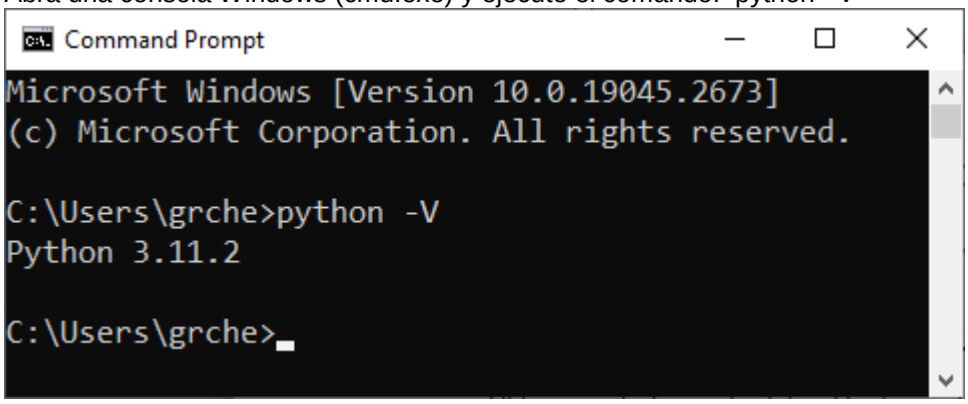
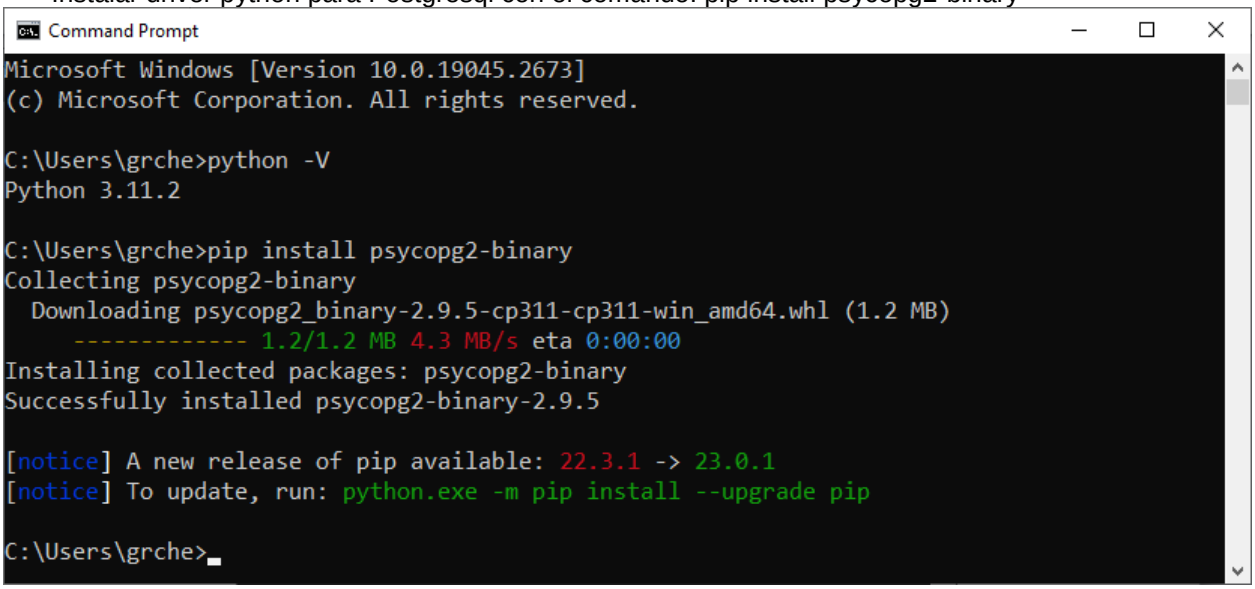
TRABAJO PRACTICO V – Desarrollo de Aplicación Client-Server con Python y Postgresql

Utilizamos la base de datos Postgresql implementada en el TP IV "Implementación de Base de Datos en Postgresql", ingrese a: https://www.grch.com.ar/docs/ap/

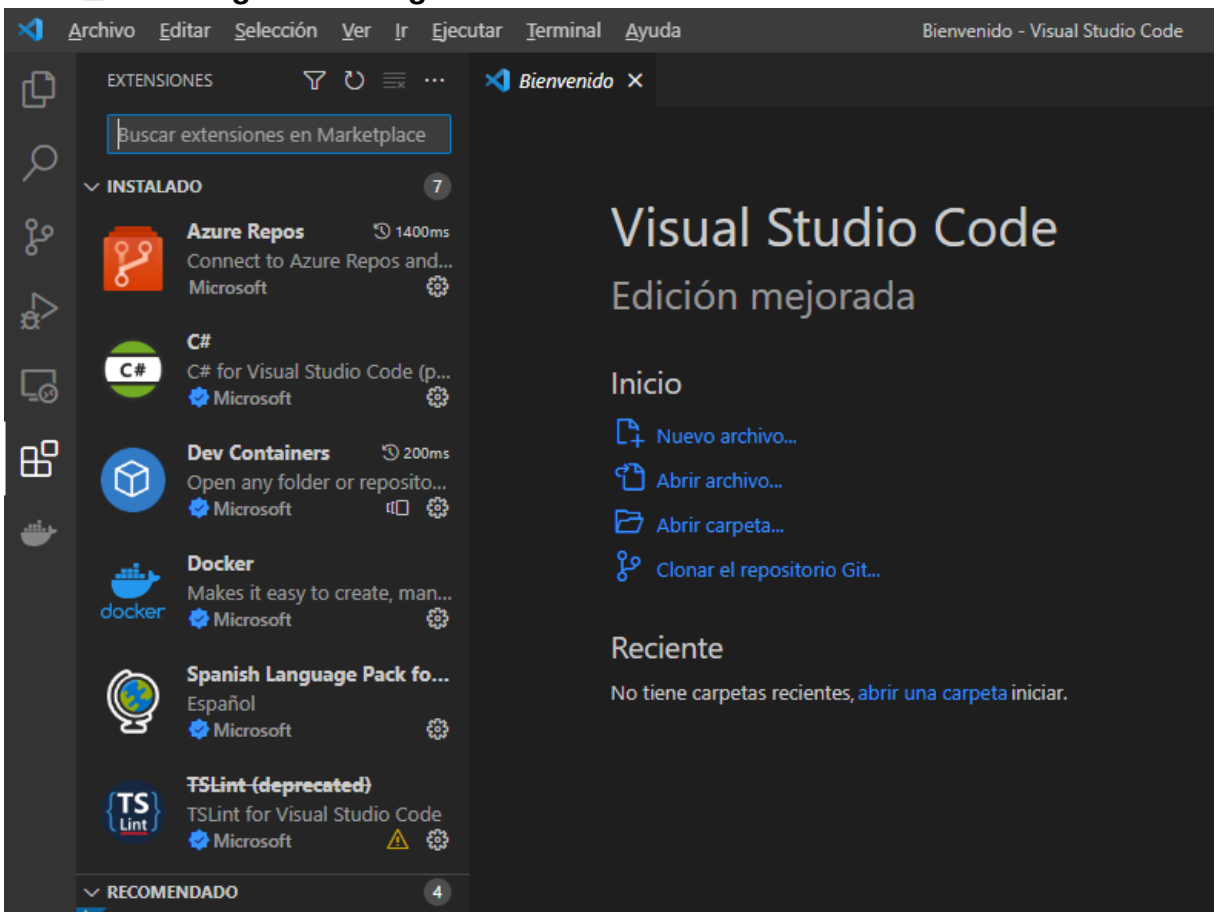
- Instalar Python 3.11 para Windows. Agregue a python.exe al PATH
- Abra una consola Windows (cmd.exe) y ejecute el comando: python –V



- Instalar driver python para Postgresql con el comando: pip install psycopg2-binary

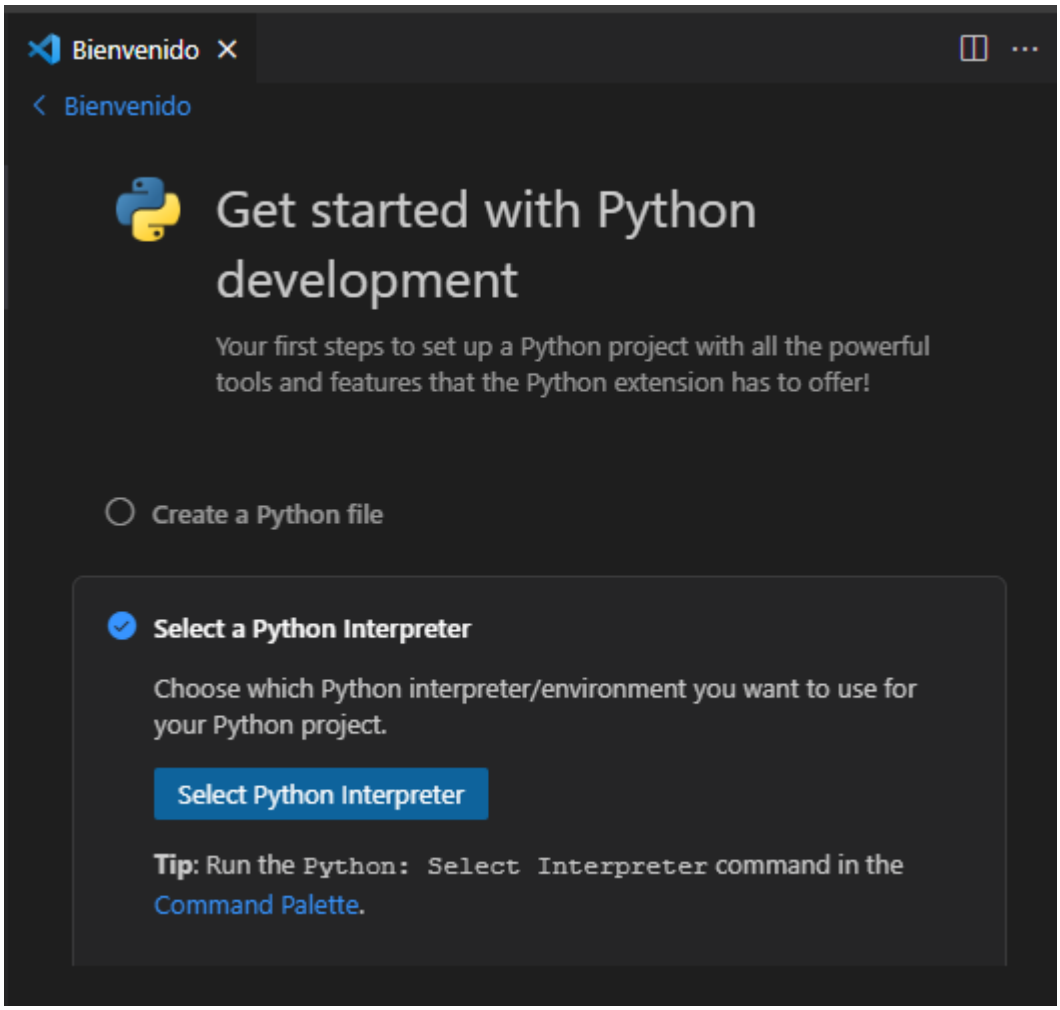


- Instalar Visual Studio Code (VSC) para Windows (última versión)
- Configurar VSC para trabajar con python[i], abra VSC, click en barra lateral izquierda extensiones:

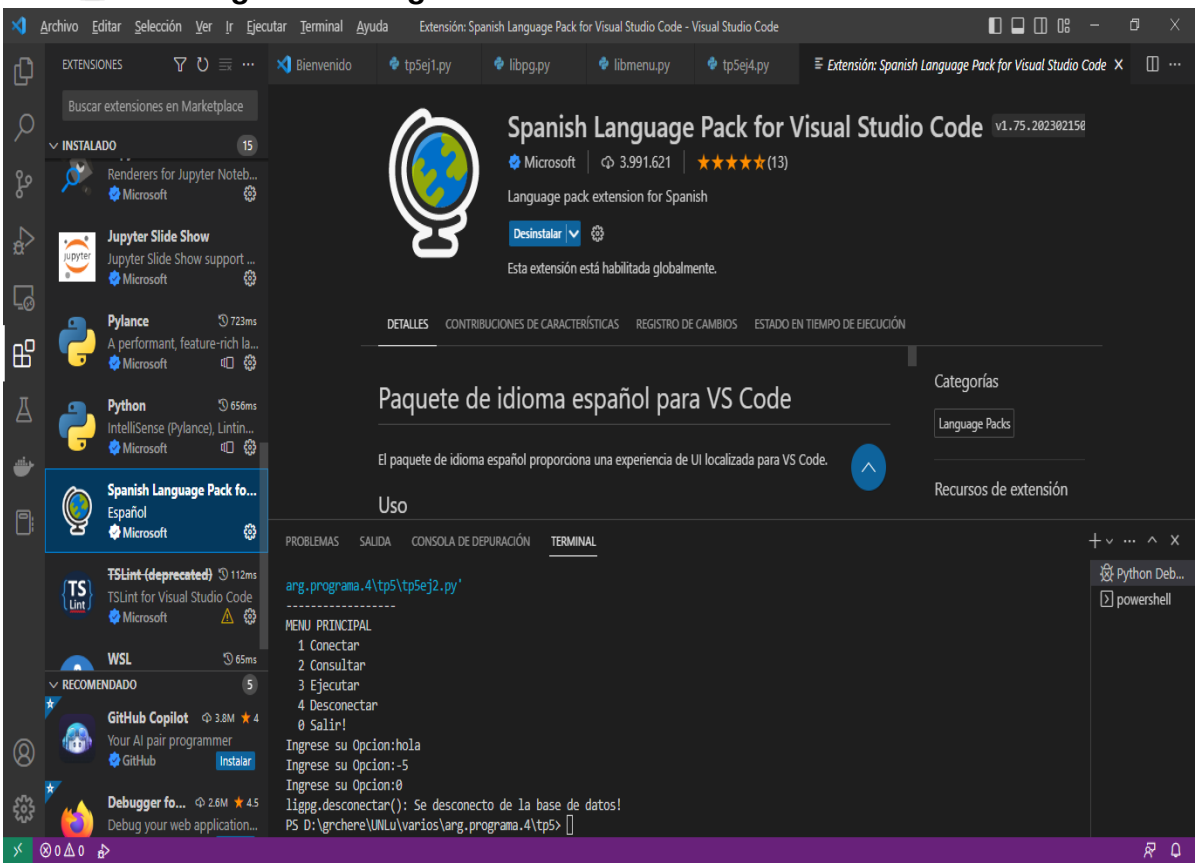


En buscar escriba “python”

instale la extensión “Python extension for VSC” de Microsoft, seleccione el intérprete python a utilizar (dando click en botón “Select a Python Interpreter” y seleccionando el archivo python.exe de la carpeta en donde Ud instaló python previamente:



Tal vez sea útil instalar soporte para idioma español en VSC, de la misma forma, en extensiones busque “Spanish Language Pack ..” e instálelo, aquí podemos ver la extensión instalada:



Programas fuentes mencionados aquí, disponibles en https://www.grch.com.ar/docs/ap/tp5/

1) Pruebo la conexión a la base de datos creada en el TP IV, supongamos que mi base de datos se llama tp4ej1, que el usuario postgres tiene como clave postgres (sino haga los cambios que necesite). Escriba el siguiente programa python y grábelo como tp5ej1.py ejecútelo desde VSC usando Ctrl+F5

```
import psycopg2
conexion=None
try:
    # Connecto a una base de datos existente
    conexion =
psycopg2.connect(user="postgres",password="postgres",host="localhost",port="54
32",database="tp4ej1")
    # Creo un cursor para hacer operaciones sobre la base de datos conectada
    cursor = conexion.cursor()
    # Imprimo deatos de la conexion con el servidor
    print("Informacion Servidor PostgreSQL:")
    print(conexion.get_dsn_parameters(), "\n")
    # Ejecuto una consulta SQL
    cursor.execute("SELECT version();")
    # Obtengo el resultado de la consulta SQL
    record = cursor.fetchone()
    print("Ud esta conectado a: ", record, "\n")

except Exception as error:
    print("Error conectando con PostgreSQL:",error)

finally:
    if (conexion):
        cursor.close()
        conexion.close()
        print("Conexion con PostgreSQL cerrada")
```

Si todo ok, un resultado posible de la ejecución de este programa:

```
Informacion Servidor PostgreSQL:
{'user': 'postgres', 'channel_binding': 'prefer', 'dbname': 'tp4ej1', 'host':
'localhost', 'port': '5432', 'options': '', 'sslmode': 'prefer',
'sslcompression': '0', 'sslsni': '1', 'ssl_min_protocol_version': 'TLSv1.2',
'gssencmode': 'disable', 'krbsrvname': 'postgres', 'target_session_attrs':
'any'}

Ud esta conectado a:  ('PostgreSQL 14.7, compiled by Visual C++ build 1914, 64-
bit',)

Conexion con PostgreSQL cerrada
```

2) Analizando el código del punto anterior, podemos observar las operaciones básicas sobre la base de datos: conectar, desconectar, consultar y ejecutar. Podemos agrupar estas operaciones en un programa python que se encargue de las operaciones básicas sobre la base de datos; a este programa lo llamé libpg.py :

```
import psycopg2

def conectar(usuario,clave,servidor,puerto,baseDeDato):
    conexion=None
    try:
        # Connecto a una base de datos existente
        conexion =
psycopg2.connect(user=usuario,password=clave,host=servidor,port=puerto,databas
e=baseDeDato)
        # Creo un cursor para hacer operaciones sobre la base de datos
conectada
        cursor = conexion.cursor()
        # Ejecuto una consulta SQL
        cursor.execute("SELECT version();")
        # Obtengo el resultado de la consulta SQL
        record = cursor.fetchone()
        print("ligpg.conectar(): Ud esta conectado a: ", record, "\n")
        return conexion

    except Exception as error:
        print("libpg.conectar(): Error conectando con PostgreSQL:",error)
        return conexion

    finally:
        return conexion

def desconectar(conexion):
    try:
        if conexion:
            conexion.close()
        print("ligpg.desconectar(): Se desconecto de la base de datos!")
    except Exception as error:
        print("libpg.desconectar(): Error :",error," desconectando base de
datos!")

def consultar(conexion,sql):
    tuplas = None
    try:
        if conexion:
            # Creo un cursor para hacer operaciones sobre la base de datos
conectada
            cursor = conexion.cursor()
            # Ejecuto una consulta SQL
            cursor.execute(sql)
            print("libpg.consultar(): Ejecute consulta!")
```

```
            tuplas = cursor.fetchall()
        return tuplas
    except Exception as error:
        print("libpg.consultar(): Error :",error," en consulta:",sql)
        return tuplas

def ejecutar(conexion,sql):
    try:
        if conexion:
            # Creo un cursor para hacer operaciones sobre la base de datos
conectada
            cursor = conexion.cursor()
            # Ejecuto una consulta SQL
            cursor.execute(sql)
            conexion.commit()
            print("libpg.ejecutar(): Ejecute consulta!")
    except Exception as error:
        print("libpg.ejecutar(): Error :",error," en consulta:",sql)
```

De esta forma, podemos usar la función conectar:

```
conn = conectar("postgres","postgres","localhost","5432","tp4ej1")
```

uso de la función consultar (sql select):

```
        sql = "SELECT * from localidad"
        tuplas = consultar(conn,sql)
        if (tuplas):
            print("Resultado Consulta:",len(tuplas)," tuplas!")
            for tupla in tuplas:
                print(f"Codigo Postal: {tupla[0]} Localidad: {tupla[1]}")
        else:
            print(f"La Consulta: {sql} no dio ningun resultado!")
```

uso de la función ejecutar (sql insert, update, delete, instrucciones DDL, etc):

```
        postal = ingresoNum("Codigo Postal:")
        localidad = input("Localidad:")
        sql = f"insert into localidad(cpostal,nombre)
values({postal},'{localidad}')"
        ejecutar(conn,sql)
```

uso de la función desconectar:

```
        desconectar(conn)
```

3) Seguramente vamos a necesitar algunas operaciones de ingreso por teclado, tal como: ingresoNum(), menú(), etc; con el mismo criterio vamos a agrupar todas las operaciones de teclado, de consola, en libmenu.py

```
def menu(titulo,opciones):
    print("------------------")
    print(titulo)
    n=1
    for opcion in opciones:
        print(f"{n:3} {opcion}")
        n=n+1
    print("  0 Salir!")
    opcion=-1
    while opcion < 0 or opcion >= n:
        opcion = ingresoNum("Ingrese su Opcion:")

    return opcion

def ingresoNum(prompt):
    while True:
        try:
            opcion = int(input(prompt))
            return opcion
        except Exception as error:
            opcion = -1
```

uso de la función menu:

```
opcion = menu("MENU PRINCIPAL",["Conectar","Consultar",
"Ejecutar","Desconectar"])
```

esto provocará que se muestre en consola un menú con las opciones: 1 Conectar, 2 Consultar, 3 Ejecutar, 4 Desconectar, 0 Salir y le pedirá al usuario que ingrese su opción, en caso de que ésta fuese equivocada o bien un dato no numérico, volverá a pedir el ingreso de la opción:

```
------------------
MENU PRINCIPAL
  1 Conectar
  2 Consultar
  3 Ejecutar
  4 Desconectar
  0 Salir!
Ingrese su Opcion:hola
Ingrese su Opcion:-5
Ingrese su Opcion:0
ligpg.desconectar(): Se desconecto de la base de datos!
```

4) Juntando todo, creamos el programa tp5ej4.py que usa las funciones implementadas en libpg.py y en libmenu.py :

```
import psycopg2
import libpg
import libmenu
from libpg import *
from libmenu import *

conn = None
opcion = -1
while opcion != 0:
    opcion = menu("MENU PRINCIPAL",["Conectar","Consultar",
"Ejecutar","Desconectar"])
    if opcion == 1:
        conn = conectar("postgres","postgres","localhost","5432","tp4ej1")
    elif opcion == 2:
        sql = "SELECT * from localidad"
        tuplas = consultar(conn,sql)
        if (tuplas):
            print("Resultado Consulta:",len(tuplas)," tuplas!")
            for tupla in tuplas:
                print(f"Codigo Postal: {tupla[0]} Localidad: {tupla[1]}")
        else:
            print(f"La Consulta: {sql} no dio ningun resultado!")
    elif opcion == 3:
        postal = ingresoNum("Codigo Postal:")
        localidad = input("Localidad:")
        sql = f"insert into localidad(cpostal,nombre)
values({postal},'{localidad}')"
        ejecutar(conn,sql)
    else:
        desconectar(conn)

exit(0)
```

5) Utilice pgAdmin4 para ingresar datos de prueba en las tablas: localidad, producto, provincia, zona, pc. Esta base de datos tiene varias reglas semánticas (ejemplo: (*) factura.gravado = sum(compuesta.subtotal) para productos gravados) que pueden implementarse utilizando las capacidades de programación que ofrezca la propia base de datos o bien, cuando ésta no lo provea, se debe realizar a través de la aplicación. La implementación de estas reglas a través de postgresql están fuera del alcance de este curso, por lo tanto, vamos a implementarlas utilizando python. Crear el programa tp5ej5.py con el siguiente menú de opciones:

1 Clientes
2 Facturas
3 Pagos
0 Salir

para el menú 1 Clientes, implemente las siguientes opciones:

1 Agregar nuevo Cliente
2 Borrar Cliente
3 Mostrar Clientes
0 Salir

Para agregar un nuevo cliente, tenga en cuenta todas las reglas (*) de la base de datos, ingrese los datos del cliente (cuando ingresa el código de cliente, verifique que no exista): código, razón, cuit, calle, número, piso, depto., contacto, código postal, provincia y zona. Dar de alta con saldo deudor igual a cero. Preguntar si el cliente es mayorista o minorista y luego de dar de alta en tabla cliente, dar de alta también en tabla mayorista o minorista, según corresponda. Si es mayorista, ingresar el % de descuento.

Para borrar un cliente, ingrese parte de la razón social y el programa intentará borrar todos los clientes cuya razón coincida o comience con lo ingresado. Esta operación dará error, si el cliente a borrar tiene facturas relacionadas.

Mostrar Clientes lista por pantalla los datos más importantes de los clientes (indicando si es mayorista o minorista).

6) Crear el programa tp5ej6.py (copiando tp5ej5.py). Para el menú 2 Facturas, implemente las siguientes opciones:

   1 Facturar
   2 Teminar Factura
   0 Salir

   Para Facturar, generar automáticamente el número de factura (último número más uno o bien 1 si no hay facturas previas), fecha de la factura es igual a la fecha actual, gravado = 0, nogravado = 0, monto = 0, preguntar si es factura a consumidor final (sin cliente relacionado) o bien indicar el código de cliente, si esta factura es con envío o no, pagada = NO, terminada = NO, totalpagado = 0. Dar de alta en factura e indicar al usuario el número de factura creado. Comenzar un loop para ir agregando productos a dicha factura, hasta que el usuario decida no ingresar más productos. El precio facturado de un producto no puede ser inferior al precio sugerido, la cantidad es siempre >= 1, el subtotal es igual a cantidad * precio, dar de alta en tabla "compuesta", sumar a columna gravado o nogravado, dependiendo si el producto es o no gravado, actualizar monto de la factura.

   Para terminar una factura, ingresar el número de la factura, verificar que exista y no este terminada. Poner terminada = SI, sumar monto de la factura al saldo deudor del cliente, si dicha factura esta relacionada con un cliente. Descontar del stock de producto las cantidades de los productos de esta factura.

7) Crear el programa tp5ej7.py (copiando tp5ej6.py). Para el menú 3 Pagos, mostrar en pantalla todas las facturas no pagas, terminadas, con saldo deudor (totalpagado < monto), el usuario ingresa el número de la factura, se asume la fecha de hoy para la fecha del pago, ingresa el monto del pago, el monto debe ser >= 1 y <= al saldo deudor de la factura, dar de alta en tabla pago, sumar al totalpagado de la factura, si el totalpagado es >= al monto de la factura, entonces poner a la factura como pagada = SI, restar del saldo deudor del cliente el monto del pago ingresado.

[i] Mas info en https://code.visualstudio.com/docs/python/python-tutorial