



ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD: INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

CARRERA: SOFTWARE

CASOS DE PRUEBA

FECHA

05-10-2022

INTEGRANTES

JUAN ROMAN 6751

CHRISTIAN OBANDO 6711

FABRICIO RODRIGUEZ 6491

Descripción del producto de software

Desarrollo de un software con la capacidad de obtener las raíces reales y complejas de un polinomio válido de 2do grado, donde se deberá ingresar valores por parte del usuario.

Funcionalidades

F1	El sistema deberá permitir el ingreso de datos numéricos.
F2	El sistema calcula las raíces reales (iguales o diferentes) y complejas (conjugadas).

Enlace del código:

<https://github.com/CrissCraxz/Polinomio-y-Triangulo/tree/main/Polynomial>

Funcionalidad de Ingreso y validación de datos

Reglas : $((a,b,c) \in \mathbb{R}) \quad \& \quad ((a,b,c) \leq 2^80) \quad \& \quad (a \neq 0)$

Casos de prueba

Caso de prueba	Entradas	Salida
Caso de prueba 1	(0,1,2)	(mensaje “el coeficiente no puede ser igual a 0”)

Captura de prueba unitaria

```
test.py > ...
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5  def test_polinomio(self):
6      a = 0
7      b = 10
8      c = 2
9      self.assertEqual(ecuacionSegundoGrado(a,b,c),"El coeficiente a no puede ser igual a cero")
10
11 if __name__ == "__main__":
12     unittest.main()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER: VARIABLES

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
El coeficiente a no puede ser igual a cero
.

Ran 1 test in 0.000s
OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Caso de prueba	Entradas	Salida
Caso de prueba 2	(a,1,2)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```
test.py > TestPol > test_polinomio
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5  def test_polinomio(self):
6      a = "a"
7      b = 1
8      c = 2
9      self.assertEqual(ecuacionSegundoGrado(a,b,c),"valor ingresado no numerico")
10 if __name__ == "__main__":
11     unittest.main()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER: VARIABLES

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.

Ran 1 test in 0.000s
OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Caso de prueba	Entradas	Salida
Caso de prueba 3	(,1,2)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

test.py > ...
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5      def test_polinomio(self):
6          a = ""
7          b = 1
8          c = 2
9          self.assertEqual(ecuacionSegundoGrado(a,b,c),"valor ingresado no numerico")
10
11 if __name__ == "__main__":
12     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

```

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.
-----
Ran 1 test in 0.000s

OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> 

```

Caso de prueba	Entradas	Salida
Caso de prueba 4	(b, v, 4)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

test.py > TestPol > test_polinomio
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5      def test_polinomio(self):
6          a = "b"
7          b = "v"
8          c = 4
9          #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a))))
10         #self.assertEqual(ecuacionSegundoGrado(a,b,c),"raices imaginarias")
11         self.assertEqual(ecuacionSegundoGrado(a,b,c),("valor ingresado no numerico"))
12
13 if __name__ == "__main__":
14     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

```

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.
-----
Ran 1 test in 0.000s

OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> 

```

Caso de prueba	Entradas	Salida
Caso de prueba 5	(5, v, 4)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

testpy > TestPol > test_polinomio
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5      def test_polinomio(self):
6          a = 5
7          b = "v"
8          c = 4
9          #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a)))
10         #self.assertEqual(ecuacionSegundoGrado(a,b,c),("raices imaginarias"))
11         self.assertEqual(ecuacionSegundoGrado(a,b,c),("valor ingresado no numerico"))
12 if __name__ == "__main__":
13     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER: VARIABLES

PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.

Ran 1 test in 0.000s
OK
PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Caso de prueba	Entradas	Salida
Caso de prueba 6	(v, 5, v)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

testpy > ...
1  import math
2  import unittest
3  from polinomio import ecuacionSegundoGrado
4  class TestPol(unittest.TestCase):
5      def test_polinomio(self):
6          a = "v"
7          b = 5
8          c = "v"
9          #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a)))
10         #self.assertEqual(ecuacionSegundoGrado(a,b,c),("raices imaginarias"))
11         self.assertEqual(ecuacionSegundoGrado(a,b,c),("valor ingresado no numerico"))
12 if __name__ == "__main__":
13     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL JUPYTER: VARIABLES

PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.

Ran 1 test in 0.000s
OK
PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Caso de prueba	Entradas	Salida
Caso de prueba 7	(6, 5, v)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

testpy > TestPol > test_polinomio
1 import math
2 import unittest
3 from polinomio import ecuacionSegundoGrado
4 class TestPol(unittest.TestCase):
5     def test_polinomio(self):
6         a = 6
7         b = 5
8         c = "v"
9         #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a)))
10        #self.assertEqual(ecuacionSegundoGrado(a,b,c),("raices imaginarias"))
11        self.assertEqual(ecuacionSegundoGrado(a,b,c),("valor ingresado no numerico"))
12 if __name__ == "__main__":
13     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.

Ran 1 test in 0.000s
OK
PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Caso de prueba	Entradas	Salida
Caso de prueba 8	(t, 5, 6)	(mensaje “valor ingresado no numérico”)

Captura de prueba unitaria

```

testpy > TestPol > test_polinomio
1 import math
2 import unittest
3 from polinomio import ecuacionSegundoGrado
4 class TestPol(unittest.TestCase):
5     def test_polinomio(self):
6         a = "t"
7         b = 5
8         c = 6
9         #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a)))
10        #self.assertEqual(ecuacionSegundoGrado(a,b,c),("raices imaginarias"))
11        self.assertEqual(ecuacionSegundoGrado(a,b,c),("valor ingresado no numerico"))
12 if __name__ == "__main__":
13     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
valor ingresado no numerico
.

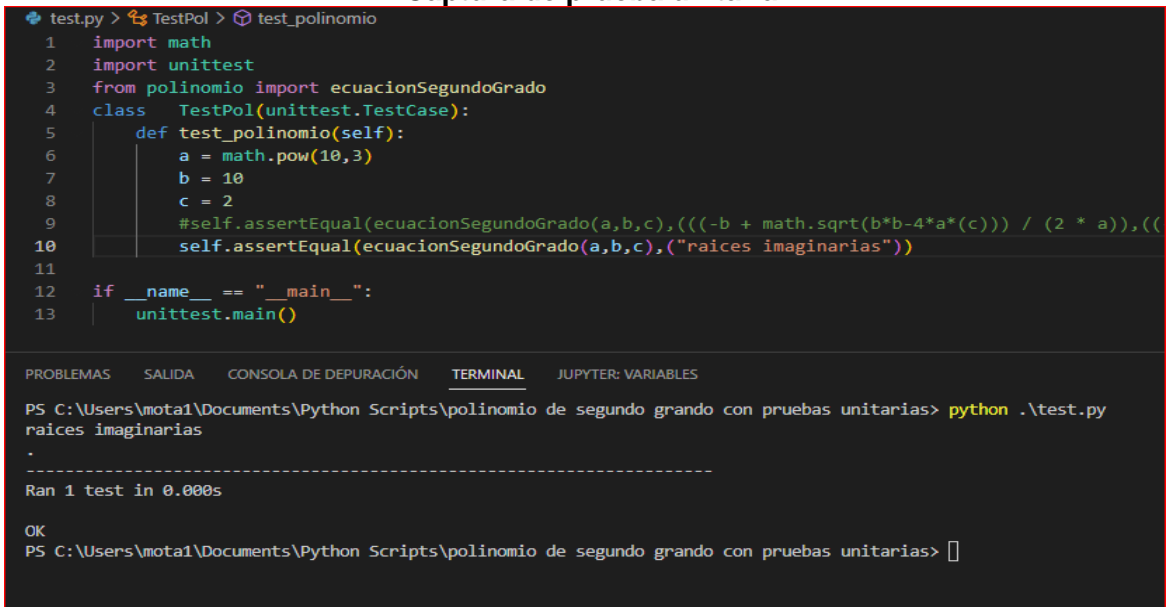
Ran 1 test in 0.000s
OK
PS C:\Users\motai\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Funcionalidad 2 de calcula las raíces reales (iguales o diferentes) y complejas (conjugadas).

Reglas : $((a,b,c) \in \mathbb{R}) \quad \& \quad ((a,b,c) \leq 2^80) \quad \& \quad (a \neq 0)$

Caso de prueba	Entradas	Salida
Caso de prueba 9	(1000,10,2)	(mensaje “raíces imaginarias”)

Captura de prueba unitaria



```

test.py > TestPol > test_polinomio
1 import math
2 import unittest
3 from polinomio import ecuacionSegundoGrado
4 class TestPol(unittest.TestCase):
5     def test_polinomio(self):
6         a = math.pow(10,3)
7         b = 10
8         c = 2
9         #self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((
10        self.assertEqual(ecuacionSegundoGrado(a,b,c),("raíces imaginarias"))
11
12 if __name__ == "__main__":
13     unittest.main()

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

```

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
raíces imaginarias
.
-----
Ran 1 test in 0.000s

OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

```

Caso de prueba	Entradas	Salida
Caso de prueba 10	(1,-1,-6)	(3,-2)

Captura de prueba unitaria

```
test.py > TestPol > test_polinomio
1 import math
2 import unittest
3 from polinomio import ecuacionSegundoGrado
4 class TestPol(unittest.TestCase):
5     def test_polinomio(self):
6         a = 1
7         b = -1
8         c = -6
9         self.assertEqual(ecuacionSegundoGrado(a,b,c),((-b + math.sqrt(b*b-4*a*(c))) / (2 * a)),((-b - math.sqrt(b*b-4*a*(c))) / (2 * a)))
10        #self.assertEqual(ecuacionSegundoGrado(a,b,c),("raices imaginarias"))
11
12 if __name__ == "__main__":
13     unittest.main()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER: VARIABLES

PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias> python .\test.py
La raíz real (+) es = 3.0
La raíz real (-) es = -2.0
.

Ran 1 test in 0.000s
OK
PS C:\Users\mota1\Documents\Python Scripts\polinomio de segundo grado con pruebas unitarias>

Matriz de trazabilidad

	F1	F2
Cp1	X	
Cp2	X	
Cp3	X	
Cp4	X	
Cp5	X	
Cp6	X	
Cp7	X	
Cp8	X	
Cp9		X
Cp10		X