

EJ1 - Area de poligonos

Algoritmos y Programación I (7540)

Alumno: Gomez, Emiliano

Padrón: 110042

Curso: Essaya

Práctica: Barbara

Ayudante a cargo: Cano, Matías

Fecha de entrega: 2/9/2022



**FACULTAD
DE INGENIERIA**

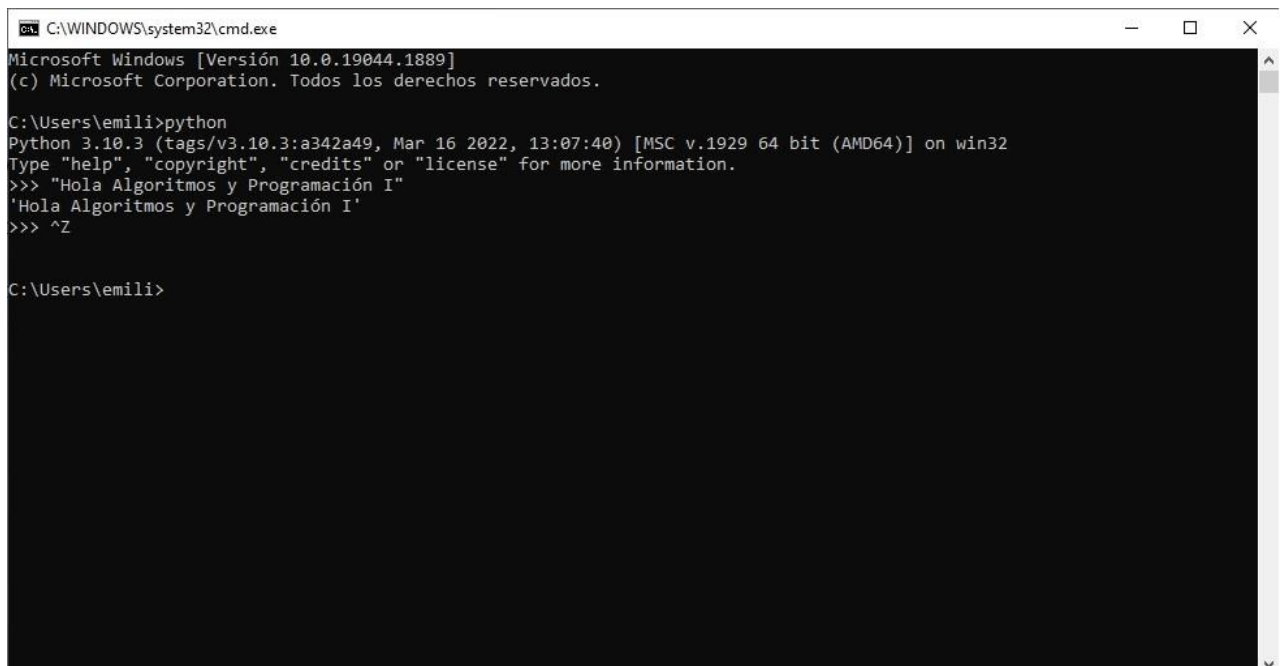
Universidad de Buenos Aires

Parte 1: Entorno de Trabajo

El entorno de trabajo utilizado para las pruebas es Windows 10, con Visual Studio Code como editor de texto y el CMD como terminal.

Parte 1.1

Primero se pide ingresar al intérprete de Python para luego correr la siguiente línea de código: ***"Hola Algoritmos y Programación I"***. Una vez hecho esto, se concluye la operación saliendo del intérprete utilizando ***"Ctrl. + z"*** (También se puede realizar la misma acción utilizando la función ***exit()***).



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.19044.1889]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\emili>python
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> "Hola Algoritmos y Programación I"
'Hola Algoritmos y Programación I'
>>> ^Z

C:\Users\emili>
```

Parte 1.2

Se crea el archivo ***"parte_1_2.py"*** al cual se le agrega la línea de código ***"Hola Algoritmos y Programación I"*** para luego ser ejecutado con el comando ***"python parte_1_2.py"***

```
C:\Users\emili\OneDrive\Escritorio\Algoritmos>python parte_1_2.py  
C:\Users\emili\OneDrive\Escritorio\Algoritmos>
```

¿Qué función debo usar para conseguir el mismo resultado que en la parte 1.1? ¿Por qué en la parte 1.1 vemos el resultado aun sin haber usado esta función?

Para obtener el mismo resultado de la parte anterior se debe utilizar la función **print()**, de este modo el resultado saldría por pantalla. La diferencia radica en que en la primer parte se está ingresando al **modo interactivo** de python, el cual sirve para hacer algunas pruebas menores y tiene ciertas ventajas como devolver ciertas operaciones por pantalla sin la necesidad de utilizar **print()**, sin embargo, en la segunda parte al estar almacenando esa información en un archivo **“.py”** y luego ejecutandolo con la consola mediante el intérprete de python si es necesario utilizar esa función. A continuación se muestra la corrección en el archivo y el resultado por consola.

```
parte_1_2.py X  
C: > Users > emili > OneDrive > Escritorio > EJ1 > parte_1_2.py  
1 print("Hola Algoritmos y Programación I")  
  
C:\Users\emili\OneDrive\Escritorio\EJ1>python parte_1_2.py  
Hola Algoritmos y Programación I  
C:\Users\emili\OneDrive\Escritorio\EJ1>_
```

Parte 2: norma.py

Se ejecuta el programa *norma.py* el cual muestra lo siguiente.

```
C:\Users\emili\OneDrive\Escritorio\EJ1>python norma.py  
C:\Users\emili\OneDrive\Escritorio\EJ1>
```

Luego, se des-comentan las últimas 2 líneas del código, lo cual produce el siguiente resultado.

```
C:\Users\emili\OneDrive\Escritorio\EJ1>python norma.py  
Traceback (most recent call last):  
  File "C:\Users\emili\OneDrive\Escritorio\EJ1\norma.py", line 17, in <module>  
    assert norma(-70, 14, z) == 111.0  
AssertionError  
C:\Users\emili\OneDrive\Escritorio\EJ1>_
```

¿Cuál es la salida del programa? ¿Podemos saber en qué línea se generó el error? ¿Cómo? ¿Qué hace la instrucción assert? Solucionar el problema. Hallar el valor de z para que ya no de error.

El programa devuelve la norma de un vector en R3.

Como muestra la imagen, se puede observar que el error está en la línea 17 del código (*line 17*).

La instrucción **assert** sirve para hacer comprobaciones y, si lo que está contenido dentro del mismo es **False** devuelve una excepción.

Para solucionar este error se debe cambiar el valor de **z** adecuado para que se cumpla la igualdad. Despejando de la ecuación se obtiene que $|z| = 85$, por lo que hay 2 valores de **z** que cumplen con la igualdad siendo **$z_1 = 85$** , **$z_2 = -85$** .

Parte 3: diferencia.py

Se abre el archivo *diferencia.py* y se prueban los casos asignados, dando como resultado.

```
C:\Users\emili\OneDrive\Escritorio\EJ1>python diferencia.py
Traceback (most recent call last):
  File "C:\Users\emili\OneDrive\Escritorio\EJ1\diferencia.py", line 9, in <module>
    assert diferencia(16, -72, -52, 55, 90, -31) == (-39, -162, -21)
  File "C:\Users\emili\OneDrive\Escritorio\EJ1\diferencia.py", line 6, in diferencia
    return dif_x, dif_y, dif_f_z
NameError: name 'dif_f_z' is not defined. Did you mean: 'dif_z'?
```

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

Se detectó un error en la línea 9 del código dando un ***NameError*** debido a que el nombre de una variable estaba mal escrita y el intérprete no podía ejecutarla. Para solucionar este problema se cambia el nombre de la variable ***diff z*** por ***dif z***.

Parte 4: Depuración

Se abre y ejecuta el archivo **prod_vect.py** el cual muestra lo siguiente.

```
C:\Users\emili\OneDrive\Escritorio\EJ1>python prod_vect.py
Traceback (most recent call last):
  File "C:\Users\emili\OneDrive\Escritorio\EJ1\prod_vect.py", line 10, in <module>
    assert mi_funcion(726, 434, 110, 488, 962, 820) == (250060, -541640, 486620)
AssertionError
```

¿Qué error muestra? ¿En qué línea?

Se muestra un error en la **línea 10** del tipo ***AssertionError***. Para depurar el programa primero se comenta la línea donde está el error para luego utilizar la función **print()** y ver el resultado por pantalla, obteniendo como resultado lo siguiente

```
10 print(producto_vectorial(726, 434, 110, 488, 962, 820))
11 #assert producto_vectorial(726, 434, 110, 488, 962, 820) == (250060, -541640, 486620)
```

[illegible]

Se observa que el resultado en la componente **Y** da un número muy grande, el cual no cumple con la igualdad planteada anteriormente, y otro **AssertionError** en la línea siguiente. Después de revisar las variables de la función se encuentra un error de tipo en una de ellas y se reemplaza ****** (que significa potencia) por ***** (que significa multiplicar). Luego, el programa funciona sin problemas.

Renombrar la función y las variables de forma que sus nombres sean representativos. ¿Por qué es importante hacer esto?

Es importante que las funciones y variables tengan nombres significativos debido a que al escribir código este debe ser legible y fácil de interpretar, ya que muchas veces este va a ser revisado por otra persona la cual es posible que nunca haya visto el código y deba interpretarlo.

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

Si se puede, utilizando **punto y coma (;)** para separar las distintas variables, sin embargo, esto lleva a un código difícil de leer. A continuación se mostrarán las diferencias entre una forma y otra.

```
def producto_vectorial(x1, y1, z1, x2, y2, z2):  
    """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""  
    x3 = y1*z2 - z1*y2 ; y3 = z1*x2 - x1*z2 ; z3 = x1*y2 - y1*x2 ; return x3, y3, z3
```

```
def producto_vectorial(x1, y1, z1, x2, y2, z2):  
    """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""  
    x3 = y1*z2 - z1*y2  
    y3 = z1*x2 - x1*z2  
    z3 = x1*y2 - y1*x2  
    return x3, y3, z3
```

Si bien en la primera imagen se observa un código más compacto, en la segunda es mucho más fácil de interpretar lo que la función hace.

También se podría escribir toda la lógica dentro del **return**, obteniendo.

```
def producto_vectorial(x1, y1, z1, x2, y2, z2):  
    """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""  
    return y1*z2 - z1*y2, z1*x2 - x1*z2, x1*y2 - y1*x2
```

Parte 5: Reutilizando funciones

Se crea el archivo **vectores.py** donde se ponen todas las funciones anteriores.

```
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 producto_vectorial(x1, y1, z1, x2, y2, z2):
9     """Recibe las coordenadas de dos vectores en R3 y devuelve el producto vectorial"""
10    res1 = y1*z2 - z1*y2
11    res2 = z1*x2 - x1*z2
12    res3 = x1*y2 - y1*x2
13    return res1, res2, res3
14
15 def norma(x, y, z):
16     """Recibe un vector en R3 y devuelve su norma"""
17     return (x**2 + y**2 + z**2) ** 0.5
18
```

Se pide resolver el ejercicio 3.4.d de la guía re-utilizando las funciones anteriores. Para ello, se crea el archivo **area_triangulo.py** y se realizan 5 pruebas utilizando `assert`. Se observa que la función devuelve un *int*, esto se hace para facilitar las pruebas debido a que en algunas de ellas los decimales eran demasiados.

```
1 from vectores import diferencia, producto_vectorial, norma
2
3 def area_triangulo(x1, y1, z1, x2, y2, z2, x3, y3, z3):
4     #Devuelve el area de un triangulo dados 3 vectores en el espacio R3
5
6     ab_x, ab_y, ab_z = diferencia(x1, y1, z1, x2, y2, z2)
7     ac_x, ac_y, ac_z = diferencia(x1, y1, z1, x3, y3, z3)
8
9     res_x, res_y, res_z = producto_vectorial(ab_x, ab_y, ab_z, ac_x, ac_y, ac_z)
10
11    norma_res = norma(res_x, res_y, res_z)
12
13    area = norma_res / 2
14
15    return float(area)
16
17 assert(area_triangulo(5,8,-1,-2,3,4,-3,3,0)) == 19.45507645834372
18 assert(area_triangulo(1,2,3,4,5,6,7,8,9)) == 0.0
19 assert(area_triangulo(2,7,8,26,89,10,6,2,1)) == 370.7344062802912
20 assert(area_triangulo(1,7,8,9,0,4,21,70,698)) == 3630.868353438334
21 assert(area_triangulo(89,75,97,90,53,35,28,5,3)) == 2354.7348046011466
```

¿Cuál es la importancia de reutilizar funciones?

Reutilizar funciones es importante sobre todo cuando necesitamos realizar distintas operaciones y programas más complejos. A su vez, esto permite **optimizar tiempo**, debido a que no será necesario re-escribir las operaciones, con llamar las funciones ya escritas y pasar los parámetros correspondientes para obtener el resultado.

Por otro lado, las funciones nos ayudan a **reutilizar** código y **evitar repeticiones**. Esto además permite que si hay un bug, solo hay que **fixearlo** en un lugar, obteniendo como resultado un código más fácil de mantener a lo largo del tiempo.

Notas de re-entrega:

- Cambio en el código area_triangulo.py (Desempaquetado y tipo de dato que devuelve)
- Correcciones tomadas en cuenta y agregadas en los items correspondientes