

Ejercicio 1 – Área de Polígonos

Materia: 75.40 - Algoritmos y Programación I

Catedra: 04 – Essaya

Práctica: Barbara

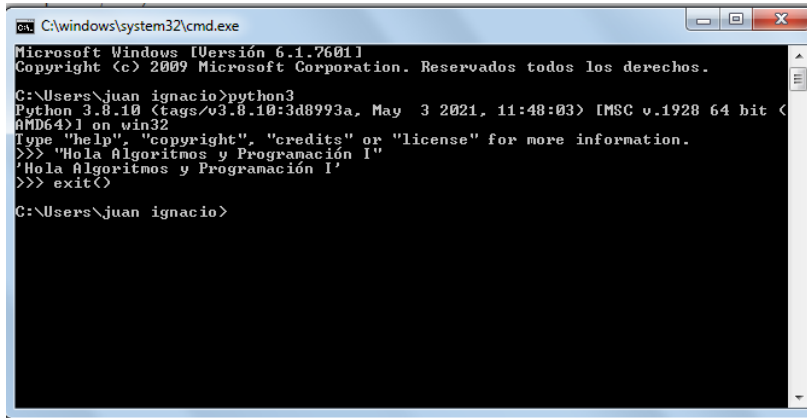
Alumno: Juan Ignacio Martínez

Padrón: 94785

Corrector: Franco Lighterman

Parte 1: Entorno de Trabajo

Parte 1.1



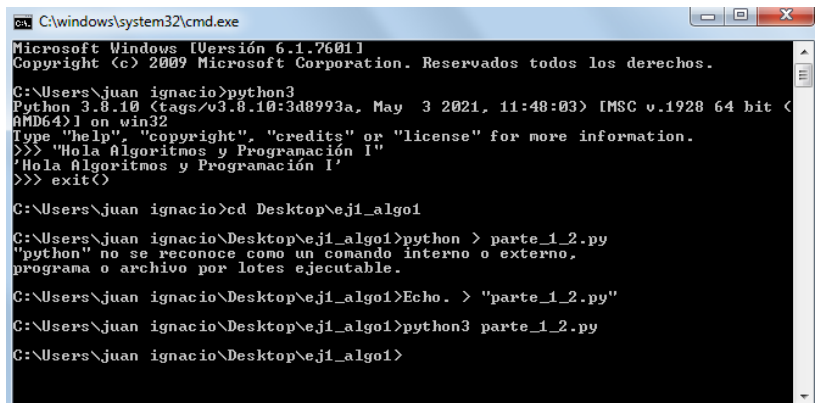
```
C:\windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\juan ignacio>python3
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hola Algoritmos y Programación I"
'Hola Algoritmos y Programación I'
>>> exit()

C:\Users\juan ignacio>
```

Parte 1.2

1.2.4



```
C:\windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\juan ignacio>python3
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> "Hola Algoritmos y Programación I"
'Hola Algoritmos y Programación I'
>>> exit()

C:\Users\juan ignacio>cd Desktop\ej1_algo1
C:\Users\juan ignacio\Desktop\ej1_algo1>python > parte_1_2.py
"python" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
C:\Users\juan ignacio\Desktop\ej1_algo1>Echo. > "parte_1_2.py"
C:\Users\juan ignacio\Desktop\ej1_algo1>python3 parte_1_2.py
'Hola Algoritmos y Programación I'
C:\Users\juan ignacio\Desktop\ej1_algo1>
```

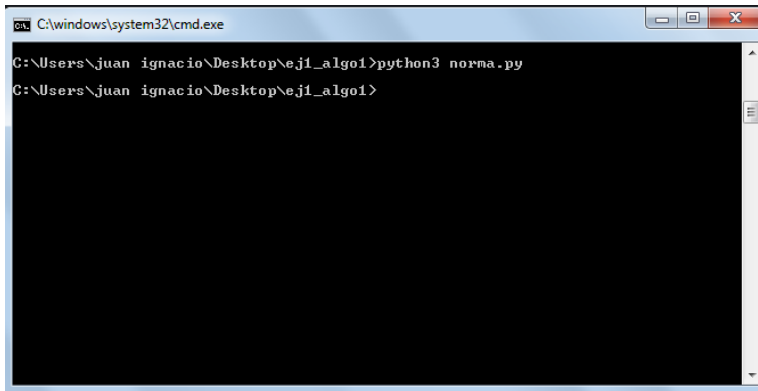
1.2.5. Para obtener el mismo resultado que en 1.1 debo utilizar la función Print. Es decir, debo cambiar el código dentro del archivo por:

Print("Hola Algoritmos y programación I")

En 1.1 vemos el resultado sin haber usado la función print porque estamos dentro del intérprete de Python que nos imprime el resultado de una expresión luego de evaluarla. En cambio, cuando Python ejecuta un programa .py no imprime nada en la pantalla a menos que le indiquemos que lo haga. Por eso en parte_1_2.py llamo a la función print.

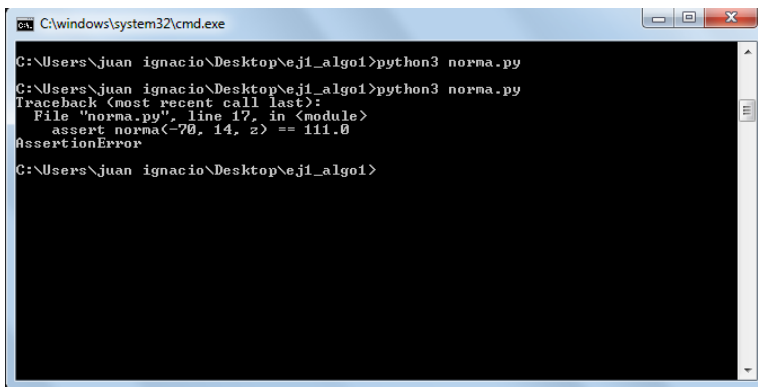
Parte 2: norma.py

2.2



```
C:\windows\system32\cmd.exe
C:\Users\juan_ignacio\Desktop\ej1_algo1>python3 norma.py
C:\Users\juan_ignacio\Desktop\ej1_algo1>
```

2.5



```
C:\windows\system32\cmd.exe
C:\Users\juan_ignacio\Desktop\ej1_algo1>python3 norma.py
C:\Users\juan_ignacio\Desktop\ej1_algo1>python3 norma.py
Traceback (most recent call last):
  File "norma.py", line 17, in <module>
    assert norma(-70, 14, z) == 111.0
AssertionError
C:\Users\juan_ignacio\Desktop\ej1_algo1>
```

2.5.1

La salida del programa es: AssertionError

2.5.2

En la terminal me indica que el error se produjo en la línea 17 de mi archivo donde se encuentra escrita la siguiente línea de código:

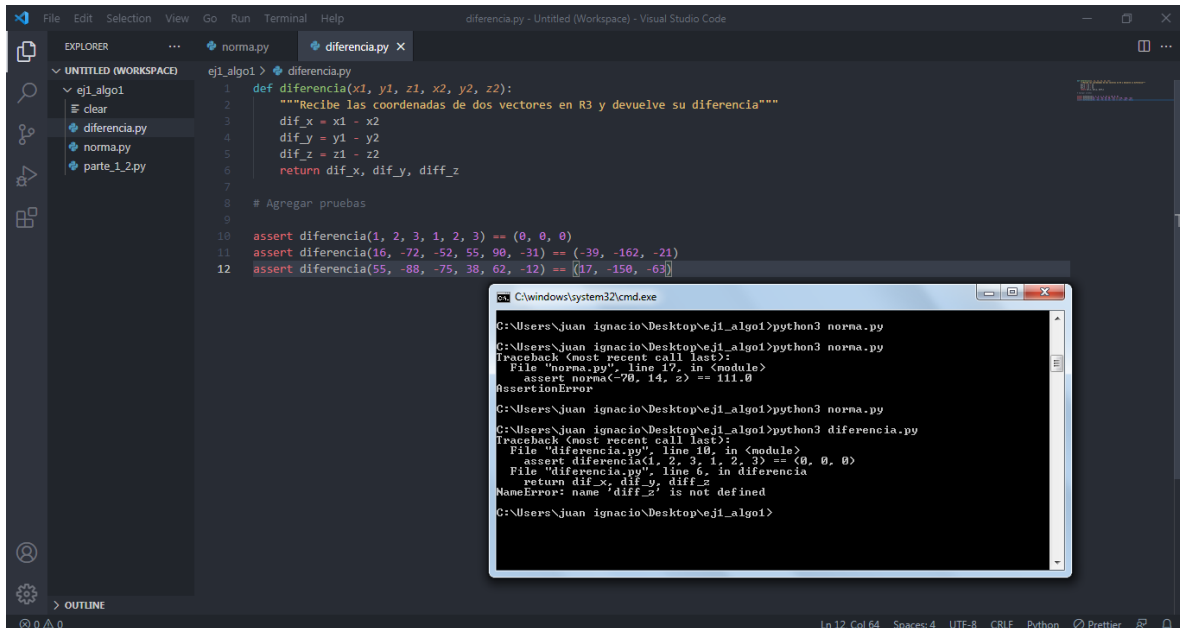
```
assert norma(-70, 14, z) == 111.0
```

2.5.3

La instrucción assert recibe una expresión booleana. Si esta expresión devuelve true, la instrucción assert no hace nada, pero si la expresión devuelve false se produce un error.

Parte 3: Diferencia.py

3.3



The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows a workspace with files: 'ej1_algo1', 'diferencia.py', 'norma.py', and 'parte_1_2.py'. The main editor displays the code in 'diferencia.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 # Agregar pruebas
9
10 assert diferencia(1, 2, 3, 1, 2, 3) == (0, 0, 0)
11 assert diferencia(16, -72, -52, 55, 90, -31) == (-39, -162, -21)
12 assert diferencia(55, -88, -75, 38, 62, -12) == (17, -150, -63)
```

The terminal window shows the following output:

```
C:\Windows\system32\cmd.exe
C:\Users\Juan Ignacio\Desktop\ej1_algo1>python3 norma.py
C:\Users\Juan Ignacio\Desktop\ej1_algo1>python3 norma.py
Traceback (most recent call last):
  File "norma.py", line 17, in <module>
    assert norma(-70.14, z) == 111.0
AssertionError

C:\Users\Juan Ignacio\Desktop\ej1_algo1>python3 norma.py
C:\Users\Juan Ignacio\Desktop\ej1_algo1>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

C:\Users\Juan Ignacio\Desktop\ej1_algo1>
```

3.4

Se detectó un error en la línea 6 que se debe a que no tenemos definida la variable `diff_z` cuyo valor intentamos devolver. La variable definida dentro de la función lleva el nombre de `dif_z`.

Parte 4: Depuración

4.4

La terminal muestra el error `AssertionError` en la línea 10 de mi código.

4.6

Es importante nombrar las funciones y las variables de manera tal que sus nombres sean representativos ya que esté mismo código lo puede tener que leer un compañero de trabajo o de proyecto y es importante que se entienda lo mejor posible. También puede pasar que lo tenga que leer uno mismo en un futuro y si los nombres no están bien definidos va a llevar más trabajo recordar y entender que es lo que hace el código.

4.7

Si, se puede escribir todo el cuerpo de una función en una sola línea. En vez de almacenar en distintas variables los valores que me devuelven las expresiones, puedo escribir directamente las expresiones en la línea del return. Por ejemplo, de la siguiente manera:

```
def mi_funcion(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

5.4

Es importante reutilizar funciones porque no es una buena práctica repetir código cuando no es necesario.