

UNIVERSIDAD IBEROAMERICANA

Desarrollo Unidad de aprendizaje II - Actividad 4 - Manejo del lenguaje de programación Python

Juan Camilo Bocanegra Osorio
jbocane8@ibero.edu.co

25 de mayo de 2025

DESCRIPCIÓN DE LA ACTIVIDAD

Cordial saludo estudiantes, bienvenidos a la actividad 4.

Teniendo en cuenta que uno de los resultados de aprendizaje del curso tiene que ver con apropiar las bases fundamentales del lenguaje de programación Python para el desarrollo de algoritmos de alto rendimiento, esta actividad es para trabajar en equipo y les ayudará a practicar la sintaxis del lenguaje de programación Python de manera guiada.

Para el desarrollo de esta tarea se deben seguir los pasos a continuación:

1. Descargar el editor de código Visual Studio Code siguiendo este link:

<https://code.visualstudio.com/>

Quienes no desean descargar el editor de código pueden trabajar en Google Colab en línea: <https://colab.research.google.com/?hl=es>

2. Revise el manual del editor de código Visual Studio Code:

<https://code.visualstudio.com/docs>

3. Descargar el documento guía del taller para la utilización de líneas de código en lenguaje Python.
4. Realice la interpretación de cada línea de código dentro del taller y deje evidencia de los resultados junto con la interpretación.

Los recursos que pueden consultar para interiorizar la actividad son:

- Hinojosa Gutiérrez, Á. (2015). Python paso a paso. RA-MA Editorial. (pág. 39 - 160)
- Nolasco Valenzuela, J. S. (2018). Python: aplicaciones prácticas. RA-MA Editorial. (pág. 21 - 49)

Recuerde adjuntar la actividad al resultado de aprendizaje correspondiente, en formato PDF.

Entregable

La actividad debe ser entregada en formato PDF teniendo en cuenta los siguientes aspectos:

1. Desarrollar la totalidad del taller.
2. Incluir evidencias como imágenes de la ejecución de los códigos propuestos.
3. Cada respuesta debe tener su descripción e interpretación.

Cuide aspectos importantes como ortografía y coherencia al escribir para que la interpretación sea entendible, portada.

Para el desarrollo de la actividad se nos da el siguiente problema:

Una de las fortalezas deportivas que tiene Colombia y que más glorias le ha dado al país es el patinaje. La competencia internacional implica competir sobre patines que día a día tienen más tecnología y menos peso. En las competencias se definen por reglamento las dimensiones de los patines que utiliza cada deportista, se busca unificar las condiciones para que las carreras sean justas

La Asociación de Patinaje a nivel mundial define un reglamento técnico en el cual se encuentran las especificaciones de los patines, sin embargo, la mayoría de diseños suelen establecer algunos detalles por fuera de las medidas reglamentarias. Las medidas aprobadas por ejemplo en las ruedas son:

- El diámetro de las ruedas debe ubicarse entre 100 y 120 cm.
- Grosor deben medir máximo 80 mm y mínimo 70 mm.
- Ancho debe ser mínimo 90 mm y máximo 110 mm

Por tal razón, le han encomendado a usted la selección de los diseños que cumplen con las condiciones del reglamento en las ruedas, así que debe construir el software que procesa los datos de las bases de datos donde reposan los diseños en el sistema. Su misión es crear un programa en Python que permita mostrarle a los deportistas los precios de la lista de los patines que cumplen con los requerimientos para su consideración.

Entrada

La entrada estará conformada por $N + 1$ líneas:

- La primera línea recibirá un número N que equivale a la cantidad de registros en la base de datos. Cada registro representa el diseño de cada par de ruedas de unos patines.
- Cada una de las siguientes N líneas estará formada por 4 números separados por espacios que representan las diferentes características de las ruedas. Por ejemplo, la fila 110 85 100 12000 representa un par de ruedas con 110 cm de diámetro, 85 mm de grosor, 100 mm de ancho y cuesta 12000 dólares.

Salida

- El programa imprimirá el precio de cada par de ruedas de la base de datos que cumplen con los criterios del reglamento.
- Si no existe ningún registro en la base de datos que cumpla los criterios del reglamento, el programa imprimirá "NO DISPONIBLE"

Entradas de ejemplo	2 280 190 275 1045 235 165 280 820
	3 270 165 275 1195 110 85 100 12000 225 180 225 1120
Salida de ejemplo	NO DISPONIBLE 12000

El código suministrado y comentado es el siguiente:

```

EXPLORER
...
actividad_4.py X
actividad_4.py > ...
1 # Juan Camilo Bocanegra Osorio - Computación de alto desempeño para Big Data (YESID DIAZ 21042025_C2_202541)
2 N = int(input()) # Solicita ingresar un número entero y lo almacena en la variable N.
3 db = [N] # Crea una lista llamada db y almacena el valor de N como su primer elemento.
4 for i in range(N): # Itera N veces, donde N es el número ingresado.
5     c = input() # Solicita ingresar una cadena de texto.
6     db.append(c.split(' ')) # Divide la cadena de texto en una lista de palabras y la agrega a db.
7 P1 = 240 # Inicializa P1 con el valor 240.
8 P2 = 300 # Inicializa P2 con el valor 300.
9 P3 = 160 # Inicializa P3 con el valor 160.
10 P4 = 180 # Inicializa P4 con el valor 180.
11 P5 = 240 # Inicializa P5 con el valor 240.
12 P6 = 275 # Inicializa P6 con el valor 275.
13 P7 = 50 # Inicializa P7 con el valor 50.
14 parametros = [P1, P2, P3, P4, P5, P6, P7] # Crea una lista llamada parametros que contiene los valores de P1 a P7.
15 disponibilidad = False # Inicializa la variable disponibilidad como False.
16 exito = [] # Crea una lista vacía llamada exito.
17 for i in range(db[0]): # Itera db[0] veces, donde db[0] es el valor de N.
18     # int(db[i+1][0]) >= parametros[0] verifica si el primer valor de la lista dentro de la iteración es mayor o igual a P1.
19     # int(db[i+1][1]) >= parametros[2] verifica si el segundo valor de la lista dentro de la iteración es mayor o igual a P3.
20     # int(db[i+1][1]) <= parametros[3] verifica si el segundo valor de la lista dentro de la iteración es menor o igual a P4.
21     # int(db[i+1][2]) >= parametros[4] verifica si el tercer valor de la lista dentro de la iteración es mayor o igual a P5.
22     # int(db[i+1][2]) <= parametros[5] verifica si el tercer valor de la lista dentro de la iteración es menor o igual a P6.
23     if (int(db[i+1][0]) >= parametros[0]) and \
24         (int(db[i+1][1]) >= parametros[2]) and \
25         (int(db[i+1][1]) <= parametros[3]) and \
26         (int(db[i+1][2]) >= parametros[4]) and \
27         (int(db[i+1][2]) <= parametros[5]):
28         exito.append(int(db[i+1][3])) # Agrega el cuarto valor de db[i+1] a la lista exito.
29         disponibilidad = True # Establece disponibilidad como True.
30 if not disponibilidad: # Valida si disponibilidad es False.
31     print("NO DISPONIBLE") # Imprime "NO DISPONIBLE".
32 else: # Si por el contrario disponibilidad es True.
33     for i in exito: # Itera sobre cada elemento en exito.
34         print(i) # Imprime el elemento.
35         print() # Imprime una línea en blanco.
36

```

```
# Juan Camilo Bocanegra Osorio - Computación de alto desempeño para Big Data
(YESID DIAZ 21042025_C2_202541)
N = int(input()) # Solicita ingresar un número entero y lo almacena en la
variable N.
db = [N] # Crea una lista llamada db y almacena el valor de N como su primer
elemento.
for i in range(N): # Itera N veces, donde N es el número ingresado.
    c = input() # Solicita ingresar una cadena de texto.
    db.append(c.split(' ')) # Divide la cadena de texto en una lista de palabras
y la agrega a db.
P1 = 240 # Inicializa P1 con el valor 240.
P2 = 300 # Inicializa P2 con el valor 300.
P3 = 160 # Inicializa P3 con el valor 160.
P4 = 180 # Inicializa P4 con el valor 180.
P5 = 240 # Inicializa P5 con el valor 240.
P6 = 275 # Inicializa P6 con el valor 275.
P7 = 50 # Inicializa P7 con el valor 50.
parametros = [P1, P2, P3, P4, P5, P6, P7] # Crea una lista llamada parámetros que
contiene los valores de P1 a P7.
disponibilidad = False # Inicializa la variable disponibilidad como False.
exito = [] # Crea una lista vacía llamada exito.
for i in range(db[0]): # Itera db[0] veces, donde db[0] es el valor de N.
    # int(db[i+1][0]) >= parametros[0] verifica si el primer valor de la lista
dentro de la iteración es mayor o igual a P1.
    # int(db[i+1][1]) >= parametros[2] verifica si el segundo valor de la lista
dentro de la iteración es mayor o igual a P3.
    # int(db[i+1][1]) <= parametros[3] verifica si el segundo valor de la lista
dentro de la iteración es menor o igual a P4.
    # int(db[i+1][2]) >= parametros[4] verifica si el tercer valor de la lista
dentro de la iteración es mayor o igual a P5.
    # int(db[i+1][2]) <= parametros[5] verifica si el tercer valor de la lista
dentro de la iteración es menor o igual a P6.
    if (int(db[i+1][0]) >= parametros[0]) and \
(int(db[i+1][1]) >= parametros[2]) and \
(int(db[i+1][1]) <= parametros[3]) and \
(int(db[i+1][2]) >= parametros[4]) and \
(int(db[i+1][2]) <= parametros[5]):
        exito.append(int(db[i+1][3])) # Agrega el cuarto valor de db[i+1] a la
lista exito.
        disponibilidad = True # Establece disponibilidad como True.
if not disponibilidad: # Valida si disponibilidad es False.
    print("NO DISPONIBLE") # Imprime "NO DISPONIBLE".
else: # Si por el contrario disponibilidad es True.
    for i in exito: # Itera sobre cada elemento en exito.
```

```
print(i) # Imprime el elemento.  
print() # Imprime una línea en blanco.
```

Enlace a [Github](#).

Luego de verificar el código, propongo una versión optimizada, donde creamos métodos según la recepción de los datos y las validaciones con los parámetros de disponibilidad:

```
# Juan Camilo Bocanegra Osorio - Computación de alto desempeño para Big Data  
(YESID DIAZ 21042025_C2_202541)  
def get_input_data(): # Función para ingresar los datos  
    N = int(input()) # Solicita ingresar un número entero y lo almacena en la  
    variable N.  
    db = [N] # Crea una lista llamada db y almacena el valor de N como su primer  
    elemento.  
    for _ in range(N): # Itera N veces, donde N es el número ingresado,  
    ignorando el valor iterado (_).  
        c = input() # Solicita ingresar una cadena de texto.  
        db.append(list(map(int, c.split()))) # Divide la cadena de texto en una  
        lista de enteros y la agrega a db, mapeando los valores como enteros y luego  
        recreando la lista.  
    return db  
  
def check_availability(db, parametros): # Función que analiza los datos y valida  
    disponibilidad.  
    disponibilidad = False # Inicializa la variable disponibilidad como False.  
    exito = [] # Crea una lista vacía llamada exito.  
    for i in range(db[0]): # Itera db[0] veces, donde db[0] es el valor de N.  
        if all([ # Devuelve True si todas las validaciones son correctas, de lo  
        contrario, False.  
            db[i+1][0] >= parametros[0], #verifica si el primer valor de la lista  
            dentro de la iteración es mayor o igual al primer valor de parametros.  
            parametros[2] <= db[i+1][1] <= parametros[3], # Verifica si el  
            segundo valor de la lista dentro de la iteración está entre los valores 3 y 4 de  
            parámetros.  
            parametros[4] <= db[i+1][2] <= parametros[5] # Verifica si el tercer  
            valor de la lista dentro de la iteración está entre los valores 5 y 6 de  
            parámetros.  
        ]):  
            exito.append(db[i+1][3]) # Agrega el cuarto valor de db[i+1] a la  
            lista exito.  
            disponibilidad = True # Establece disponibilidad como True.  
    return disponibilidad, exito # Devuelve los 2 valores para que la función  
    principal actúe.
```

```
def main():
    parametros = [240, 300, 160, 180, 240, 275, 50] # Define los parámetros
    db = get_input_data() # Obtiene los datos de entrada
    disponibilidad, exito = check_availability(db, parametros) # Verifica la
    disponibilidad y obtiene los éxitos

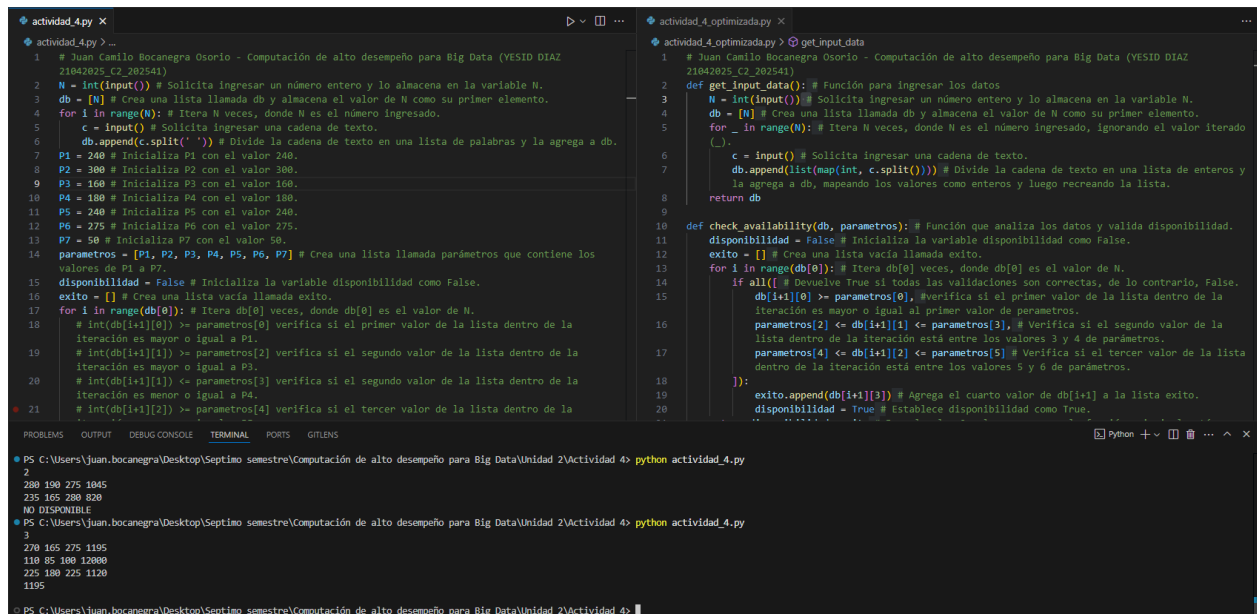
    if not disponibilidad: # Valida si disponibilidad es False.
        print("NO DISPONIBLE") # Imprime "NO DISPONIBLE".
    else: # Si por el contrario disponibilidad es True.
        for i in exito: # Itera sobre cada elemento en exito.
            print(i) # Imprime el elemento.
        print()

if __name__ == "__main__": # Valida si se está ejecutando directamente o es
    importado.
    main() # Ejecuta la función principal.
```

Enlace a [Github](#).

PRUEBAS SOBRE EL CÓDIGO

Se realizan las pruebas a través de los comandos:



The screenshot shows a code editor with two files open: `actividad_4.py` and `actividad_4_optimizada.py`. The left pane shows `actividad_4.py` with comments in Spanish and code for input, validation, and output. The right pane shows `actividad_4_optimizada.py` with similar code but with optimizations. The bottom pane shows the terminal output of running `python actividad_4.py`, which displays the parameters, availability status, and the resulting success list.

```
PS C:\Users\Juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4.py
2
280 190 275 1045
235 165 280 820
NO DISPONIBLE
PS C:\Users\Juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4.py
3
270 165 275 1195
110 85 100 12000
225 180 225 1120
1195
```

```

# Juan Camilo Bocanegra Osorio - Computación de alto desempeño para Big Data (YESID DIAZ
21042025_C2_202541)
N = int(input()) # Solicita ingresar un número entero y lo almacena en la variable N.
db = [N] # Crea una lista llamada db y almacena el valor de N como su primer elemento.
for i in range(N): # Itera N veces, donde N es el número ingresado.
    c = input() # Solicita ingresar una cadena de texto.
    db.append(c.split(' ')) # Divide la cadena de texto en una lista de palabras y la agrega a db.
P1 = 240 # Inicializa P1 con el valor 240.
P2 = 300 # Inicializa P2 con el valor 300.
P3 = 160 # Inicializa P3 con el valor 160.
P4 = 180 # Inicializa P4 con el valor 180.
P5 = 240 # Inicializa P5 con el valor 240.
P6 = 275 # Inicializa P6 con el valor 275.
P7 = 50 # Inicializa P7 con el valor 50.
parametros = [P1, P2, P3, P4, P5, P6, P7] # Crea una lista llamada parametros que contiene los
valores de P1 a P7.
disponibilidad = False # Inicializa la variable disponibilidad como False.
exitos = [] # Crea una lista vacía llamada exitos.
for i in range(len(db)): # Itera db[0] veces, donde db[0] es el valor de N.
    # int(db[i+1][0]) >= parametros[0] verifica si el primer valor de la lista dentro de la
    # iteración es mayor o igual a P1.
    # int(db[i+1][1]) >= parametros[2] verifica si el segundo valor de la lista dentro de la
    # iteración es mayor o igual a P3.
    # int(db[i+1][1]) <= parametros[3] verifica si el segundo valor de la lista dentro de la
    # iteración es menor o igual a P4.
    # int(db[i+1][2]) >= parametros[4] verifica si el tercer valor de la lista dentro de la
    # iteración es mayor o igual a P5.
    # int(db[i+1][2]) <= parametros[5] verifica si el tercer valor de la lista dentro de la
    # iteración es menor o igual a P6.
    # int(db[i+1][3]) <= parametros[6] verifica si el cuarto valor de la lista dentro de la
    # iteración es menor o igual a P7.
    # Si todas las condiciones se cumplen, se agrega el valor de db[i+1] a la lista exitos.
    # Si alguna condición no se cumple, se establece disponibilidad como True.
    if (int(db[i+1][0]) >= parametros[0]) and (int(db[i+1][1]) >= parametros[2]) and (int(db[i+1][1]) <= parametros[3]) and (int(db[i+1][2]) >= parametros[4]) and (int(db[i+1][2]) <= parametros[5]) and (int(db[i+1][3]) <= parametros[6]):
        exitos.append(db[i+1])
    else:
        disponibilidad = True
if disponibilidad == False:
    print('Disponible')
else:
    print('No disponible')

```

```

# Juan Camilo Bocanegra Osorio - Computación de alto desempeño para Big Data (YESID DIAZ
21042025_C2_202541)
def get_input_data(): # Función para ingresar los datos
    N = int(input()) # Solicita ingresar un número entero y lo almacena en la variable N.
    db = [N] # Crea una lista llamada db y almacena el valor de N como su primer elemento.
    for _ in range(N): # Itera N veces, donde N es el número ingresado, ignorando el valor iterado
        c = input() # Solicita ingresar una cadena de texto.
        db.append(list(map(int, c.split(' ')))) # Divide la cadena de texto en una lista de enteros y
        la agrega a db, mapeando los valores como enteros y luego recreando la lista.
    return db

def check_availability(db, parametros): # Función que analiza los datos y valida disponibilidad.
    disponibilidad = False # Inicializa la variable disponibilidad como False.
    exitos = [] # Crea una lista vacía llamada exitos.
    for i in range(len(db)): # Itera db[0] veces, donde db[0] es el valor de N.
        if all([
            db[i+1][0] >= parametros[0], # Verifica si el primer valor de la lista dentro de la
            iteración es mayor o igual al primer valor de parametros.
            db[i+1][1] >= parametros[2], # Verifica si el segundo valor de la lista dentro de la
            iteración está entre los valores 3 y 4 de parametros.
            db[i+1][1] <= parametros[3], # Verifica si el segundo valor de la lista dentro de la
            iteración está entre los valores 5 y 6 de parametros.
            db[i+1][2] >= parametros[4], # Verifica si el tercer valor de la lista dentro de la
            iteración está entre los valores 5 y 6 de parametros.
            db[i+1][2] <= parametros[5], # Verifica si el tercer valor de la lista dentro de la
            iteración está entre los valores 5 y 6 de parametros.
            db[i+1][3] <= parametros[6] # Verifica si el cuarto valor de la lista dentro de la
            iteración está entre los valores 5 y 6 de parametros.
        ]):
            exitos.append(db[i+1])
        else:
            disponibilidad = True
    if disponibilidad == False:
        print('Disponible')
    else:
        print('No disponible')

```

```

235 165 280 820
NO DISPONIBLE
PS C:\Users\Juan.Bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4.py
3
270 165 275 1195
110 85 100 12000
225 180 225 1120
1195

PS C:\Users\Juan.Bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4_optimizada.py
2
280 190 275 1845
235 165 280 820
NO DISPONIBLE

PS C:\Users\Juan.Bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4_optimizada.py
3
270 165 275 1195
110 85 100 12000
225 180 225 1120
1195

PS C:\Users\Juan.Bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4>

```

En ambos casos las salidas son similares, con una situación particular: la respuesta para el segundo caso es diferente al caso de prueba del taller:

En el taller la salida del segundo caso es 12000 y en el código obtenido y en el optimizado es 1195, esto porque la única lista que cumple las condiciones no es la indicada por el taller:

Entrada 110 85 100 12000:

Condición	Valor lista	Valor parametros	Resultado taller	Resultado código
<code>(int(db[i+1][0]) >= parametros[0])</code>	110	240	True	False
<code>(int(db[i+1][1]) >= parametros[2])</code>	85	160	True	False
<code>(int(db[i+1][1]) <= parametros[3])</code>	85	180	True	True
<code>(int(db[i+1][2]) >= parametros[4])</code>	100	240	True	False
<code>(int(db[i+1][2]) <= parametros[5])</code>	100	275	True	True

Entrada 270 165 275 1195:

Condición	Valor lista	Valor parametros	Resultado taller	Resultado código
<code>(int(db[i+1][0]) >= parametros[0])</code>	270	240	?	True
<code>(int(db[i+1][1]) >= parametros[2])</code>	165	160	?	True
<code>(int(db[i+1][1]) <= parametros[3])</code>	165	180	?	True
<code>(int(db[i+1][2]) >= parametros[4])</code>	275	240	?	True
<code>(int(db[i+1][2]) <= parametros[5])</code>	275	275	?	True

La siguiente imagen, obtenida de Python tutor muestra de manera más didáctica como funciona el código para este caso:

Python Tutor: Visualize Code and Get AI Help for [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

```

Python 3.11
known limitations
6 P1 = 240
7 P2 = 300
8 P3 = 160
9 P4 = 180
10 P5 = 240
11 P6 = 275
12 P7 = 50
13 parametros = [P1,P2,P3,P4,P5,P6,P7]
14 disponibilidad=False
15 exito=[]
16 for i in range(db[0]):
17     if (int(db[i+1][0]) >= parametros[0]) and \
18         (int(db[i+1][1]) >= parametros[2]) and \
19         (int(db[i+1][1]) <= parametros[3]) and \
20         (int(db[i+1][2]) >= parametros[4]) and \
21         (int(db[i+1][2]) <= parametros[5]):
22         exito.append(int(db[i+1][3]))
23         disponibilidad = True
24 if not disponibilidad:
25     print("NO DISPONIBLE")
26 else:

```

Print output (drag lower right corner to resize)

```

3
270 165 275 1195
110 85 100 12000
225 180 225 1120
1195

```

Frames

Global frame

N	3
db	
i	1195
c	"225 180 225 1120"
P1	240
P2	300
P3	160
P4	180
P5	240
P6	275
P7	50
parametros	
disponibilidad	True
exito	

Objects

list

0	1	2	3
3			

list

0	1	2	3
"225"	"180"	"225"	"1120"

list

0	1	2	3
"270"	"165"	"275"	"1195"

list

0	1	2	3
"110"	"85"	"100"	"12000"

list

0	1	2	3	4	5	6
240	300	160	180	240	275	50

list

0	
1195	

Done running (40 steps)

Otro caso de prueba es el siguiente:

```
PS C:\Users\juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4.py
8
250 160 241 2500
230 175 290 14000
180 160 250 10000
240 161 275 8500
220 170 210 4500
260 130 275 9600
250 167 274 11950
190 160 280 9900
2500
8500
11950

PS C:\Users\juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4>

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

PS C:\Users\juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4> python actividad_4_optimizada.py
8
250 160 241 2500
230 175 290 14000
180 160 250 10000
240 161 275 8500
220 170 210 4500
260 130 275 9600
250 167 274 11950
190 160 280 9900
2500
8500
11950

PS C:\Users\juan.bocanegra\Desktop\Septimo semestre\Computación de alto desempeño para Big Data\Unidad 2\Actividad 4>
```

Ambos códigos cumplen con las condiciones descritas en el taller

CONCLUSIÓN SOBRE EL USO DE LA SINTAXIS EN PYTHON

La sintaxis es sencilla y legible, lo que hace más fácil la escritura y comprensión del código. Algunas conclusiones que pude notar sobre el uso de la sintaxis en el código son:

- La entrada de los datos es más sencilla, ya que solo se requiere una línea de código para leer y convertir datos desde consola, no requiere importar librerías o declarar tipos en las variables.

La sentencia `N = int(input())` solicita al usuario que ingrese un valor, lo cual es útil para definir la cantidad de datos que se procesarán.

La sentencia `db = [N]` inicializa una lista con el número de entradas como primer elemento.

- La iteración se parece bastante a otros lenguajes, por ello es fácil de integrar.

`for i in range(N)` utiliza un bucle for para iterar N veces, permitiendo la entrada de múltiples líneas de datos.

- La inicialización y gestión de los datos es más intuitiva y sencilla de gestionar.

`db.append(c.split(' '))` divide la cadena de texto en una lista de palabras y la agrega a db.

Los parámetros (P1, P2, etc.) se definen claramente facilitando su uso en condiciones posteriores.

- Las declaraciones de condición son más completas.

`if (int(db[i+1][0]) >= parametros[0]) and ...:` Utiliza condiciones para verificar si los datos cumplen ciertos criterios, y la función `all([...])` puede optimizar estas verificaciones.
`if not disponibilidad` se encarga de verificar si no se encontraron datos que cumplan las condiciones y muestra un mensaje adecuado.

- Por último, puedo notar que la estructura del código es modular y clara, lo que facilita su mantenimiento y comprensión.

BIBLIOGRAFÍA

- Hinojosa Gutiérrez, Á. (2015). Python paso a paso. RA-MA Editorial. (pág 39 - 160)
- Nolasco Valenzuela, J. S. (2018). Python: aplicaciones prácticas. RA-MA Editorial. (pág 21 - 49)