabajadores a los enclaves para satisfacer la demanda de cada cliente expresada en horas de trabajo.

Los diferentes trabajadores se desempeñan de manera diferente en los diferentes centros. Por ello que se ha definido un medible al que se denomina afinidad que relaciona el desempeño (o incluso la voluntad del CEE de enviarlo a dicho enclave por ejemplo por razones formativas).

La legislación laboral española tiene ciertos requerimientos respecto a las horas máximas que una persona ha de trabajar por semana e incluso el reparto de los denominados fines de semana. Simplificando se puede decir que monitores y peones tienen diferentes capacidades máximas semanales (según convenio) y que en cualquier caso cada 11 días se debe disfrutar de dos días consecutivos de descanso según estatuto.

Con estos condicionantes se puede asumir que un trabajador estará en cualquier día asignado a un enclave, en fin de semana o tendrá un día libre. Evidentemente es posible asignar dos días libres seguidos, pero en ese caso el modelo debería asignarlo como fin de semana (para que, además de ser legal, lo parezca)

El condicionante de fin de semana llevó a la organización a planificar la actividad incluyendo dos fines de semana. Con el paso del tiempo se ha visto que (en función de los datos disponibles) la lógica de gestión es planificar la actividad los jueves, desde el viernes hasta el lunes de la semana siguiente, donde los primeros 3 días (de viernes a domingo) están congelados.

Hay que recoger el plan de la semana anterior (por saber si ha habido modificaciones de finde, días libres y demás) y sobretodo porque hay que mantener lo previsto para viernes, sábado y domingo.

De un plan de nivel superior que indica en qué lugar se espera que trabaje una persona en el periodo. Junto con información sobre la experiencia de cada trabajador se define un nivel de afinidad al puesto de trabajo.

Todas las personas tienen que tener algo que hacer, o estar de descanso semanal o tener día libre.

MODELADO

INDICES

Sea i el índice de los individuos (0..I-1)

Sea k el índice de los centros de trabajo (0..K-1)

Sea t cada uno de los días de la semana (0..T-1)

** Comenzamos en 0 porque el modelo lo vamos a implementar en Python y en ese lenguaje las listas empiezan con 0

PARÁMETROS

S	Las horas semana que como máximo puede trabajar un peon
R	Las horas semana que como máximo puede trabajar un monitor
F	El número de días en el horizonte de cálculo que están congeladas
T	El número de días que tiene el horizonte de cálculo

Γ(k)	Número máximo de peones por monitor aceptables en ese puesto de trabajo
H(k)	Las horas al día que se trabajan en el centro de trabajo k
Π (i)	Vale 1 si el trabajador i es Peon
Ω (i)	Vale 1 si el trabajador i es Monitor

A(i,k)	La afinidad asignada a la asignación de i a k
D(k,t)	Demanda expresada en horas de k en t

$\Phi(i,t)$	Lugar de trabajo previsto para i en t en el programa congelado (-1 si es día libre).
H(j,t)	Horas de jornada laboral en j,t (horas máximas que puede trabajar)
$\Omega(i,k)$	Si el trabajador k es de tipo i

(i1,i2)	Parejas de trabajadores que no se quiere que trabajen juntos

VARIABLES

$\delta_{i,k,t}$	1 si el trabajador i ha sido asignado a k en t
$\gamma_{i,t}$	1 si el día t es el primer día del fin de semana de i
$\varphi_{i,t}$	1 si el día t es un día libre para i

OBJETIVO

Maximizar la afinidad del plan, reduciendo al mínimo los días libres.

$$Max \sum_{i,k,t} A_{i,k}\delta_{i,k,t} - \sum_{k,t} \varphi_{i,t}$$

RESTRICCIONES

Cada día o bien trabaja en algún sitio o está de descanso semanal.

$$\gamma_{i,0} + \varphi_{i,0} + \sum_{k} \delta_{i,k,0} \leq 1 \qquad \forall i \qquad \text{(r1)}$$

$$\gamma_{i,t} + \gamma_{i,t-1} + \varphi_{i,t} + \sum_{k} \delta_{i,k,t} \leq 1 \quad \forall i,t \qquad \text{(r2)}$$

Los días del periodo congelado están asignados o bien como día libre o bien trabajando en algún lugar.

$$\gamma_{i,t} + \varphi_{i,t} = 1 \quad \forall i / \Phi_{i,0} = -1 \qquad \text{(r3)}$$

$$\gamma_{i,t} + \varphi_{i,t} = 1 \quad \forall i, 0 < t < F / \Phi_{i,t} = -1 \quad \text{(r4)}$$

$$\sum_{k} k\delta_{i,k,t} = \Phi_{i,t} \quad \forall i, t < F / \Phi_{i,t} > -1 \quad \text{(r5)}$$

El equipo a enviar tiene que tener un ratio peon-monitor menor que el máximo establecido.

$$\sum_i \Omega_i \delta_{i,k,t} \leq \Gamma_j \sum_i \Pi_i \delta_{i,k,t} \qquad \forall k,t \quad \text{(r6)}$$

Tiene que haber un fin de semana al menos cada 9 días

$$\sum_{\tau=0}^{8} \gamma_{i,t+\tau} \geq 1 \qquad \forall i, t < T-9 \quad \text{(r7)}$$

Pero no más de dos findes en el periodo planificado

$$\sum_t \gamma_{i,t} \leq 2 \qquad \forall i \qquad \text{(r8)}$$

Cada trabajador sólo trabaja 5 días de cada 7

$$\sum_{\tau=t}^{t+6} \left(\gamma_{i,\tau} + \gamma_{i,\tau+1} + \varphi_{i,\tau} \right) \geq 2 \quad \forall k,t \qquad \text{(r9)}$$

Entregar a cada cliente las horas que estaban previstas

$$\sum_i H_k \delta_{i,k,t} \geq Q_{k,t} \qquad \forall k,t \quad \text{(r10)}$$

Pero no más de un 20% más de las que estaban previstas (para no tener a la gente parada)

$$\sum_i H_k \delta_{i,k,t} \leq 1.2 Q_{k,t} \qquad \forall k,t \quad \text{(r11)}$$

Cada persona no trabaja más que las horas que estaba previsto que trabaje en una semana según su categoría

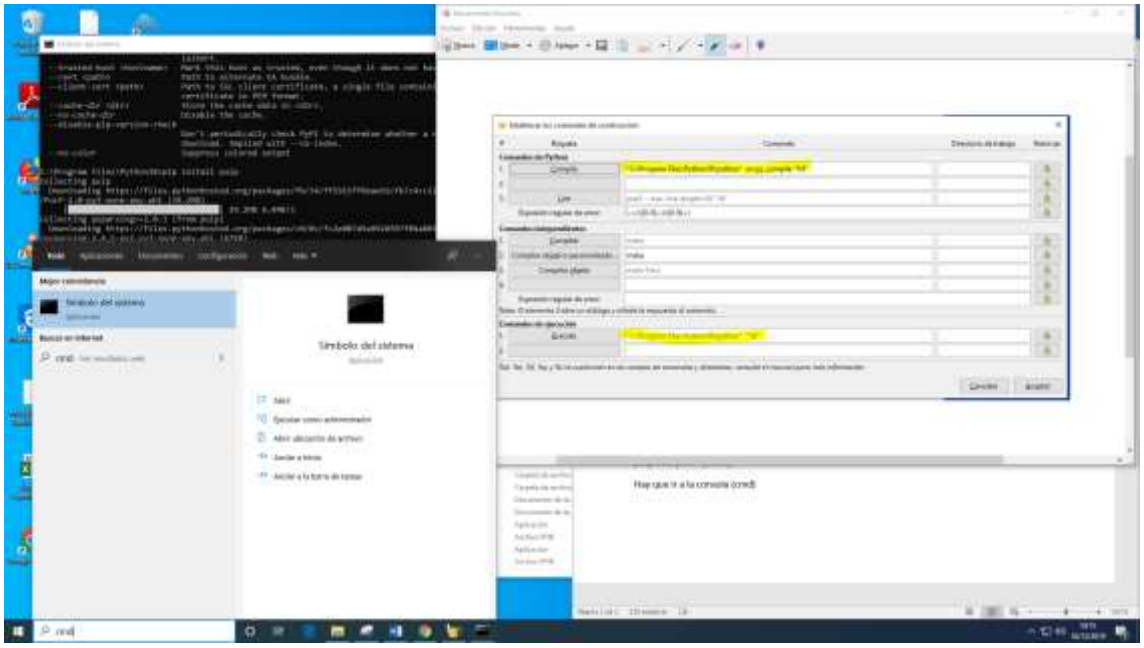
$$\sum_k H_{k,t} \delta_{i,k,t} \leq S_i \Pi_i + R\Omega_i \qquad \forall k,t \quad \text{(r12)}$$

INSTALAR PULP

La idea que tengo es utilizar pulp. Una librería que sirve para gestionar programación lineal.

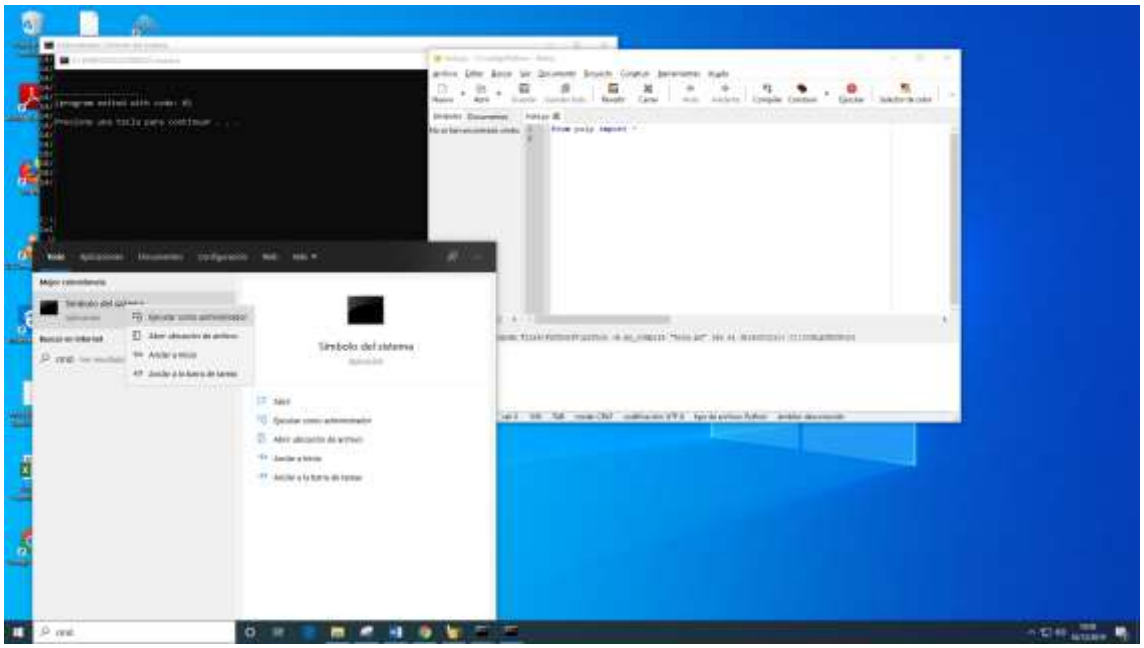
Para ello me he de descargar el paquete. Lo más fácil es utilizar pip (que es un instalador de paquetes para Python) que viene preinstalado.

Hay que abrir la consola (cmd) e ir al lugar donde está Python en mi caso en c:/program files/Python38 por la vía de usar comandos de msdos (lo cual me hace sentir joven). La ventana de comandos se ejecuta a través de cmd



Y entonces cuando estás en c:\Program Files\Python38 ejecutar **pip install pulp** como indican en https://pythonhosted.org/PuLP/main/installing_pulp_at_home.html

Importante: En mi caso he tenido que abrir el cmd como administrador con el botón derecho sobre "símbolo de sistema" porque si no no tenía permisos.



Como también quiero utilizar el mysql conector he aprovechado y me lo he instalado también pip install mysql-connector y el pandas pip install pandas (El pandas es una librería para análisis de datos que es una extensión de numpy (así que numpy también queda instalado). Numpy es un paquete para trabajar con números.

LA CARGA DE DATOS

Los datos podría cargarlos directamente aunque es verdad que es un poco tedioso.

```
HorasDemanda = [[42,0,0,0,42,42,42,42,42,0,0,42],
[60,0,0,0,60,60,60,60,60,0,0,60],
[9,0,0,0,9,9,9,9,0,0,0,9],
[12,24,60,60,24,12,12,12,12,24,24,12],
[35,0,0,0,35,35,35,35,35,0,0,35]]

Afinidad = [ [0 for k in centrosTrabajo] for i in trabajadores]
AfinidadPeon = [[100,10,0,70,50,50,40,10,22,11,40,10,22,11,10,710,150,140,100,10,0,70,50,40,10,22],
[50,40,10,22,23,34,45,67,3,50,40,10,22,11,65,56,3,2,50,40,10,22,23,34,45,67],
[70,50,50,40,10,22,11,40,10,22,3,65,56,3,2,40,50,13,70,50,40,10,22,40,50,13],
[0,70,50,40,10,3,50,40,10,22,11,65,56,3,2,22,30,40,0,70,50,40,10,22,30,40],
[10,0,70,50,40,10,50,40,10,22,11,22,3,65,56,3,2,60,10,0,70,50,40,10,22,60]]

HorasDiaCentro = {
0:7,
1:7.5,
2:7.5,
3:6,
4:7
}
RatioMax = {
0:5,
1:5,
2:5,
3:5,
4:5
}
```

Si te fijas los datos los ponemos en formato listas (o listas de listas si son n-dimensionales). El modo de acceder a cada elemento es HorasDemanda[k][t]

Pero hay otros datos (HorasDiaCentro) que los he asignado como *diccionarios*. Se accede igual. La diferencia entre unas estructuras y otras la explica bien esta web. https://www.tutorialpython.com/listas-en-python/

En lugar de escribirlo a mano podría haberlos leído de ficheros **csv** (hay un paquete para la lectura/escritura de ficheros). Pero preferí recogerlos directamente de una base de datos base de datos con motor mysql y he utilizado el paquete mysql.connector. Como algunos de los datos los tengo en formato fecha he utilizado el paquete **datetime**

```
1   from pulp import *
2   import csv
3   import mysql.connector
4   import datetime
5
```

Todos esos import sirven para decirle a nuestro programa que debemos considerar las funciones de los paquetes.

Me he conectado a la base de datos con el conector . No sé muy bien que es cursor pero todos los links visitados lo utilizan para ejecutar las queries (creo que es el que indica en qué tabla y línea de la tabla estás pero no estoy seguro)

```
11  mydb = mysql.connector.connect(host="[redacted]", user="[redacted]", passwd="[redacted]", database="[redacted]
12  mycursor = mydb.cursor()
```

Los datos en la tabla tb_CentrosTrabajo.

idCentroTrabaqjo	nombreCT	horasDia	ratioMaxima
0	Favara	7	5
1	Espurna	7	5
2	Smattex	8	4
3	Boga	6	4
4	Mapubli	8	4
5	Pincasa	7	3

Y descargo los datos de la tabla de centros de trabajo en las diferentes variables, listas y diccionarios

```
15  #Abro una tabla y cargo datos de centros de trabajo
16  mycursor.execute("SELECT * FROM tb_centrosTrabajo")
17  tbCentrosTrabajo = mycursor.fetchall()
18
19  numCentrosTrabajo = len(tbCentrosTrabajo)
20  #Ahora voy a crear un conjunto de diccionarios y  llenarlas (populate) con los datos de la tabla
21  nombreCentro = {}
22  ratioMax = {}
23  horasDiaCentro = {}
24  for k in range(numCentrosTrabajo):
25      nombreCentro[k]=tbCentrosTrabajo[k][1]
26      horasDiaCentro[k]=int(tbCentrosTrabajo[k][2])
27      ratioMax[k]=tbCentrosTrabajo[k][3]
28      #print(nombreCentro[k],horasDiaCentro[k],ratioMax[k])
```

Lo mismo aplica a los trabajadores que están en la tabla tb_trabajadores

idTrabajador	nombreTrabajador	esMonitor	esPeon	jornadaAnual
0	Pedro	0	1	1672
1	Paco	0	1	1672
2	Pepe	0	1	1672
3	Pa	0	1	1672
4	Pablo	0	1	1672
5	Pac	0	1	1672
6	Pura	0	1	1672
7	Palmo	0	1	1672
8	Pilar	0	1	1672
9	Pa?a	0	1	1672
10	Pau	0	1	1672
11	Pa	0	1	1672
12	Pepe	0	1	1672
13	Patricio	0	1	1672
14	Patricia	0	1	1672
15	Pepco	0	1	1672

Y que se recupera con estos valores

```
#Abro una tabla y cargo datos de trabajadores
mycursor.execute("SELECT * FROM tb_trabajadores")

tbTrabajadores = mycursor.fetchall()

numTrabajadores = len(tbTrabajadores)
#Ahora voy a crear un conjunto de diccionarioss y  llenarlas (populate) con los datos de la tabla
nombreTrabajador = {}
esMonitor = {}
esPeon = {}
jornadaAnual = {}
for i in range(numTrabajadores):
    nombreTrabajador[i]=tbTrabajadores[i][1]
    esMonitor[i]=tbTrabajadores[i][2]
    esPeon[i]=tbTrabajadores[i][3]
    jornadaAnual[i] = tbTrabajadores[i][4]
    #print(nombreTrabajador[i],esMonitor[i],esPeon[i],jornadaAnual[i])
```

Diferente es el modo de cargar datos de afinidades, porque es una matriz bidimensional que guardo en una tabla en tercera forma normal –que tampoco sé muy bien lo que es. (centro de trabajo, trabajador, valor de la afinidad).

idCentroTrabajo	idTrabajador	Afinidad
0	19	302
5	42	329
0	25	15
0	15	68
2	10	89
5	10	84
0	8	47
1	24	76
1	13	2
2	16	98
0	38	319

Y por tanto su recuperación es dferente.

```
48  #Abro una tabla y cargo datos de afinidades
49  mycursor.execute("SELECT * FROM tb_afinidad")
50
51  tbAfinidad = mycursor.fetchall()
52  afinidad = [ [0 for k in range(numTrabajadores)] for i in range(numCentrosTrabajo)]
53
54  #Ahora voy a llenar la lista afinidad con valores
55  for s in range(len(tbAfinidad)):
56      i=int(tbAfinidad[s][1])
57      k=int(tbAfinidad[s][0])
58      afinidad[k][i]=int(tbAfinidad[s][2])
59      #print(k,i,afinidad[k][i])
```

La tabla demanda es más complicada porque uno de los datos es la fecha, que evidentemente no es necesaria para el modelo de programación matemática, pero que será cómo obtenga la información.

idDia	idCentro	demanda
2019-12-20	0	70
2019-12-21	0	0
2019-12-22	0	0
2019-12-23	0	70
2019-12-24	0	70
2019-12-25	0	0
2019-12-26	0	70
2019-12-27	0	70
2019-12-28	0	0
2019-12-29	0	0
2019-12-30	0	70
2019-12-31	0	70
2019-12-20	1	35
2019-12-21	1	0
2019-12-22	1	0
2019-12-23	1	35

Y por tanto su recuperación es aún más compleja.

```
68
69  #Y ahora la tabla demanda
70  inicioPlan = datetime.datetime.strptime("20-12-2019", "%d-%m-%Y")
71  finPlan = datetime.datetime.strptime("31-12-2019", "%d-%m-%Y")
72  dias= [t for t in range(int((finPlan-inicioPlan).days)+1)]
73  numDias=len(dias)
74
75  mycursor.execute("SELECT * FROM tb_demanda")
76
77  tbDemanda = mycursor.fetchall()
78  horasDemanda = [ [0 for t in dias] for i in range(numCentrosTrabajo)]
79
80  for s in range(len(tbDemanda)):
81      t=(tbDemanda[s][0]-inicioPlan.date()).days
82      #el metodo .date lo he de añadir porque he creado un datetime y necesitaba un date
83
84      if ((t>=0) and (t<numDias)):
85          k=int(tbDemanda[s][1])
86          horasDemanda[k][t]=int(tbDemanda[s][2])
87      #el print de abajo es para ver si funciona
88      #print(k,t,horasDemanda[k][t])
89
```

Los valores de inicioPlan y finPlan, en realidad depende de cada ejecución por eso los asigno directamente

Es también una tabla en 3ª forma normal la del plan congelado. Quizá aquí lo particular es que la codificación que he decidido tener marca como -1 la opción de que el trabajador no tuviera ningún centro de trabajo asignado.

```
#Captura el plan prefijado
#Lo que voy a hacer es poner en libres todos los trabajadores y luego asignar los que están asignados
centroTrabajoAsignado=[[-1 for t in range(diasCongelado)] for i in range(numTrabajadores)]

mycursor.execute("SELECT * FROM tb_planPrevio")
tbProgramaCongelado = mycursor.fetchall()
for s in range(len(tbProgramaCongelado)):
    t=(tbProgramaCongelado[s][0]-inicioPlan.date()).days
    #el metodo .date lo he de añadir porque he creado un datetime y necesitaba un date
    if (t<diasCongelado):
        i=int(tbProgramaCongelado[s][1])
        centroTrabajoAsignado[i][t]=int(tbProgramaCongelado[s][2])
```

LA GENERACIÓN DEL MODELO EN PULP

Para definir el modelo se crea el objeto que he denominado “prob”.

Para definir las variables se utiliza el objeto LpVariable que utiliza un tipo de datos básico en Python que son las tuplas. (Fíjate que las tuplas de las variables se escriben [(i,k,t)]). https://uniwebsidad.com/libros/algoritmos-python/capitulo-7/tuplas

```
prob  = LpProblem("AsignacionCentros", LpMaximize)

deltaTrabajador = LpVariable.dicts("delta",
            [(i,k,t) for i in range(numTrabajadores) for k in range(numCentrosTrabajo) for t in dias],
            0,1, LpBinary)


gammaTrabajador = LpVariable.dicts("gamma",
            [(i,t) for i in range(numTrabajadores)  for t in dias],
            0,1, LpBinary)
phiTrabajador = LpVariable.dicts("phi",
            [(i,t) for i in range(numTrabajadores)  for t in dias],
            0,1, LpBinary)


prob += lpSum( afinidad[k][i]*deltaTrabajador[(i,k,t)] for k in range(numCentrosTrabajo) for i in range(numTr
        -lpSum(phiTrabajador[(i,t)] for i in range(numTrabajadores) for t in range(numDias))
```

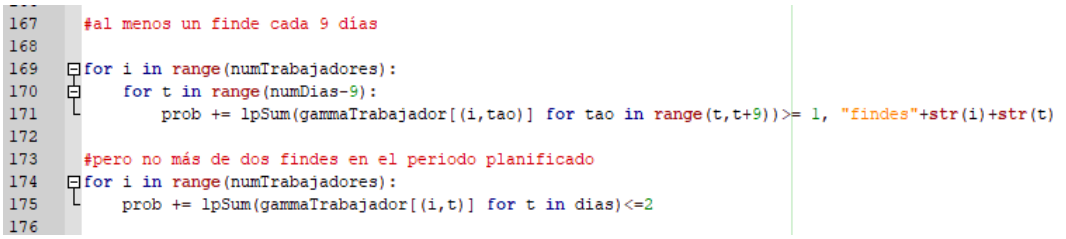
Y el objetivo (las últimas líneas del recorte anterior) son el objetivo.

Sabemos que son el objetivo porque no le ponemos ningún indicador de desigualdad a los sumatorios.

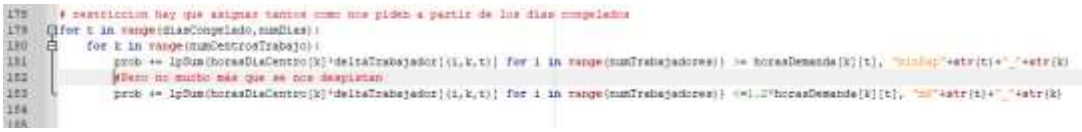
Como sí se le ponen a las restricciones

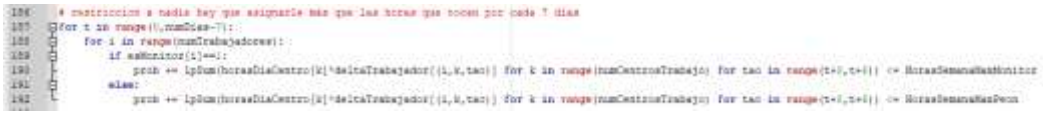
```
#Restriccion de los datos congelados de los primeros N periodos

for t in range(diasCongelado):
    for i in range(numTrabajadores):
        if centroTrabajoAsignado[i][t]==-1:
            if (t==0):
                prob += phiTrabajador[(i,t)] + gammaTrabajador[(i,t)]  == 1
            else:
                prob += phiTrabajador[(i,t)] + gammaTrabajador[(i,t)] +gammaTrabajador[(i,t-1)]   == 1
            for k in range(numCentrosTrabajo):
                prob += deltaTrabajador[(i,k,t)] == 0
        if centroTrabajoAsignado[i][t]>=0:
            for k in range(numCentrosTrabajo):
                if k==centroTrabajoAsignado[i][t]:
                    prob += deltaTrabajador[(i,k,t)] == 1
                else:
                    prob += deltaTrabajador[(i,k,t)] == 0
            prob += phiTrabajador[(i,t)] == 0
            prob += gammaTrabajador[(i,t)] == 0
```

```
# restriccion ratio Monitores Dia
for t in range(diasCongelado, numDias):
    for k in range(numCentrosTrabajo):
        prob += (lpSum(esPeon[i]*deltaTrabajador[(i,k,t)] for i in range(numTrabajadores)) \
        - lpSum(ratioMax[k]*esMonitor[i]*deltaTrabajador[(i,k,t)] for i in range(numTrabajadores))) <= 0

```



```
#al menos un finde cada 9 días

for i in range(numTrabajadores):
    for t in range(numDias-9):
        prob += lpSum(gammaTrabajador[(i,tao)] for tao in range(t,t+9))>= 1, "findes"+str(i)+str(t)

#pero no más de dos findes en el periodo planificado
for i in range(numTrabajadores):
    prob += lpSum(gammaTrabajador[(i,t)] for t in dias)<=2

```





Si el modelo lo queremos ver en lp (serviría para resolverlo y/o depurarlo en gusek podríamos utilizar la función prob.writeln("nombrefichero"

Pero si queremos resolver con prob.solve() es suficiente. Atención no olvidarse de los paréntesis de cierre.

Prob.status devuelve un número que nos indica si se se ha resuelto (https://www.coin-or.org/PuLP/constants.html) y con LpStatus conozco qué significa el numerito.

with status from solver.

LpStatus key	string value	numerical value
LpStatusOptimal	"Optimal"	1
LpStatusNotSolved	"Not Solved"	0
LpStatusInfeasible	"Infeasible"	-1
LpStatusUnbounded	"Unbounded"	-2
LpStatusUndefined	"Undefined"	-3

SALIDA DE RESULTADOS

Los resultados se pueden escribir en pantalla.

```
#la linea de abajo sirve para escribir en formato lp el modelo, eso facilita la depuración
prob.writeLP("costoEsquina.lp")

#Si no pones el parentesis en la funcion solve es como si no hubieras puesto nada
prob.solve()

#prob.Status devuelve un numero lpStatus es un diccionario
print(LpStatus[prob.status])

if prob.status!=-1:

    for t in dias:
        print("Findesemana",end=":")
        for i in range(numTrabajadores):
            if t>1 :
                if (gammaTrabajador[(i,t)].varValue+gammaTrabajador[(i,t-1)].varValue)>=1:
                    print(nombreTrabajador[i],end=',')
            else:
                if (gammaTrabajador[(i,t)].varValue)>=1:
                    print(nombreTrabajador[i],end=',')


        print()
        print("dia libre",end=":")
        for i in range(numTrabajadores):
            if phiTrabajador[(i,t)].varValue>=1:
                print(nombreTrabajador[i],end=',')
        print()


        for k in range(numCentrosTrabajo):
            listaMonitores=""
            listaPeones=""
            for i in range(numTrabajadores):
                a=deltaTrabajador[(i,k,t)].varValue
                if a>=1:
                    if esMonitor[i]==1: listaMonitores+=nombreTrabajador[i]+", "
                    if esPeon[i]==1: listaPeones+=nombreTrabajador[i]+", "
            print (t,nombreCentro[k],"monitores:",listaMonitores)
            print (t,nombreCentro[k],"peones:",listaPeones)

# da la impresión que una lpvariable a la que no se le asigna valor, tiene como valor None que no es un valor numérico
#De hecho me ha permitido comprobar en varias ocasiones que había restricciones que no estaban recorriendo bien el rango
# era una impresión errónea de None cuando no ha resuelto
```

Eso es relativamente fácil, quizá lo único a destacar es que las variables que el modelo no utiliza no las devuelve como cero sino como “None” lo cual puede complicar la lectura.

También podemos escribir los resultados en una tabla de la base de datos.

```
sql_borrar="Delete from tb_resultados"
mycursor.execute(sql_borrar)
mydb.commit()

for t in dias:
    for i in range(numTrabajadores):
        if gammaFindeTrabajador+gammaLibreTrabajador==1:
            sql = "INSERT INTO tb_resultados (idDia, idCentroTrabajo, idTrabajador, horasDia) VALUES (%s,%s,%s,%s)"
            fecha=inicioPlan+datetime.timedelta(days=t)
            val = (str(fecha),"DescansoSemanal",nombreTrabajador[i],"0")
            mycursor.execute(sql, val)
        for k in range(numCentrosTrabajo):
            if deltaTrabajador[(i,k,t)].varValue==1:
                sql = "INSERT INTO tb_resultados (idDia, idCentroTrabajo, idTrabajador, horasDia) VALUES (%s,%s,%s,%s)"
                fecha=inicioPlan+datetime.timedelta(days=t)
                val = (str(fecha),nombreCentro[k],nombreTrabajador[i],str(horasDiaCentro[k]))
                mycursor.execute(sql, val)
mydb.commit()
```

El mydb.commit() es necesario para que la escritura sea efectiva.

Borrar al principio la tabla es adecuado porque cada resolución es diferente, pero no es necesario para el funcionamiento.

La escritura en un csv es relativamente sencilla

https://realpython.com/python-csv/

```
with open('PlanPorTrabajador.csv',mode='w',newline='') as ficheroPlanTrabajador:
    f = csv.writer(ficheroPlanTrabajador, delimiter=";", quotechar='"',quoting=csv.QUOTE_NONNUMERIC)
    for i in range(numTrabajadores):
        plan=[nombreTrabajador[i]]
        for t in dias:
            if phiTrabajador[(i,t)].varValue>=1:
                plan.append("Libre")
            elif gammaTrabajador[(i,t)].varValue>=1:
                plan.append("Finde")
            elif t>0:
                if gammaTrabajador[(i,t-1)].varValue>=1:
                    plan.append("Finde")
                else:
                    for k in range(numCentrosTrabajo):
                        a=deltaTrabajador[(i,k,t)].varValue
                        if a>=1:
                            plan.append(nombreCentro[k])
                            if gammaTrabajador[(i,t-1)].varValue==1:
                                plan.append("Finde")
            else:
                for k in range(numCentrosTrabajo):
                    a=deltaTrabajador[(i,k,t)].varValue
                    if a>=1:
                        plan.append(nombreCentro[k])
```

Y escribir en Excel tampoco parece complicado https://www.geeksforgeeks.org/writing-excel-sheet-using-python/. Primero hay que importar la librería. En este caso utilizo la librería xlwt

```
C:\>cd codigoPython

C:\codigoPython>pip install xlwt
Collecting xlwt
  Downloading https://files.pythonhosted.org/packages/44/48/def306413b25c3d01753603b1a222a011b8621aed27c
/xlwt-1.3.0-py2.py3-none-any.whl (99kB)
    |████████████████████████████████| 102kB 547kB/s
Installing collected packages: xlwt
Successfully installed xlwt-1.3.0

C:\codigoPython>
```

Y luego simplemente hay que usarla

```
287
288  wb = xlwt.Workbook()
289  hoja1=wb.add_sheet('Hoja1')
290 for t in dias:
291      fecha=inicioPlan+datetime.timedelta(days=t)
292      hoja1.write(0,t+1,fecha)
293 for i in range(numTrabajadores):
294      hoja1.write(i+1,0,[nombreTrabajador[i]])
295      for t in dias:
296          if phiTrabajador[(i,t)].varValue>=1:
297              hoja1.write(i+1,t+1,"Libre")
298          elif gammaTrabajador[(i,t)].varValue>=1:
299              hoja1.write(i+1,t+1,"Finde")
300          elif t>0:
301              if gammaTrabajador[(i,t-1)].varValue>=1:
302                  hoja1.write(i+1,t+1,"Finde")
303              else:
304                  for k in range(numCentrosTrabajo):
305                      a=deltaTrabajador[(i,k,t)].varValue
306                      if a>=1:
307                          hoja1.write(i+1,t+1,nombreCentro[k])
308                          if gammaTrabajador[(i,t-1)].varValue==1:
309                              hoja1.write(i+1,t+1,"Finde")
310          else:
311              for k in range(numCentrosTrabajo):
312                  a=deltaTrabajador[(i,k,t)].varValue
313                  if a>=1:
314                      hoja1.write(i+1,t+1,nombreCentro[k])
315  wb.save("PlanTrabajador.xls")
316
```