

EJ1 - Área de polígonos

Algoritmos y Programación I - Curso Essay (7540)

Práctica: Bárbara

Alumno: Barberis, Juan Celestino

Padrón: 105147

Ayudante a cargo: Javier Di Santo.

Segundo Cuatrimestre 2019

Parte 1: Entorno de Trabajo

Entorno de trabajo en Linux (Ubuntu 19.04), utilizando Visual Studio Code como editor de texto y la terminal incluida en el mismo editor de texto.

Parte 1.1

Se pide ingresar en el intérprete de Python y ejecutar la siguiente línea de código: **'Hola Algoritmos y Programación I'**. Una vez ejecutada, se procede a salir del intérprete con el comando **exit()**.

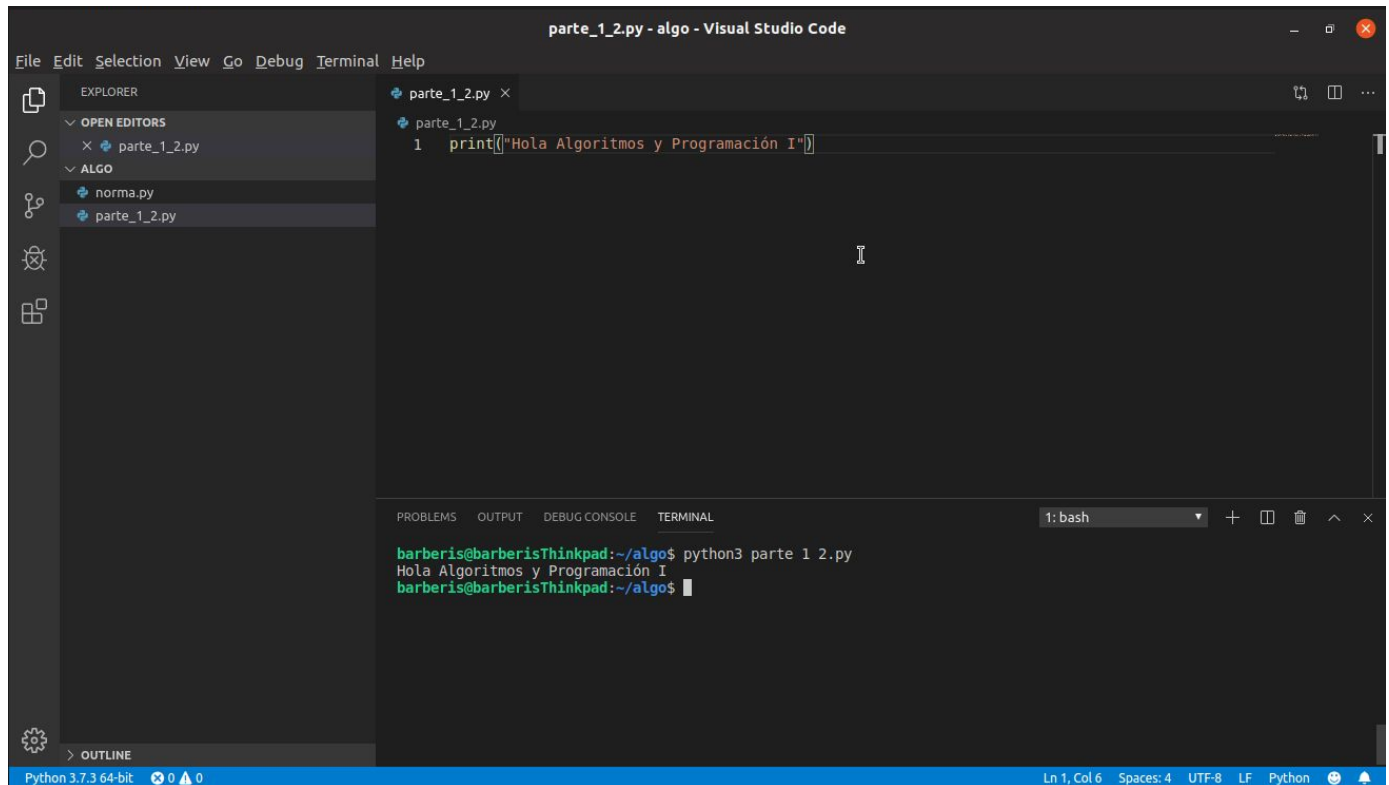
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash
barberis@barberisThinkpad:~$ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 'Hola Algoritmos y Programacion I'
'Hola Algoritmos y Programacion I'
>>> exit()
barberis@barberisThinkpad:~$
```

Parte 1.2

Se crea un archivo llamado **'parte_1_2.py'**, se le agrega la siguiente cadena de texto **"Hola Algoritmos y Programación I"** y se ejecuta por medio de la terminal con el comando **'python3 parte_1_2.py'**.

The screenshot shows the Visual Studio Code editor with a dark theme. The title bar at the top reads "parte_1_2.py - algo - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. On the left sidebar, the Explorer view is active, showing a file named "parte_1_2.py" under the "ALGO" folder. The Open Editors view also shows "parte_1_2.py". The main editor area displays the content of "parte_1_2.py", which is a single line of code: `1 "Hola Algoritmos y Programación I"`. At the bottom, the Terminal view is open, showing a bash prompt and the command `python3 parte_1_2.py` being executed. The status bar at the bottom indicates the Python version is 3.7.3 64-bit, with 0 errors and 0 warnings.

Si quisiera obtener el resultado de la **Parte 1.1**, debería agregar la función ***print()*** a la cadena de texto. Como se muestra en la siguiente imagen.



The image shows a screenshot of the Visual Studio Code editor interface. The title bar at the top reads "parte_1_2.py - algo - Visual Studio Code". The Explorer sidebar on the left shows a file tree with "parte_1_2.py" selected. The main editor window displays the following code in "parte_1_2.py":

```
1 print("Hola Algoritmos y Programación I")
```

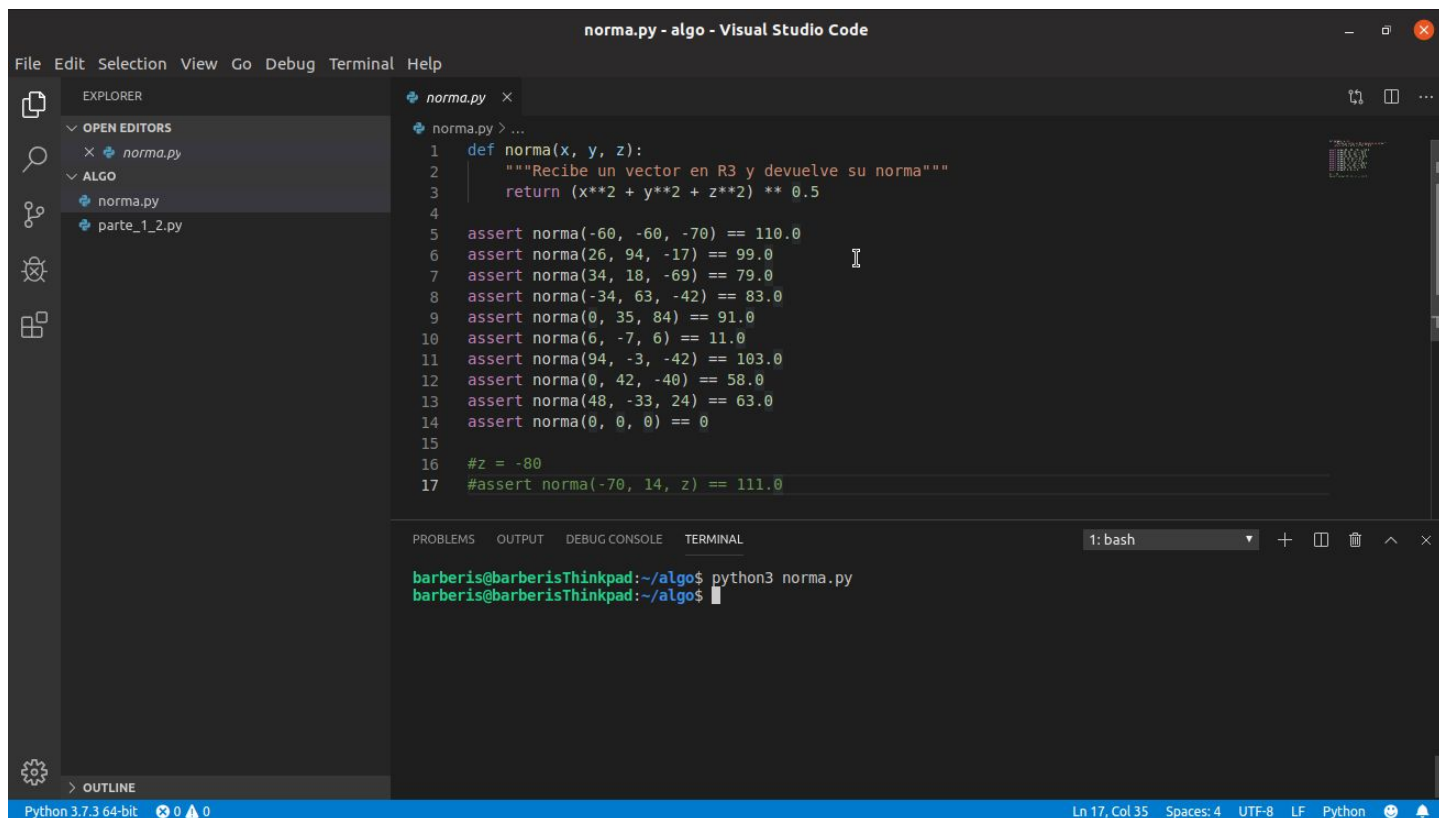
Below the editor, the TERMINAL panel is open, showing the command prompt "1: bash". The terminal output shows the execution of the script:

```
barberis@barberisThinkpad:~/algo$ python3 parte 1 2.py
Hola Algoritmos y Programación I
barberis@barberisThinkpad:~/algo$
```

The status bar at the bottom indicates "Python 3.7.3 64-bit", "0 0" errors/warnings, and "Ln 1, Col 6 Spaces: 4 UTF-8 LF Python".

Parte 2: norma.py

Se pide que ejecute el programa 'norma.py'



```
norma.py - algo - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

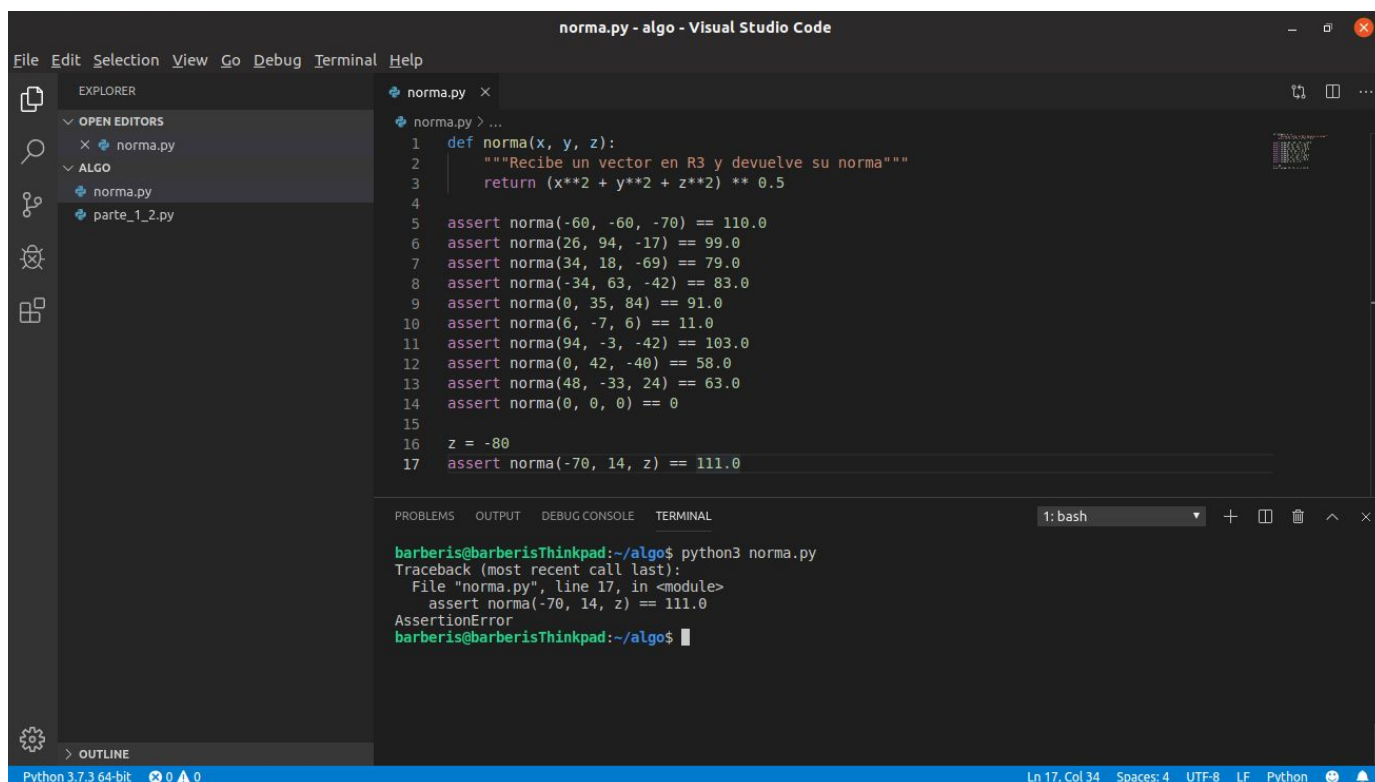
EXPLORER
  OPEN EDITORS
    norma.py
  ALGO
    norma.py
    parte_1_2.py

norma.py
1 def norma(x, y, z):
2     """Recibe un vector en R3 y devuelve su norma"""
3     return (x**2 + y**2 + z**2) ** 0.5
4
5 assert norma(-60, -60, -70) == 110.0
6 assert norma(26, 94, -17) == 99.0
7 assert norma(34, 18, -69) == 79.0
8 assert norma(-34, 63, -42) == 83.0
9 assert norma(0, 35, 84) == 91.0
10 assert norma(6, -7, 6) == 11.0
11 assert norma(94, -3, -42) == 103.0
12 assert norma(0, 42, -40) == 58.0
13 assert norma(48, -33, 24) == 63.0
14 assert norma(0, 0, 0) == 0
15
16 #z = -80
17 #assert norma(-70, 14, z) == 111.0

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash

barberis@barberisThinkpad:~/algo$ python3 norma.py
barberis@barberisThinkpad:~/algo$
```

Ahora ejecuto de nuevo el programa, pero des-comento las últimas dos líneas (la 16 y la 17).



```
norma.py - algo - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    norma.py
  ALGO
    norma.py
    parte_1_2.py

norma.py
1 def norma(x, y, z):
2     """Recibe un vector en R3 y devuelve su norma"""
3     return (x**2 + y**2 + z**2) ** 0.5
4
5 assert norma(-60, -60, -70) == 110.0
6 assert norma(26, 94, -17) == 99.0
7 assert norma(34, 18, -69) == 79.0
8 assert norma(-34, 63, -42) == 83.0
9 assert norma(0, 35, 84) == 91.0
10 assert norma(6, -7, 6) == 11.0
11 assert norma(94, -3, -42) == 103.0
12 assert norma(0, 42, -40) == 58.0
13 assert norma(48, -33, 24) == 63.0
14 assert norma(0, 0, 0) == 0
15
16 z = -80
17 assert norma(-70, 14, z) == 111.0

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash

barberis@barberisThinkpad:~/algo$ python3 norma.py
Traceback (most recent call last):
  File "norma.py", line 17, in <module>
    assert norma(-70, 14, z) == 111.0
AssertionError
barberis@barberisThinkpad:~/algo$
```

Preguntas:

¿Cuál es la salida del programa?

El programa calcula la norma de un vector en R3. En la primera ejecución, el programa se ejecuta sin errores y finaliza correctamente. En la segunda ejecución, cuando des-comentamos la línea 16 y 17, la instrucción '**assert**' devuelve un **false**, lo que Python devuelve como un **AssertionError**.

¿Qué línea generó el error? ¿Qué significa? ¿Cómo podemos solucionarlo?

La línea que generó el error es la **17**. La instrucción '**assert**' devolvió un valor **false** lo que significa que `assert norma(-70, 14, z) == 111.0` no es correcto.

Para solucionar este error, debería cumplirse la igualdad planteada, cambiando el valor de la variable **z**.

¿Por qué en el punto 2.1 no se imprimió nada y ahora sí? ¿Qué hace la instrucción assert?

En el punto **2.1** no se imprimió nada porque todas las igualdades se cumplieron y el programa pudo finalizar sin ningún problema. Cuando des-comentamos las últimas dos líneas, y al NO cumplirse la igualdad, **assert** resultó arrojando un error del tipo **AssertionError** en la línea 17.

La instrucción **Assert** se encarga de verificar si una condición se cumple o no. Si se cumple, devuelve un valor **true** y Python continúa ejecutando normalmente. En caso de no cumplirse la condición, devuelve un valor **false** y el programa detiene su ejecución, arrojando un error.

Solucionar el problema. Hallar el valor de z para que ya no de error.

Para solucionar el problema y evitar que el programa siga arrojando un **AssertionError** solo basta con hallar un valor de **z** que satisfaga la igualdad.

En este caso, con **z = 85** o **-85** la igualdad se cumple y el programa se ejecuta sin errores.

norma.py - algo - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER

OPEN EDITORS

norma.py

ALGO

norma.py

parte_1_2.py

```
1 def norma(x, y, z):
2     """Recibe un vector en R3 y devuelve su norma"""
3     return (x**2 + y**2 + z**2) ** 0.5
4
5 assert norma(-60, -60, -70) == 110.0
6 assert norma(26, 94, -17) == 99.0
7 assert norma(34, 18, -69) == 79.0
8 assert norma(-34, 63, -42) == 83.0
9 assert norma(0, 35, 84) == 91.0
10 assert norma(6, -7, 6) == 11.0
11 assert norma(94, -3, -42) == 103.0
12 assert norma(0, 42, -40) == 58.0
13 assert norma(48, -33, 24) == 63.0
14 assert norma(0, 0, 0) == 0
15
16 #Con z = 85, la igualdad de cumple. (Vale también para -85).
17 z = 85
18 assert norma(-70, 14, z) == 111.0
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: bash

```
barberis@barberisThinkpad:~/algo$ python3 norma.py
barberis@barberisThinkpad:~/algo$
```

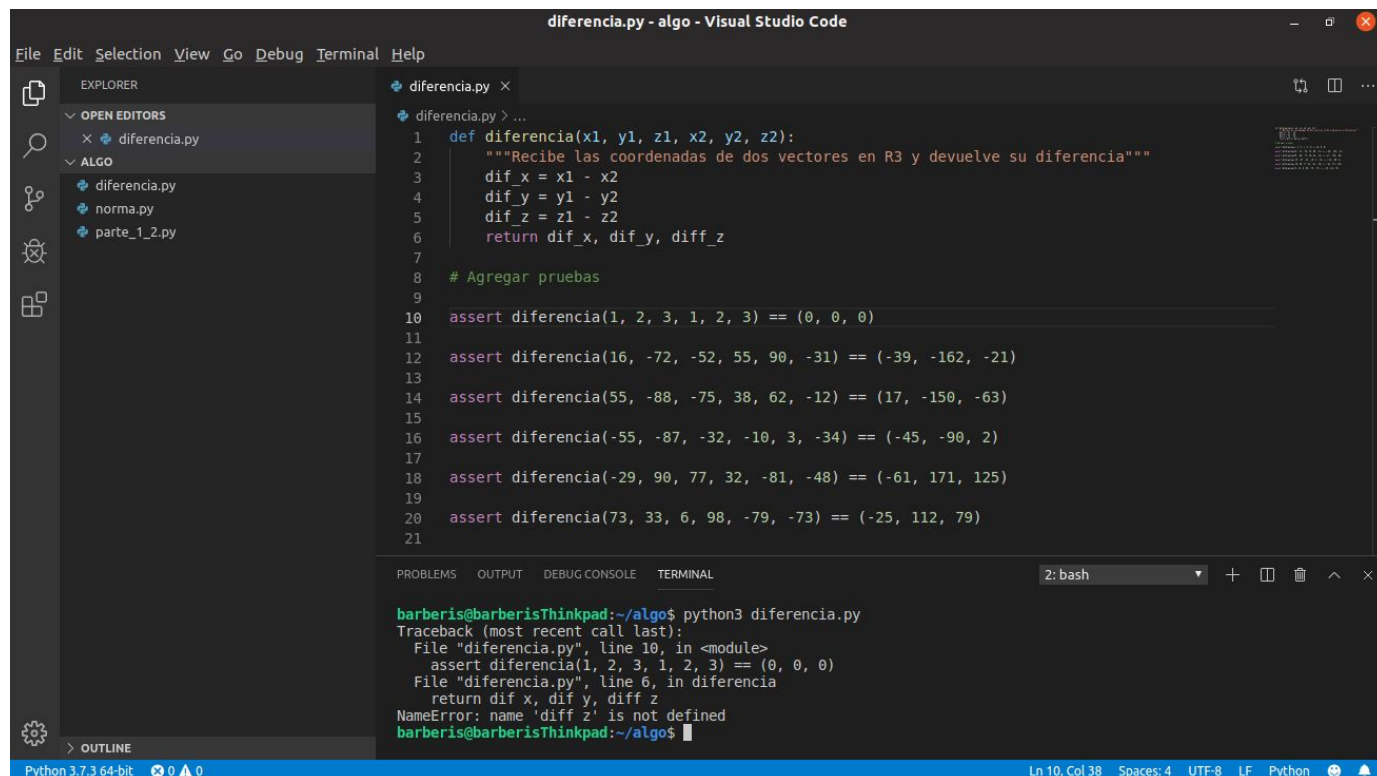
Python 3.7.3 64-bit

Ln 16, Col 61 Spaces: 4 UTF-8 LF Python

ejecución sin errores una vez modificamos el valor de z.

Parte 3: diferencia.py

Se ejecuta el archivo 'diferencia.py' agregando los diversos vectores A y B, para verificar la diferencia entre los dos utilizando la instrucción **assert** y la función definida como 'diferencia'.



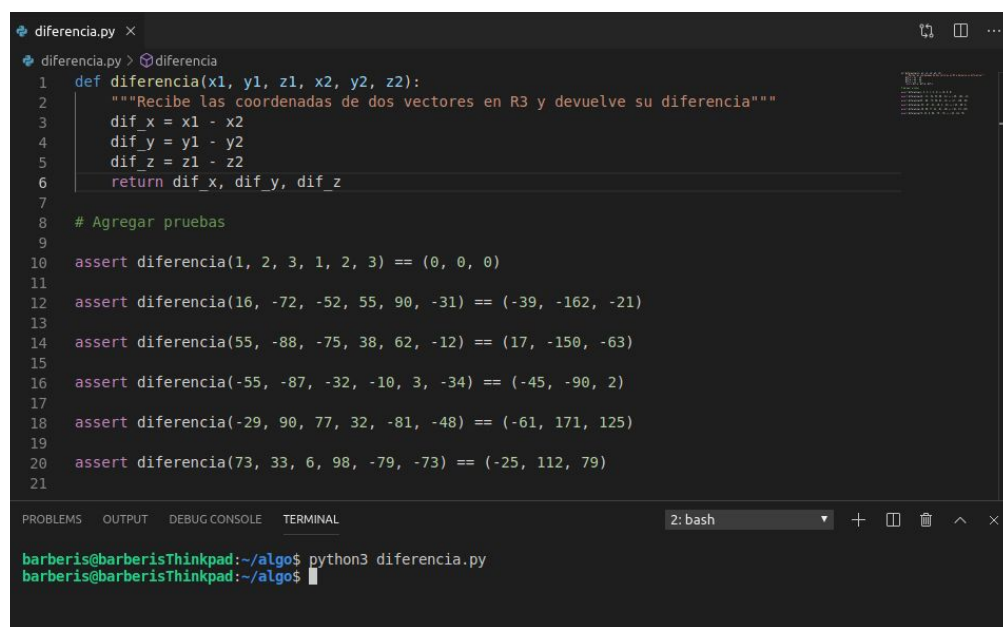
```
def diferencia(x1, y1, z1, x2, y2, z2):
    """Recibe las coordenadas de dos vectores en R3 y devuelve su diferencia"""
    dif_x = x1 - x2
    dif_y = y1 - y2
    dif_z = z1 - z2
    return dif_x, dif_y, dif_z

# Agregar pruebas
assert diferencia(1, 2, 3, 1, 2, 3) == (0, 0, 0)
assert diferencia(16, -72, -52, 55, 90, -31) == (-39, -162, -21)
assert diferencia(55, -88, -75, 38, 62, -12) == (17, -150, -63)
assert diferencia(-55, -87, -32, -10, 3, -34) == (-45, -90, 2)
assert diferencia(-29, 90, 77, 32, -81, -48) == (-61, 171, 125)
assert diferencia(73, 33, 6, 98, -79, -73) == (-25, 112, 79)
```

```
barberis@barberisThinkpad:~/algo$ python3 diferencia.py
Traceback (most recent call last):
  File "diferencia.py", line 10, in <module>
    assert diferencia(1, 2, 3, 1, 2, 3) == (0, 0, 0)
  File "diferencia.py", line 6, in diferencia
    return dif_x, dif_y, dif_z
NameError: name 'dif_z' is not defined
barberis@barberisThinkpad:~/algo$
```

¿Se detectó algún error? ¿Cuál era? ¿Qué significa? ¿Qué línea estaba fallando? ¿Cómo lo solucionaste?

Sí, se detectó un error. Era del tipo **NameError**, y aparece en la línea **6**. Error a la hora de devolver un valor por el método **return**. Se intenta devolver una variable llamada 'dif_z' la cual no está definida porque el nombre real de la variable que sí existe es 'dif_z'.



```
def diferencia(x1, y1, z1, x2, y2, z2):
    """Recibe las coordenadas de dos vectores en R3 y devuelve su diferencia"""
    dif_x = x1 - x2
    dif_y = y1 - y2
    dif_z = z1 - z2
    return dif_x, dif_y, dif_z

# Agregar pruebas
assert diferencia(1, 2, 3, 1, 2, 3) == (0, 0, 0)
assert diferencia(16, -72, -52, 55, 90, -31) == (-39, -162, -21)
assert diferencia(55, -88, -75, 38, 62, -12) == (17, -150, -63)
assert diferencia(-55, -87, -32, -10, 3, -34) == (-45, -90, 2)
assert diferencia(-29, 90, 77, 32, -81, -48) == (-61, 171, 125)
assert diferencia(73, 33, 6, 98, -79, -73) == (-25, 112, 79)
```

```
barberis@barberisThinkpad:~/algo$ python3 diferencia.py
barberis@barberisThinkpad:~/algo$
```

Una vez que se soluciona, el programa se logra ejecutar sin arrojar errores.

Parte 4: Depuración

El programa dentro del archivo ‘**prodvect.py**’ arroja un **AssertionError** en la línea **11**.

```
prodvect.py X
prodvect.py > ...
1 def mi_funcion(x1, y1, z1, x2, y2, z2):
2     """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""
3     var1 = y1*z2 - z1*y2
4     var2 = z1*x2 - x1*z2
5     var3 = x1*y2 - y1*x2
6     return var1, var2, var3
7
8
9 assert mi_funcion(54, 12, 29, 1, 11, 12) == (-175, -619, 582)
10 assert mi_funcion(71, 52, 24, 1, 11, 6) == (48, -402, 729)
11 assert mi_funcion(726, 434, 110, 488, 962, 820) == (250060, -541640, 486620)
12 assert mi_funcion(62, 12, 198, 380, 334, 490) == (-60252, 44860, 16148)
13 assert mi_funcion(-85, 807, 964, 462, 101, 474) == (285154, 485658, -381419)
14 assert mi_funcion(746, 466, 396, 910, 138, 289) == (80026, 144766, -321112)
15 assert mi_funcion(-15, 53, 105, 413, 149, 270) == (-1335, 47415, -24124)
16 assert mi_funcion(291, 413, 227, 166, 638, 284) == (-27534, -44962, 117100)
17 assert mi_funcion(192, 362, 397, 249, 598, 50) == (-219306, 89253, 24678)
18 assert mi_funcion(781, 520, 996, 348, 68, 215) == (44072, 178693, -127852)
19 assert mi_funcion(459, 971, 201, 582, 569, 703) == (568244, -205695, -303951)
20 assert mi_funcion(754, 968, 956, 231, 901, -31) == (-891364, 244210, 455746)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: bash +
barberis@barberisThinkpad:~/algo$ python3 prodvect.py
Traceback (most recent call last):
  File "prodvect.py", line 11, in <module>
    assert mi_funcion(726, 434, 110, 488, 962, 820) == (250060, -541640, 486620)
AssertionError
barberis@barberisThinkpad:~/algo$
```

Una vez depurado los errores y renombrando las variables y funciones con nombres más acordes y descriptivos, el código queda así.

```
prodvect.py X
prodvect.py > ...
1 #Nombres de variables reemplazadas por unas más acordes/descriptivas
2 def productoVectorial(x1, y1, z1, x2, y2, z2):
3     """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""
4     resultadoX = y1*z2 - z1*y2
5     resultadoY = z1*x2 - x1*z2
6     resultadoZ = x1*y2 - y1*x2
7     return resultadoX, resultadoY, resultadoZ
8
9
10 assert productoVectorial(54, 12, 29, 1, 11, 12) == (-175, -619, 582)
11 assert productoVectorial(71, 52, 24, 1, 11, 6) == (48, -402, 729)
12 assert productoVectorial(726, 434, 110, 488, 962, 820) == (250060, -541640, 486620)
13 assert productoVectorial(62, 12, 198, 380, 334, 490) == (-60252, 44860, 16148)
14 assert productoVectorial(-85, 807, 964, 462, 101, 474) == (285154, 485658, -381419)
15 assert productoVectorial(746, 466, 396, 910, 138, 289) == (80026, 144766, -321112)
16 assert productoVectorial(-15, 53, 105, 413, 149, 270) == (-1335, 47415, -24124)
17 assert productoVectorial(291, 413, 227, 166, 638, 284) == (-27534, -44962, 117100)
18 assert productoVectorial(192, 362, 397, 249, 598, 50) == (-219306, 89253, 24678)
19 assert productoVectorial(781, 520, 996, 348, 68, 215) == (44072, 178693, -127852)
20 assert productoVectorial(459, 971, 201, 582, 569, 703) == (568244, -205695, -303951)
21 assert productoVectorial(754, 968, 956, 231, 901, -31) == (-891364, 244210, 455746)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: bash +
barberis@barberisThinkpad:~/algo$ python3 prodvect.py
barberis@barberisThinkpad:~/algo$
```

¿Se puede escribir el cuerpo de la función en una línea? ¿Cómo?

Sí, se puede. Donde va un salto de línea, solo es necesario agregar un punto y coma (;) y será suficiente para que la función se ejecute en una línea.

Parte 5: Reutilizando funciones

Creo el archivo '**vectores.py**' y coloco allí todas las funciones utilizadas anteriormente.

```
vector.es.py x
vector.es.py > ...
1  def diferencia(x1, y1, z1, x2, y2, z2):
2      """Recibe las coordenadas de dos vectores en R3 y devuelve su diferencia"""
3      dif_x = x1 - x2
4      dif_y = y1 - y2
5      dif_z = z1 - z2
6      return dif_x, dif_y, dif_z
7
8  def norma(x, y, z):
9      """Recibe un vector en R3 y devuelve su norma"""
10     return (x**2 + y**2 + z**2) ** 0.5
11
12  def productoVectorial(x1, y1, z1, x2, y2, z2):
13     """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""
14     resultadoX = y1*z2 - z1*y2
15     resultadoY = z1*x2 - x1*z2
16     resultadoZ = x1*y2 - y1*x2
17     return resultadoX, resultadoY, resultadoZ
18
```

Se pide que se resuelva el ejercicio 3.4.d de la guía de ejercicios utilizando las funciones del módulo '**vectores.py**'. La resolución del ejercicio se realiza en un archivo llamado **area_triangulo.py**.

Defino la función `areaTriangulo()` dentro de mi archivo **area_triangulo.py** y realizo 5 pruebas para comprobar que funciona (mediante el intérprete):

```
>>> areaTriangulo(5, 8, -1, -2, 3, 4, -3, 3, 0)
Resultado del área del triángulo: 19.45507645834372
(ejemplo que aparece en la guía)
```

```
>>> areaTriangulo(1, 0, 1, 2, 2, 2, 0, 0, 0)
Resultado del área del triángulo: 1.4142135623730951
```

```
>>> areaTriangulo(5, 6, 7, 5, 4, 3, 1, 1, 1)
Resultado del área del triángulo: 9.797958971132712
```

```
>>> areaTriangulo(1, 7, 4, 23, 54, 66, 89, 54, 1)
Resultado del área del triángulo: 3515.96050176904
```

```
>>> areaTriangulo(67, 1, 2, 0, 0, 0, 1, 1, 0)
Resultado del área del triángulo: 33.03028912982749
```

El reutilizado de funciones:

Este ejercicio fue realizado en su totalidad utilizando las tres funciones que habíamos utilizado antes. La importancia de reutilizar funciones se ve cuando necesitamos realizar una operación varias veces durante el desarrollo de nuestro programa, y podemos prescindir de realizar las operaciones cada vez que lo requerimos. Cuando ya tengo definida una función, solo basta con llamarla y pasarle los parámetros necesarios, para que esta me devuelva un resultado.