Curso de Python Avanzado

Scotiabank | **Belatrix**

Instructor: Alan Badillo Salas

Bienvenida

Bienvenidos al curso de **Python Avanzado** para Scotiabank, brindado por Belatrix.

En este curso aprenderás a desarrollar una programación avanzada con Python.

El curso está dirigido a profesionales relacionados al área de desarrollo, soporte y análisis de datos.

Se requieren conocimientos previos de Python Intermedio para poder cubrir satisfactoriamente este curso.

Temario

Módulo 1: Programación Funcional en Python

1. Introducción a la programación funcional
2. Funciones de orden superior
3. Lambdas y expresiones generadoras

Módulo 2: Diseño de Patrones Avanzados

1. Patrones de diseño comunes en Python
2. Aplicación práctica de patrones en el desarrollo de software

Módulo 3: Manipulación Eficiente de Datos

1. Uso avanzado de NumPy y Pandas
2. Operaciones vectorizadas y broadcasting

Módulo 4: Concurrencia y Paralelismo en Python

1. Hilos y procesos en Python
2. Multiprocessing y Asyncio

Módulo 5: Optimización de Código

1. Estrategias para mejorar el rendimiento
2. Perfilado de código y herramientas de optimización

Módulo 6: Seguridad en Desarrollo Python

1. Principios básicos de seguridad
2. Mejores prácticas de codificación segura

Módulo 7: Despliegue y Escalabilidad

1. Configuración de entornos de producción
2. Estrategias para escalabilidad horizontal y vertical

Módulo 4: Concurrencia y Paralelismo en Python

1. Hilos y procesos en Python
2. Multiprocessing y Asyncio

Los hilos y procesos en Python son esenciales para entender la programación concurrente y paralela, aspectos fundamentales en el desarrollo de aplicaciones eficientes y rápidas.

Hilos (Threads)

Los hilos son una forma de ejecutar múltiples tareas o subprocesos simultáneamente, permitiendo que una aplicación realice múltiples operaciones al mismo tiempo. En Python, el módulo `threading` es una de las maneras de implementar hilos.

Características de los Hilos:

- **Comparten memoria**: Los hilos de un mismo proceso comparten el mismo espacio de memoria, lo que facilita la comunicación entre ellos pero también requiere la gestión de acceso concurrente a recursos compartidos.
- **GIL (Global Interpreter Lock)**: Python tiene un mecanismo llamado GIL que permite que solo un hilo se ejecute en el intérprete en cualquier momento dado. Esto puede ser un limitante cuando se busca obtener beneficios de múltiples núcleos de CPU en operaciones intensivas de CPU.

Ejemplo de uso de `threading`:

```
import threading

def funcion_para_hilo():
    print("Hilo en ejecución")

# Crear un hilo
hilo = threading.Thread(target=funcion_para_hilo)
hilo.start()

# Esperar a que el hilo termine
hilo.join()
print("Hilo finalizado")
```

Procesos

Los procesos son instancias completamente independientes del intérprete de Python. Cada proceso tiene su propio espacio de memoria y su propio intérprete. Esto los hace más robustos en términos de aislamiento y seguridad, pero más costosos en términos de recursos y comunicación.

Características de los Procesos:

- **Memoria independiente**: Cada proceso tiene su propio espacio de memoria.
- **No afectados por el GIL**: Multiprocesamiento en Python permite aprovechar múltiples CPU y núcleos porque cada proceso tiene su propio intérprete de Python y, por lo tanto, su propio GIL.

Ejemplo de uso de `multiprocessing`:

```
from multiprocessing import Process

def funcion_para_proceso():
    print("Proceso en ejecución")

# Crear un proceso
proceso = Process(target=funcion_para_proceso)
proceso.start()

# Esperar a que el proceso termine
proceso.join()
print("Proceso finalizado")
```

Cuándo usar Hilos vs Procesos

- **Uso de Hilos**: Cuando la tarea implica operaciones de E/S o espera activa, y se puede beneficiar de la compartición fácil de información.
- **Uso de Procesos**: Cuando la tarea es intensiva en CPU y se desea evitar el GIL para mejorar el rendimiento utilizando múltiples núcleos de CPU.

En resumen, la elección entre hilos y procesos depende del tipo de tarea que necesitas realizar y de los recursos del sistema. En el contexto de Python, es crucial comprender estos conceptos para diseñar aplicaciones que sean tanto eficientes como efectivas.

```
import time
import random

def tarea(id):
  print(f"Se ha iniciado la tarea ({id})")
  # TODO: La tarea puede tardar algún tiempo en resolverse
  n = random.randint(1, 10)
  print(f"La tarea ({id}) tomará {n} segundos en completarse")
  time.sleep(n)
  print(f"Han transcurrido {n} segundos para la tarea ({id})")
  print(f"La tarea ({id}) ha finalizado")

start = time.time()

tarea(1)
tarea(2)
tarea(3)
tarea(4)

end = time.time()

delta = end - start

print(f"La tarea se resolvió en {delta} segundos")
```

```
    Se ha iniciado la tarea (1)
    La tarea (1) tomará 5 segundos en completarse
    Han transcurrido 5 segundos para la tarea (1)
    La tarea (1) ha finalizado
    Se ha iniciado la tarea (2)
    La tarea (2) tomará 5 segundos en completarse
    Han transcurrido 5 segundos para la tarea (2)
    La tarea (2) ha finalizado
    Se ha iniciado la tarea (3)
    La tarea (3) tomará 2 segundos en completarse
    Han transcurrido 2 segundos para la tarea (3)
    La tarea (3) ha finalizado
    Se ha iniciado la tarea (4)
    La tarea (4) tomará 2 segundos en completarse
    Han transcurrido 2 segundos para la tarea (4)
    La tarea (4) ha finalizado
    La tarea se resolvió en 14.016944408416748 segundos
```

```
import threading

t1 = threading.Thread(target=tarea, args=(1,))
t2 = threading.Thread(target=tarea, args=(2,))
t3 = threading.Thread(target=tarea, args=(3,))
```

```
t4 = threading.Thread(target=tarea, args=(4,))

start = time.time()

t1.start()
t2.start()

t1.join(), t2.join() # Esperar a que terminen la ejecución que ocurre dentro de c

end = time.time()

delta = end - start

print(f"La tareas 1 y 2 han terminado ({delta} segundos)")

start = time.time()

t3.start()
t4.start()

t3.join(), t4.join()

end = time.time()
delta = end - start

print(f"La tareas 3 y 4 han terminado ({delta} segundos)")
```

```
    Se ha iniciado la tarea (1)Se ha iniciado la tarea (2)
    La tarea (2) tomará 5 segundos en completarse

    La tarea (1) tomará 1 segundos en completarse
    Han transcurrido 1 segundos para la tarea (1)
    La tarea (1) ha finalizado
    Han transcurrido 5 segundos para la tarea (2)
    La tarea (2) ha finalizado
    La tareas 1 y 2 han terminado (5.007771253585815 segundos)
    Se ha iniciado la tarea (3)Se ha iniciado la tarea (4)
    La tarea (4) tomará 3 segundos en completarse

    La tarea (3) tomará 9 segundos en completarse
    Han transcurrido 3 segundos para la tarea (4)
    La tarea (4) ha finalizado
    Han transcurrido 9 segundos para la tarea (3)
    La tarea (3) ha finalizado
    La tareas 3 y 4 han terminado (9.013838291168213 segundos)
```

```
import random
import time

productos = [
    { "id": 1, "nombre": "Pepsi", "precio": 11.0 },
    { "id": 2, "nombre": "Fanta", "precio": 16.24 },
    { "id": 17, "nombre": "Coca-Cola", "precio": 17.5 },
    { "id": 19, "nombre": "Topo Chico", "precio": 19.9 },
    { "id": 20, "nombre": "Galletas Emperador", "precio": 13.89 },
    { "id": 25, "nombre": "Galletas Marías", "precio": 14.5 },
    { "id": 28, "nombre": "Gansito", "precio": 21.5 },
]

compras = [
    { "clienteId": 1, "productos": [17, 19, 25, 28] }
]

def clienteSeleccionaProductos(clienteId, k):
  productosSeleccionados = []
  for i in range(k):
    producto = random.choice(productos)
    productoId = producto["id"]
    productosSeleccionados.append(productoId)
    time.sleep(2)
  return {
      "clienteId": clienteId,
      "productos": productosSeleccionados
  }

compra1 = clienteSeleccionaProductos(3, 5)
compra2 = clienteSeleccionaProductos(4, 10)

print(compra1)
print(compra2)
```

```
    {'clienteId': 3, 'productos': [19, 17, 17, 2, 2]}
    {'clienteId': 4, 'productos': [20, 28, 1, 1, 25, 19, 19, 17, 25, 28]}
```

```
import random
import time

productos = [
    { "id": 1, "nombre": "Pepsi", "precio": 11.0 },
    { "id": 2, "nombre": "Fanta", "precio": 16.24 },
    { "id": 17, "nombre": "Coca-Cola", "precio": 17.5 },
    { "id": 19, "nombre": "Topo Chico", "precio": 19.9 },
    { "id": 20, "nombre": "Galletas Emperador", "precio": 13.89 },
    { "id": 25, "nombre": "Galletas Marías", "precio": 14.5 },
    { "id": 28, "nombre": "Gansito", "precio": 21.5 },
]

compras = [
    { "clienteId": 1, "productos": [17, 19, 25, 28] }
]

def clienteSeleccionaProductos(clienteId, k):
  productosSeleccionados = []
  for i in range(k):
    producto = random.choice(productos)
    productoId = producto["id"]
    productosSeleccionados.append(productoId)
    time.sleep(1)
  compras.append({
      "clienteId": clienteId,
      "productos": productosSeleccionados
  })

from threading import Thread

t1 = Thread(target=clienteSeleccionaProductos, args=(3, 3))
t2 = Thread(target=clienteSeleccionaProductos, args=(4, 2))

t1.start()
t2.start()

t1.join(), t2.join()

print(compras)
```

```
     [{'clienteId': 1, 'productos': [17, 19, 25, 28]}, {'clienteId': 4, 'productos
```

```
compras = []

hilos = []

# Registramos los 100 hilos (uno para cada cliente)
for clienteId in range(1, 101):
  k = random.randint(1, 10)
  t = Thread(target=clienteSeleccionaProductos, args=(clienteId, k))
  hilos.append(t)

# Inicializar cada hilo
for t in hilos:
  t.start()

# Esperar que cada hilo finalice
for t in hilos:
  t.join() # Espera a que el hilo `t` termine o continue

# Garantizamos que todos los hilos se completaron
print(compras) # 100 clientes -> tardan máximo 100 * 10s = 1,000s | 100 clientes
```

```
    [{'clienteId': 1, 'productos': [20]}, {'clienteId': 2, 'productos': [2]}, {'c
```

Ejemplo del uso de hilos para simular una venta

Vamos a ver cómo implementar un hilo en Python que utilice parámetros para simular una venta de cliente. En este ejemplo, vamos a usar el módulo `threading` para crear un hilo que maneje la información de una venta, como el nombre del cliente y el monto de la venta.

Ejemplo: Simulación de Venta de Cliente con Hilos

Primero, vamos a definir una función que represente la acción de procesar una venta. Esta función recibirá el nombre del cliente y el monto de la venta como parámetros. Luego, crearemos un hilo para cada venta simulada, pasando los argumentos necesarios a la función.

Aquí tienes el código:

```
import threading
import time

def procesar_venta(nombre_cliente, monto_venta):
    print(f"Procesando venta para {nombre_cliente} por un monto de ${monto_venta}")
    # Simulamos algún procesamiento que toma tiempo
```

```
    time.sleep(2)
    print(f"Venta procesada para {nombre_cliente}: ${monto_venta}")

# Lista de ventas a procesar
ventas = [
    ("Alice", 200),
    ("Bob", 150),
    ("Charlie", 300)
]

hilos = []

# Crear un hilo para cada venta
for nombre, monto in ventas:
    hilo = threading.Thread(target=procesar_venta, args=(nombre, monto))
    hilos.append(hilo)
    hilo.start()

# Esperar a que todos los hilos terminen
for hilo in hilos:
    hilo.join()

print("Todas las ventas han sido procesadas.")
```

Explicación del Código:

1. **Definición de la Función:** `procesar_venta` toma dos parámetros: `nombre_cliente` y `monto_venta`. Esta función simplemente imprime un mensaje para simular el procesamiento de una venta.
2. **Simulación de Procesamiento:** Usamos `time.sleep(2)` para simular que el procesamiento de la venta toma algo de tiempo. Esto ayuda a visualizar cómo los hilos operan en paralelo.
3. **Creación de Hilos:** Iteramos sobre la lista de `ventas` y creamos un hilo para cada una. Pasamos `nombre` y `monto` a la función `procesar_venta` a través del argumento `args`.
4. **Inicio de Hilos:** Cada hilo se inicia con `hilo.start()`.
5. **Espera de Hilos:** `hilo.join()` asegura que el programa principal espere a que todos los hilos terminen antes de imprimir que todas las ventas han sido procesadas.

Este enfoque es útil en escenarios reales donde, por ejemplo, podrías necesitar procesar múltiples transacciones de clientes de manera concurrente, optimizando así los tiempos de respuesta en sistemas que manejan un alto volumen de operaciones.

```
!pip install pymssql
```

```
Collecting pymssql
  Downloading pymssql-2.3.0-cp310-cp310-manylinux_2_28_x86_64.whl (4.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.6/4.6 MB 12.3 MB/s eta 0:00:00
Installing collected packages: pymssql
Successfully installed pymssql-2.3.0
```

```
import pymssql

conn = pymssql.connect("3.93.192.216", "test_curso", "TestCurso$123", "test_curso

def consultarFrutaAleatoria():
  cursor = conn.cursor()

  cursor.execute("select top(1) id, nombre, precio from frutas order by newid()")

  fruta = cursor.fetchone()

  cursor.close()

  return fruta # Devuelve una 3-tupla

consultarFrutaAleatoria() # Devuelve una 3-tupla (que se puede desacoplar en 3-va
```

```
(270, 'Papaya Texas', 123.37964355331724)
```

```
MAX_INTENTOS = 10

# Se requiere una función llamada clienteCompraFrutasAleatorias
# que reciba un identificador del cliente y un presupuesto
# para seleccionar aleatoriamente frutas (de la base de datos)
# e irlas acumulando hasta que se acabe su presupesto o el máximo de intentos.
# Si no es capaz de comprar una fruta con el presupuesto que le queda
# entonces seleccionará otra fruta e intentará acumular esa otra fruta
# pero gastará un intento.
# Cuando no queden intentos, debemos reportar
#  - el identificador del cliente
#  - el presupuesto original
#  - la lista de frutas que adquirió
#  - el precio total de todas esas frutas
#  - el presupuesto sobrante (lo que le sobró del presupuesto original)
```

```
def clienteCompraFrutasAleatorias(clienteId, presupuesto):
  # TODO 1: Definir las variables iniciales (estado inicial)
  intentos = 0                        # Este irá incrementando
  presupuestoOriginal = presupuesto  # Este se mantendrá con el valor original
  presupuestoActual = presupuesto    # Este irá disminuyendo por cada fruta adqui
  frutas = []                         # Esta Irá creciendo por cada fruta adquirid
  totalGastado = 0.0                  # Este irá incrementando por cada fruta adqu

  # TODO 4: Repetir el TODO 2, mientras no se haya superado el máximo de intentos
  while intentos < MAX_INTENTOS:
    # TODO 2: Consultar una fruta aleatoriamente (desde la base de datos)
    frutaConsultada = consultarFrutaAleatoria() # Es una 3-tupla (x, y, z)

    frutaId, nombre, precio = frutaConsultada # Desacomplamos la 3-tupla en 3-var

    # TODO 3: Verificar si el precio de la fruta está dentro del presupuesto actu
    if precio <= presupuestoActual: # Si puedo comprar la fruta a ese precio con
      frutas.append(frutaConsultada) # Registramos la fruta (agregamos la 3-tupla
      totalGastado = totalGastado + precio
      presupuestoActual = presupuestoActual - precio
      # presupuestoActual = presupuestoOriginal - totalGastado
    else: # No puedo comprar la fruta con el presupuesto actual
      intentos = intentos + 1

  # TODO 5: Reporte
  print(f"CLIENTE ({clienteId})")
  print("=" * 40)
  print("Presupuesto original:           ${:6.2f}".format(presupuestoOriginal))
  print("Total de frutas adquiridas:    {:8d}".format(len(frutas)))
  print("-" * 40)
  for frutaAdquirida in frutas:
    id, nombre, precio = frutaAdquirida # (frutaAdquirida => frutaConsultada 3-tu
    print("  [{:3d}]   {:<20s}  ${:6.2f}".format(id, nombre, precio))
  if len(frutas) == 0:
    print("  No hay frutas adquiridas")
  print("-" * 40)
  print("Total gastado:                  ${:6.2f}".format(totalGastado))
  print("Presupuesto sobrante:           ${:6.2f}".format(presupuestoActual))
  print("-" * 40)
  print("Eficiancia:                     %{:6.2f}".format(totalGastado / presupue
  print("=" * 40)
  print()

clienteCompraFrutasAleatorias(123, 200)
clienteCompraFrutasAleatorias(124, 400)
```

```
    CLIENTE (123)
    ========================================
    Presupuesto original:           $200.00
    Total de frutas adquiridas:           5
    ----------------------------------------
      [228]   Papaya del Caribe     $ 67.87
      [  3]   Papaya Tropical       $ 23.99
      [213]   Papaya Michoacán      $ 71.52
      [  3]   Papaya Tropical       $ 23.99
      [294]   Mango Huston         $  8.17
    ----------------------------------------
    Total gastado:                  $195.54
    Presupuesto sobrante:           $  4.46
    ----------------------------------------
    Eficiancia:                     % 97.77
    ========================================

    CLIENTE (124)
    ========================================
    Presupuesto original:           $400.00
    Total de frutas adquiridas:           5
    ----------------------------------------
      [263]   Guayaba Huston        $ 94.54
      [210]   Fresa Michoacán       $158.34
      [  1]   Manzana Golden        $ 88.56
      [216]   Fresa Costa           $ 32.64
      [209]   Fresa Huston          $ 23.60
    ----------------------------------------
    Total gastado:                  $397.68
    Presupuesto sobrante:           $  2.32
    ----------------------------------------
    Eficiancia:                     % 99.42
    ========================================



import random

for i in range(1000):
  clienteCompraFrutasAleatorias(i + 1, random.randint(50, 2000))

    CLIENTE (1)
    ========================================
    Presupuesto original:          $1689.00
    Total de frutas adquiridas:          29
    ----------------------------------------
      [224]   Piña Michoacán        $ 30.69
      [  4]   Mango Texas           $ 12.55
      [293]   Piña de Brasil        $192.04
      [250]   Piña Costa            $  4.48
      [216]   Fresa Costa           $ 32.64
      [611]   Guanabana 4           $123.00
      [327]   Pera del Caribe       $ 50.93
```

```
  [293]   Piña de Brasil        $192.04
  [216]   Fresa Costa           $ 32.64
  [214]   Fresa del Caribe      $ 83.52
  [239]   Pera Costa            $  4.59
  [  2]   Pera Huston           $ 44.22
  [269]   Mango Michoacán       $165.81
  [224]   Piña Michoacán        $ 30.69
  [218]   Guayaba Michoacán     $196.73
  [225]   Mango de Brasil       $ 49.98
  [  4]   Mango Texas           $ 12.55
  [240]   Pera Michoacán        $130.90
  [607]   Guanabana             $123.00
  [216]   Fresa Costa           $ 32.64
  [229]   Guayaba del Caribe    $  8.10
  [229]   Guayaba del Caribe    $  8.10
  [231]   Guayaba Costa         $  5.86
  [229]   Guayaba del Caribe    $  8.10
  [229]   Guayaba del Caribe    $  8.10
  [228]   Papaya del Caribe     $ 67.87
  [294]   Mango Huston          $  8.17
  [250]   Piña Costa            $  4.48
  [231]   Guayaba Costa         $  5.86
----------------------------------------
Total gastado:                  $1670.29
Presupuesto sobrante:           $ 18.71
----------------------------------------
Eficiancia:                     % 98.89
========================================

CLIENTE (2)
========================================
Presupuesto original:           $1719.00
Total de frutas adquiridas:           16
----------------------------------------
  [215]   Papaya de Brasil      $178.19
  [228]   Papaya del Caribe     $ 67.87
  [246]   Piña del Caribe       $195.67
  [215]   Papaya de Brasil      $178.19
  [218]   Guayaba Michoacán     $196.73
  [612]   Guanabana 5           $123.00
  [208]   Papaya Huston         $111.39
  [217]   Mango Costa           $ 18.54
  [235]   Piña Texas            $ 40.23
  [303]   Fresa Texas           $197.34
  [220]   Papaya Costa          $169.49
  [613]   Guanabana 6           $123.00
  [290]   Mango del Caribe      $ 40.44
```

```
import random
import threading

hilos = []
```

```
for i in range(1000):
  t = threading.Thread(target=clienteCompraFrutasAleatorias, args=(i + 1, random.
  hilos.append(t)

for t in hilos:
  t.start()

for t in hilos:
  t.join()

# Dividir los hilos en bloques de 4 máximo (o el número de núcleos de nuestro CPU
```

```
    Exception in thread Thread-347 (clienteCompraFrutasAleatorias):
    Traceback (most recent call last):
      File "src/pymssql/_pymssql.pyx", line 447, in pymssql._pymssql.Cursor.execut
    Exception in thread Thread-348 (clienteCompraFrutasAleatorias):
    Traceback (most recent call last):
      File "src/pymssql/_pymssql.pyx", line 447, in pymssql._pymssql.Cursor.execut
      File "src/pymssql/_mssql.pyx", line 1125, in pymssql._mssql.MSSQLConnection.
      File "src/pymssql/_mssql.pyx", line 1156, in pymssql._mssql.MSSQLConnection.
      File "src/pymssql/_mssql.pyx", line 1289, in pymssql._mssql.MSSQLConnection.
      File "src/pymssql/_mssql.pyx", line 1852, in pymssql._mssql.check_cancel_and
      File "src/pymssql/_mssql.pyx", line 1898, in pymssql._mssql.raise_MSSQLDatab
    pymssql._mssql.MSSQLDatabaseException  File "src/pymssql/_mssql.pyx", line 112
      File "src/pymssql/_mssql.pyx", line 1156, in pymssql._mssql.MSSQLConnection.
      File "src/pymssql/_mssql.pyx", line 1289, in pymssql._mssql.MSSQLConnection.
      File "src/pymssql/_mssql.pyx", line 1852, in pymssql._mssql.check_cancel_and
      File "src/pymssql/_mssql.pyx", line 1898, in pymssql._mssql.raise_MSSQLDatab
    pymssql._mssql.MSSQLDatabaseException: (0, b'Unknown error')

    During handling of the above exception, another exception occurred:

    Traceback (most recent call last):
      File "/usr/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
        self.run()
      File "/usr/lib/python3.10/threading.py", line 953, in run
        self._target(*self._args, **self._kwargs)
      File "<ipython-input-55-607099d4ef3b>", line 28, in clienteCompraFrutasAleat
      File "<ipython-input-34-973f02c3d241>", line 8, in consultarFrutaAleatoria
      File "src/pymssql/_pymssql.pyx", line 465, in pymssql._pymssql.Cursor.execut
    pymssql.exceptions: (0, b'Unknown error').OperationalError

    During handling of the above exception, another exception occurred:

    Traceback (most recent call last):
      File "/usr/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
    : (0, b'Unknown error')
    Exception in thread Thread-349 (clienteCompraFrutasAleatorias):
        self.run()
      File "/usr/lib/python3.10/threading.py", line 953, in run
    Traceback (most recent call last):
        self._target(*self._args, **self._kwargs)
```

```
  File "<ipython-input-55-607099d4ef3b>", line 28, in clienteCompraFrutasAleat
  File "<ipython-input-34-973f02c3d241>", line 8, in consultarFrutaAleatoria
  File "src/pymssql/_pymssql.pyx", line 465, in pymssql._pymssql.Cursor.execut
pymssql.exceptions.OperationalError: (0, b'Unknown error')
Exception in thread Thread-350 (clienteCompraFrutasAleatorias):
  File "src/pymssql/_pymssql.pyx", line 447, in pymssql._pymssql.Cursor.execut
Traceback (most recent call last):
  File "src/pymssql/_pymssql.pyx", line 447, in pymssql._pymssql.Cursor.execut
  File "src/pymssql/_mssql.pyx", line 1125, in pymssql._mssql.MSSQLConnection.
  File "src/pymssql/_mssql.pyx", line 1156, in pymssql._mssql.MSSQLConnection.
  File "src/pymssql/_mssql.pyx", line 1289, in pymssql._mssql.MSSQLConnection.
  File "src/pymssql/_mssql.pyx", line 1852, in pymssql._mssql.check_cancel_and
  File "src/pymssql/_mssql.pyx", line 1898, in pymssql._mssql.raise_MSSQLDatab
pymssql._mssql.MSSQLDatabaseException: (0, b'Unknown error')

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
  [219]  Guayaba Texas          $199.35
```

Multiprocessing

En Python, `multiprocessing` es un módulo que permite crear procesos, cada uno con su propio intérprete de Python y espacio de memoria. Esto significa que puedes realizar computación paralela en Python, superando el Global Interpreter Lock (GIL) que limita la ejecución simultánea de múltiples hilos en el mismo intérprete.

Características de `multiprocessing`:

- **Paralelismo real**: A diferencia de los hilos, que son limitados por el GIL, `multiprocessing` permite que cada proceso ejecute tareas en paralelo en diferentes núcleos del procesador.
- **Seguridad de memoria**: Dado que cada proceso tiene su propio espacio de memoria, los procesos no comparten datos a menos que explícitamente se configure para hacerlo (por ejemplo, usando `multiprocessing.Queue` o `multiprocessing.Pipe`).

Uso típico:

`multiprocessing` es especialmente útil para tareas que son intensivas en CPU, como procesamiento de imágenes, operaciones matemáticas pesadas, etc., donde el paralelismo puede significar una mejora significativa en el rendimiento.

Ejemplo de `multiprocessing`:

```
from multiprocessing import Process
```

```
def tarea_proceso(num):
    print(f"Proceso {num} ejecutándose")

procesos = []
for i in range(5):  # Crear 5 procesos
    p = Process(target=tarea_proceso, args=(i,))
    procesos.append(p)
    p.start()

for p in procesos:
    p.join()  # Esperar a que todos los procesos terminen

print("Todos los procesos han finalizado.")
```

Asyncio

`asyncio` es un módulo de Python que proporciona un marco para escribir código concurrente utilizando la sintaxis `async` / `await` . Es utilizado principalmente para escribir código asincrónico para operaciones basadas en E/S (como accesos a la web, bases de datos y archivos).

Características de `asyncio`:

- **Asincronía**: Permite que el código se ejecute de manera no bloqueante mediante el uso de corutinas que se suspenden y reanudan en puntos de espera.
- **Gestión de eventos**: Usa un bucle de eventos para manejar todas las operaciones asincrónicas. El bucle de eventos es el núcleo de cada programa asyncio, proporcionando una manera de programar y gestionar un alto nivel de concurrencia.

Uso típico:

`asyncio` es ideal para aplicaciones de red, servidores web, y otras operaciones que son I/O-bound más que CPU-bound, donde la asincronía puede reducir el tiempo de espera y aumentar la eficiencia.

Ejemplo de `asyncio`:

```
import asyncio

async def tarea_asincrona(num):
    print(f"Comenzando tarea {num}")
    await asyncio.sleep(1)  # Simular I/O con sleep
```

```
    print(f"Tarea {num} completada")

async def main():
    # Ejecutar tres tareas asincrónicas
    await asyncio.gather(
        tarea_asincrona(1),
        tarea_asincrona(2),
        tarea_asincrona(3)
    )

# Ejecutar el bucle de eventos
asyncio.run(main())
```

Comparación entre `multiprocessing` y `asyncio`

- **Cuándo usar `multiprocessing`**: Cuando necesitas superar el GIL para realizar computación intensiva en CPU que puede beneficiarse de múltiples núcleos.
- **Cuándo usar `asyncio`**: Para aplicaciones que realizan muchas operaciones de E/S y donde la capacidad de manejar muchas conexiones simultáneas es crucial.

Ambas tecnologías tienen su lugar en el ecosistema de Python y se pueden combinar para aprovechar sus respectivas ventajas en aplicaciones complejas.

```
      [229]   Guayaba del Caribe    $  8.10
```

Ejemplo de la descarga en paralelo con Threading

```
      [607]   Guanabana             $123.00
```

```
import requests
import time
from threading import Thread

def tarea_hilo(num):
    print(f"Hilo {num} ejecutándose")

    # URL del archivo que quieres descargar
    url = 'https://archive.ics.uci.edu/static/public/360/air+quality.zip'

    # Realizar la petición GET para obtener el contenido del archivo
    respuesta = requests.get(url)

    # Verificar si la petición fue exitosa (código de estado 200)
    if respuesta.status_code == 200:
        # Escribir el contenido descargado en un archivo local
        with open(f'source_{num}.zip', 'wb') as archivo:
```

```
            archivo.write(respuesta.content)
        print("¡Descarga exitosa!")
    else:
        print("Error al descargar el archivo:", respuesta.status_code)
    print(f"Hilo {num} terminado")

start = time.time()
hilos = []
for i in range(5):  # Crear 5 procesos
    t = Thread(target=tarea_hilo, args=(i,))
    hilos.append(t)
    t.start()

for t in hilos:
    t.join()  # Esperar a que todos los procesos terminen

end = time.time()

delta = end - start

print(f"Todos los hilos han finalizado en {delta} s")
```

```
Hilo 0 ejecutándose
Hilo 1 ejecutándose
Hilo 2 ejecutándose
Hilo 3 ejecutándose
Hilo 4 ejecutándose
¡Descarga exitosa!
Hilo 0 terminado
¡Descarga exitosa!
Hilo 2 terminado
¡Descarga exitosa!
Hilo 4 terminado
¡Descarga exitosa!
Hilo 3 terminado
¡Descarga exitosa!
Hilo 1 terminado
Todos los hilos han finalizado en 3.634617328643799 s

Total de frutas adquiridas:          17
```

Ejemplo de la descarga en paralelo con Multiprocessing

```
      [228]   Papaya del Caribe     $ 67.87
```

```
import requests
import time
from multiprocessing import Process

def tarea_proceso(num):
    print(f"Proceso {num} ejecutándose")
```

```
    # URL del archivo que quieres descargar
    url = 'https://archive.ics.uci.edu/static/public/360/air+quality.zip'

    # Realizar la petición GET para obtener el contenido del archivo
    respuesta = requests.get(url)

    # Verificar si la petición fue exitosa (código de estado 200)
    if respuesta.status_code == 200:
        # Escribir el contenido descargado en un archivo local
        with open(f'source_{num}.zip', 'wb') as archivo:
            archivo.write(respuesta.content)
        print("¡Descarga exitosa!")
    else:
        print("Error al descargar el archivo:", respuesta.status_code)
    print(f"Proceso {num} terminado")

start = time.time()
procesos = []
for i in range(5):  # Crear 5 procesos
    p = Process(target=tarea_proceso, args=(i,))
    procesos.append(p)
    p.start()

for p in procesos:
    p.join()  # Esperar a que todos los procesos terminen

end = time.time()

delta = end - start

print(f"Todos los procesos han finalizado en {delta} s")
```

```
Proceso 0 ejecutándose
Proceso 1 ejecutándose
Proceso 2 ejecutándose
Proceso 3 ejecutándose
Proceso 4 ejecutándose
¡Descarga exitosa!
Proceso 2 terminado
¡Descarga exitosa!
Proceso 0 terminado
¡Descarga exitosa!
Proceso 4 terminado
¡Descarga exitosa!
Proceso 3 terminado¡Descarga exitosa!

Proceso 1 terminado
Todos los procesos han finalizado en 2.4124584197998047 s
```

[225] Mango de Brasil $ 49.98

[219] Mango de Brasil $ 19.99

Ejemplo de la descarga en secuencial con Asyncio

[220] Papaya Costa $169.49

```
import requests
import asyncio
import time

async def tarea_asincrona(num):
    print(f"Tarea {num} ejecutándose")

    # URL del archivo que quieres descargar
    url = 'https://archive.ics.uci.edu/static/public/360/air+quality.zip'

    # Realizar la petición GET para obtener el contenido del archivo
    respuesta = requests.get(url)

    # Verificar si la petición fue exitosa (código de estado 200)
    if respuesta.status_code == 200:
        # Escribir el contenido descargado en un archivo local
        with open(f'source_{num}.zip', 'wb') as archivo:
            archivo.write(respuesta.content)
        print("¡Descarga exitosa!")
    else:
        print("Error al descargar el archivo:", respuesta.status_code)
    print(f"Tarea {num} terminada")

async def main():
    start = time.time()
    print("Ejecutando tareas")

    # Ejecutar tareas asincrónicas
    await asyncio.gather(
      tarea_asincrona(1),
      tarea_asincrona(2),
      tarea_asincrona(3),
      tarea_asincrona(4),
      tarea_asincrona(5),
    )

    end = time.time()

    delta = end - start

    print(f"Todas los tareas han finalizado en {delta} s")

# Ejecutar el bucle de eventos
# asyncio.run(main()) # <<< FUERA DE COLAB
```

```
await main()
```

```
Ejecutando tareas
Tarea 1 ejecutándose
¡Descarga exitosa!
Tarea 1 terminada
Tarea 2 ejecutándose
¡Descarga exitosa!
Tarea 2 terminada
Tarea 3 ejecutándose
¡Descarga exitosa!
Tarea 3 terminada
Tarea 4 ejecutándose
¡Descarga exitosa!
Tarea 4 terminada
Tarea 5 ejecutándose
¡Descarga exitosa!
Tarea 5 terminada
Todas los tareas han finalizado en 8.887385368347168 s
----------------------------------------
```

Ejemplo de la descarga en paralelo con Asyncio

```
Eficiencia:                    % 98.66
```

```
import asyncio
import time
import aiohttp

async def tarea_asincrona(num):
    print(f"Tarea {num} ejecutándose")

    url = 'https://archive.ics.uci.edu/static/public/360/air+quality.zip'

    async with aiohttp.ClientSession() as session:
        async with session.get(url) as respuesta:
            if respuesta.status == 200:
                # Escribir el contenido descargado en un archivo local
                contenido = await respuesta.read()
                with open(f'source_{num}.zip', 'wb') as archivo:
                    archivo.write(contenido)
                print("¡Descarga exitosa!")
            else:
                print(f"Error al descargar el archivo {num}: {respuesta.status}")

    print(f"Tarea {num} terminada")

async def main():
    start = time.time()
```

```
    print("Ejecutando tareas")

    # Ejecutar tareas asincrónicas
    await asyncio.gather(
        tarea_asincrona(1),
        tarea_asincrona(2),
        tarea_asincrona(3),
        tarea_asincrona(4),
        tarea_asincrona(5),
    )

    end = time.time()
    delta = end - start
    print(f"Todas las tareas han finalizado en {delta} s")

# Ejecutar el bucle de eventos
await main()
```

```
Ejecutando tareas
Tarea 1 ejecutándose
Tarea 2 ejecutándose
Tarea 3 ejecutándose
Tarea 4 ejecutándose
Tarea 5 ejecutándose
¡Descarga exitosa!
Tarea 3 terminada
¡Descarga exitosa!
Tarea 5 terminada
¡Descarga exitosa!
Tarea 1 terminada
¡Descarga exitosa!
Tarea 4 terminada
¡Descarga exitosa!
Tarea 2 terminada
Todas las tareas han finalizado en 2.7320001125335693 s
  [213]   Papaya Michoacan      $ 71.52
```

Ejemplo sin bloqueos en asyncio

```
  [217]   Mango Costa           $ 18.54
  [229]   Guayaba del Caribe    $  8.10
----------------------------------------
Total gastado:                  $996.38
Presupuesto sobrante:           $ 22.62
----------------------------------------
Eficiancia:                     % 97.78
========================================

CLIENTE (22)
========================================
Presupuesto original:           $990.00
```

```
import asyncio

contador = 0

async def incrementar(num):
    global contador
    print(f"Llamando a incrementar: {contador} ({num})")
    await asyncio.sleep(5)
    contador += 1
    print(f"I Contador: {contador} ({num})")
    print(f"Finalizando incrementar: {contador} ({num})")

async def decrementar(num):
    global contador
    print(f"Llamando a decrementar: {contador} ({num})")
    await asyncio.sleep(1)
    contador -= 1
    print(f"D Contador: {contador} ({num})")
    print(f"Finalizando decrementar: {contador} ({num})")

async def main():
    await asyncio.gather(
        incrementar(1),
        incrementar(2),
        decrementar(3),
    )

# asyncio.run(main())
await main()
```

```
Llamando a incrementar: 0 (1)
Llamando a incrementar: 0 (2)
Llamando a decrementar: 0 (3)
D Contador: -1 (3)
Finalizando decrementar: -1 (3)
I Contador: 0 (1)
Finalizando incrementar: 0 (1)
I Contador: 1 (2)
Finalizando incrementar: 1 (2)
Presupuesto sobrante:           $ 12.54
```

Ejemplo de bloqueos en asyncio

```
CLIENTE (24)
========================================
Presupuesto original:           $611.00
Total de frutas adquiridas:           7
----------------------------------------
  [611]  Guanabana 4            $132.00
```

```
import asyncio

contador = 0

async def incrementar(num, lock):
    global contador
    print(f"Llamando a incrementar: {contador} ({num})")
    async with lock:
        await asyncio.sleep(5)
        contador += 1
        print(f"I Contador: {contador} ({num})")
    print(f"Finalizando incrementar: {contador} ({num})")

async def decrementar(num, lock):
    global contador
    print(f"Llamando a decrementar: {contador} ({num})")
    async with lock:
        await asyncio.sleep(1)
        contador -= 1
        print(f"D Contador: {contador} ({num})")
    print(f"Finalizando decrementar: {contador} ({num})")

async def main():
    lock = asyncio.Lock()
    await asyncio.gather(
        incrementar(1, lock),
        incrementar(2, lock),
        decrementar(3, lock),
    )

# asyncio.run(main())
await main()
```

```
Llamando a incrementar: 0 (1)
Llamando a incrementar: 0 (2)
Llamando a decrementar: 0 (3)
I Contador: 1 (1)
Finalizando incrementar: 1 (1)
I Contador: 2 (2)
Finalizando incrementar: 2 (2)
D Contador: 1 (3)
Finalizando decrementar: 1 (3)
```

[229]	Guayaba del Caribe	$ 8.10
[209]	Fresa Huston	$ 23.60
[611]	Guanabana 4	$123.00
[209]	Fresa Huston	$ 23.60
[263]	Guayaba Huston	$ 94.54
[613]	Guanabana 6	$123.00
[609]	Guanabana 2	$123.00
[276]	[illegible]	[illegible]

```
with open("/content/sample_data/README.md") as resource:
  # Abre autómaticamente el recurso
  print(resource.read())
  # Cierra autómaticamente el recurso
```

```
This directory includes a few sample datasets to get you started.

*   `california_housing_data*.csv` is California housing data from the 1990 US
    Census; more information is available at:
    https://developers.google.com/machine-learning/crash-course/california-hou

*   `mnist_*.csv` is a small sample of the
    [MNIST database](https://en.wikipedia.org/wiki/MNIST_database), which is
    described at: http://yann.lecun.com/exdb/mnist/

*   `anscombe.json` contains a copy of
    [Anscombe's quartet](https://en.wikipedia.org/wiki/Anscombe%27s_quartet);
    was originally described in

    Anscombe, F. J. (1973). 'Graphs in Statistical Analysis'. American
    Statistician. 27 (1): 17-21. JSTOR 2682899.

    and our copy was prepared by the
    [vega_datasets library](https://github.com/altair-viz/vega_datasets/blob/4
```

```
Total gastado:                  $285.42
```

https://www.inegi.org.mx/contenidos/programas/enoe/15ymas/datosabiertos/2023/conjunto_de_datos_enoe_2023_4t_csv.zip

https://www.inegi.org.mx/contenidos/programas/enoe/15ymas/datosabiertos/2023/conjunto_de_datos_enoe_2023_3t_csv.zip

https://www.inegi.org.mx/contenidos/programas/enoe/15ymas/datosabiertos/2022/conjunto_de_datos_enoen_2022_2t_csv.zip

```
  [221]   Manzana Rubeon          $105.02
  [215]   Papaya de Brasil        $178.19
  [  2]   Pera Huston             $ 44.22
  [220]   Papaya Costa            $169.49
  [293]   Piña de Brasil          $192.04
  [210]   Fresa Michoacán         $158.34
  [  4]   Mango Texas             $ 12.55
  [  2]   Pera Huston             $ 44.22
  [239]   Pera Costa              $  4.59
  [209]   Fresa Huston           $ 23.60
----------------------------------------
Total gastado:                  $992.85
Presupuesto sobrante:           $ 10.15
----------------------------------------
Eficiancia:                     % 98.99
========================================
```

```
import asyncio # Permite ejecutar funciones asíncronas
import aiohttp # Permite descargar archivos en forma asíncrona

async def descargarEnoe(año, trimestre):
  conjunto = "enoen"

  if año >= 2023:
    conjunto = "enoe"

  url = f"https://www.inegi.org.mx/contenidos/programas/enoe/15ymas/datosabiertos

  async with aiohttp.ClientSession() as session:
    async with session.get(url) as response:
      if response.status == 200:
        try:
          data = await response.read()
          print(len(data))
          with open(f"{conjunto}_{año}_{trimestre}t.zip", "wb") as destination:
            destination.write(data)
            print(f"Se descargó el conjunto {conjunto}_{año}_{trimestre}t.zip [{l
        except:
          print(f"No se pudo descargar y guardar el conjunto {conjunto}_{año}_{tr

await descargarEnoe(2023, 1)
```

```
41532376
Se descargó el conjunto enoe_2023_1t.zip [41532376 bytes]
```

```
CLIENTE (30)
=======================================
Presupuesto original:          $810.00
Total de frutas adquiridas:         11
---------------------------------------
  [229]   Guayaba del Caribe    $  8.10
  [  4]   Mango Texas           $ 12.55
  [219]   Guayaba Texas         $199.35
  [245]   Pera Texas            $198.04
  [  2]   Pera Huston           $ 44.22
  [229]   Guayaba del Caribe    $  8.10
  [207]   Manzana Costa         $ 63.76
  [276]   Manzana Michoacán     $ 38.30
  [224]   Piña Michoacán        $ 30.69
  [298]   Manzana Texas         $145.08
  [212]   Pera de Brasil        $ 53.98
---------------------------------------
Total gastado:                 $802.17
Presupuesto sobrante:          $  7.83
---------------------------------------
Eficiancia:                    % 99.03
```

```
async def descargarConjuntosEnoe(periodos):
  tasks = []
  for año, trimestre in periodos:
    tasks.append(descargarEnoe(año, trimestre))
  await asyncio.gather(*tasks)

await descargarConjuntosEnoe([
    (2023, 4),
    (2023, 3),
    (2023, 2),
    (2023, 1),
    (2022, 4),
    (2022, 3),
    (2022, 2),
    (2022, 1),
    (2021, 4),
    (2021, 3),
    (2021, 2),
    (2021, 1),
])
```

```
33876717
Se descargó el conjunto enoen_2021_2t.zip [33876717 bytes]
40388980
Se descargó el conjunto enoen_2022_4t.zip [40388980 bytes]
41532376
Se descargó el conjunto enoe_2023_1t.zip [41532376 bytes]
43112416
Se descargó el conjunto enoen_2021_4t.zip [43112416 bytes]
44005595
Se descargó el conjunto enoe_2023_3t.zip [44005595 bytes]
44630600
Se descargó el conjunto enoen_2022_1t.zip [44630600 bytes]
42728491
Se descargó el conjunto enoe_2023_2t.zip [42728491 bytes]
43788274
Se descargó el conjunto enoe_2023_4t.zip [43788274 bytes]
45815136
Se descargó el conjunto enoen_2021_1t.zip [45815136 bytes]
40289417
Se descargó el conjunto enoen_2022_3t.zip [40289417 bytes]
41272360
Se descargó el conjunto enoen_2022_2t.zip [41272360 bytes]
51070919
Se descargó el conjunto enoen_2021_3t.zip [51070919 bytes]
```

```
[218]   Guayaba Michoacán     $196.73
[220]   Papaya Costa          $169.49
[294]   Mango Huston          $  8.17
[216]   Fresa Costa           $ 32.64
[224]   Piña Michoacán        $ 30.69
```

```
!pip install pymssql
```

```
Collecting pymssql
  Downloading pymssql-2.3.0-cp310-cp310-manylinux_2_28_x86_64.whl (4.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.6/4.6 MB 12.7 MB/s eta 0:00:00
Installing collected packages: pymssql
Successfully installed pymssql-2.3.0
```

```
import asyncio
import pymssql

async def obtenerFrutaAleatoria():
  import pymssql

  conn = pymssql.connect("3.93.192.216", "test_curso", "TestCurso$123", "test_cur

  cursor = conn.cursor()

  cursor.execute("select top(1) id, nombre, precio from frutas order by newid()")

  fruta = cursor.fetchone()

  cursor.close()

  conn.close()

  return { "id": fruta[0], "nombre": fruta[1], "precio": fruta[2] } # Devuelve un

await obtenerFrutaAleatoria()
```

```
{'id': 231, 'nombre': 'Guayaba Costa', 'precio': 5.858660496068071}
```

```
  [212]   Pera de Brasil        $ 53.98
----------------------------------------
Total gastado:                  $1422.32
Presupuesto sobrante:           $  0.68
----------------------------------------
Eficiancia:                     % 99.95
========================================

CLIENTE (34)
========================================
Presupuesto original:           $937.00
Total de frutas adquiridas:          11
----------------------------------------
  [607]   Guanabana             $123.00
  [294]   Mango Huston          $  8.17
  [269]   Mango Michoacán       $165.81
  [  3]   Papaya Tropical       $ 23.99
  [  1]   Manzana Golden        $ 88.56
```

```
async def obtenerFrutasAleatorias(n):
  frutas = []

  for i in range(n):
    fruta = await obtenerFrutaAleatoria()
    frutas.append(fruta)
    print(f"LISTO: {i}")

  return frutas

await obtenerFrutasAleatorias(10)
```

```
    LISTO: 0
    LISTO: 1
    LISTO: 2
    LISTO: 3
    LISTO: 4
    LISTO: 5
    LISTO: 6
    LISTO: 7
    LISTO: 8
    LISTO: 9
    [{'id': 239, 'nombre': 'Pera Costa', 'precio': 4.5908119914192405},
     {'id': 3, 'nombre': 'Papaya Tropical', 'precio': 23.99},
     {'id': 610, 'nombre': 'Guanabana 3', 'precio': 123.0},
     {'id': 228, 'nombre': 'Papaya del Caribe', 'precio': 67.87200048848038},
     {'id': 245, 'nombre': 'Pera Texas', 'precio': 198.0370824616128},
     {'id': 276, 'nombre': 'Manzana Michoacán', 'precio': 38.295746000785314},
     {'id': 228, 'nombre': 'Papaya del Caribe', 'precio': 67.87200048848038},
     {'id': 293, 'nombre': 'Piña de Brasil', 'precio': 192.04067832784654},
     {'id': 216, 'nombre': 'Fresa Costa', 'precio': 32.64463420992833},
     {'id': 250, 'nombre': 'Piña Costa', 'precio': 4.475788045732223}]
      [235]   Piña Texas            $ 40.23
      [  3]   Papaya Tropical       $ 23.99
      [  4]   Mango Texas           $ 12.55
      [263]   Guayaba Huston        $ 94.54
      [294]   Mango Huston          $  8.17
      [231]   Guayaba Costa         $  5.86
      [  3]   Papaya Tropical       $ 23.99
    ----------------------------------------
    Total gastado:                  $1831.42
    Presupuesto sobrante:           $  1.58
    ----------------------------------------
    Eficiancia:                     % 99.91
    ========================================

    CLIENTE (36)
    ========================================
    Presupuesto original:           $122.00
    Total de frutas adquiridas:           4
    ----------------------------------------
```

```
async def obtenerFrutasAleatorias(n):
  frutas = []

  for i in range(n):
    fruta = obtenerFrutaAleatoria()
    frutas.append(fruta)
    print(f"AGREGADO: {i}")

  return await asyncio.gather(*frutas)

await obtenerFrutasAleatorias(10)
```

```
AGREGADO: 0
AGREGADO: 1
AGREGADO: 2
AGREGADO: 3
AGREGADO: 4
AGREGADO: 5
AGREGADO: 6
AGREGADO: 7
AGREGADO: 8
AGREGADO: 9
[{'id': 3, 'nombre': 'Papaya Tropical', 'precio': 23.99},
 {'id': 610, 'nombre': 'Guanabana 3', 'precio': 123.0},
 {'id': 246, 'nombre': 'Piña del Caribe', 'precio': 195.671967939344},
 {'id': 265, 'nombre': 'Guayaba de Brasil', 'precio': 127.60549684240735},
 {'id': 211, 'nombre': 'Fresa de Brasil', 'precio': 51.48794038821201},
 {'id': 210, 'nombre': 'Fresa Michoacán', 'precio': 158.33645138680038},
 {'id': 239, 'nombre': 'Pera Costa', 'precio': 4.5908119914192405},
 {'id': 239, 'nombre': 'Pera Costa', 'precio': 4.5908119914192405},
 {'id': 293, 'nombre': 'Piña de Brasil', 'precio': 192.04067832784654},
 {'id': 216, 'nombre': 'Fresa Costa', 'precio': 32.64463420992833}]
  [217]   Mango Cobra           $ 10.91
----------------------------------------
Total gastado:                  $1412.86
Presupuesto sobrante:           $ 48.14
----------------------------------------
Eficiancia:                     % 96.70
========================================

CLIENTE (38)
========================================
Presupuesto original:           $664.00
Total de frutas adquiridas:           7
----------------------------------------
  [  2]   Pera Huston           $ 44.22
  [215]   Papaya de Brasil      $178.19
  [298]   Manzana Texas         $145.08
  [613]   Guanabana 6           $123.00
  [207]   Manzana Costa         $ 63.76
  [229]   Guayaba del Caribe    $  8.10
  [213]   Papaya Michoacán      $ 71.52
```

```
----------------------------------------
Total gastado:                  $633.87
Presupuesto sobrante:           $ 30.13
----------------------------------------
Eficiancia:                     % 95.46
```