Quadratic Equations

1. `Rational` class

The `Rational` class has 2 private variables to represent the numerator and denominator of a fraction.

Define necessary accessors and mutators associated with the class. In addition, it also has but not limited to the following methods.

Method	Description

	The following examples are using `r1` and `r2` for demonstration purposes.
	`r1 = Rational(3)`
	`r2 = Rational(6, 4)`

`get_value()`	Returns the floating-point value of the rational number.
	e.g.
	`print(r1.get_value())  # 3.0`
	`print(r2.get_value())  # 1.5`

`__str__()`	Returns string representation of the number in its fractional form.
	e.g.
	`print(r1)  # 3`
	`print(r2)  # (3/2)`

`__add__(other)`
`__mul__(other)`
`__neg__(other)`
`__sub__(other)`
`__truediv__(other)`
	Magic methods related to mathematical operations.
	e.g.
	`print(r1 + r2)  # (9/2)`
	`print(r1 * r2)  # (9/2)`
	`print(-r2)  # (-3/2)`
	`print(r1 - r2)  # (3/2)`
	`print(r1 / r2)  # 2`

`__gt__(other)`
`__eq__(other)`
`__lt__(other)`
	Magic methods related to comparison operations.
	e.g.
	`print(r1 > r2)  # True`
	`print(r1 == r2)  # False`
	`print(r1 < r2)  # False`

`get_sqrt()`	Add this method to class `Rational` **after** you have implemented the `Real` class. Returns a `Real` object which stores the square root of the current rational number in its exact form. *e.g.* `print(r2.get_sqrt())  # (1/2)√6`

2. `Real` class

The `Real` class is a subclass of `Rational` class. On top of the current 2 variables, it has a 3rd private variable storing the irrational value under the square root sign.

Define necessary accessors and mutators associated with the class. In addition, it also has but not limited to the following methods.

Method	**Description**

	The following examples are using `ir1` to `ir5` for demonstration purposes.
	`ir1 = Real(3)`
	`ir2 = Real(6, 4)`
	`ir3 = Real(3, 1, 2)`
	`ir4 = Real(6, 4, 2)`
	`ir5 = Real(6, 4, 3)`

`get_value()`	Returns the floating-point value of the real number. *e.g.* `print(ir3.get_value())   # 4.242640687119286`

`__str__()`	Returns string representation of the number in its exact form.
	*Note: "`\u221A`" is the Unicode value for the sign " √ ".
	e.g.
	`print(ir1)  # 3`
	`print(ir2)  # (3/2)`
	`print(ir3)  # 3√2`
	`print(ir4)  # (3/2)√2`

`__add__(other)`
`__mul__(other)`
`__neg__(other)`
`__sub__(other)`
`__truediv__(other)`
	Magic methods related to mathematical operations.
	For now, you may assume the add and sub operation can only operate on real numbers with the same value under the square root sign in their exact forms. You should improve this once you finish implementing the RealSeq class.
	e.g.
	`print(ir3 + ir4)  # (9/2)√2`
	`print(ir3 * ir5)  # (9/2)√6`
	`print(-ir4)  # (-3/2)√2`
	`print(ir4 - ir3)  # (-3/2)√2`
	`print(ir3 / ir5)  # (2/3)√6`

3. `RealSeq` class

The `RealSeq` class is used to store a sequence of real of objects when their value under the square root sign are different.

Define necessary accessors and mutators associated with the class. In addition, it also has but not limited to the following methods.

Method	**Description**

	The following examples are using `ir1` to `ir5` for demonstration purposes.
	`ir3 = Real(3, 1, 2)`
	`ir5 = Real(6, 4, 3)`
	`rs1 = ir3 + ir5`

`get_value()`	Returns the floating-point value of all the real numbers stored in the sequence. *e.g.* `print(rs1.get_value())   # 6.840716898472602`
`__str__()`	Returns string representation of the sequence of real numbers in their exact forms. *e.g.* `print(rs1)   # 3√2 + (3/2)√3`

4. `QuadraticEquation` class

The `QuadraticEquation` class is used to store the coefficients in a quadratic equation, namely the `a`, `b` and `c` values of the form `ax^2 + bx + c = 0`.

Define necessary accessors and mutators associated with the class. In addition, it also has but not limited to the following methods.

Method	Description

	The following example is used for demonstration purposes.
	`a = Real(3, 2)`
	`b = Real(5)`
	`c = Real(1)`
	`qe1 = QuadraticEquation(a, b, c)`

`has_roots()`	Returns `True` if there are real roots for the equation, `False` otherwise. *e.g.* `print(qe1.has_roots())  # True`

`__str__()`	Display the equation in its human readable form:
	`ax^2 + bx + c = 0`
	e.g.
	`print(qe1)   # (3/2)x^2 + 5x + 1 = 0`

`get_roots_values()`	Returns the floating-point values of the 2 real roots as a tuple.
	e.g.
	`print(qe1.get_roots_values())`
	`# (-0.2137003521531089, -3.1196329811802244)`

`get_roots()`	Returns the exact form of 2 real roots as a tuple.
	e.g.
	`print(qe1.get_roots())`
	`# ((-5/3) + (1/3)√19, (-5/3) + (-1/3)√19)`

`Complete_sq()`	Prints out the break-down of steps to take if one is to use complete the square method to solve a quadratic equation.
	`print(qe1.complete_sq())`
	`# Solving following equation using complete the square method`
	`# (3/2)x^2 + 5x + 1 = 0`
	`# Step 1: Divide value of a across the equation.`
	`# 1x^2 + (10/3)x = (-2/3)`
	`# Step 2: Add square of b/2 to both sides.`
	`# 1x^2 + (10/3)x + (25/9) = (19/9)`
	`# Step 3: Complete the square.`
	`# (x + (5/3))^2 = (19/9)`
	`# Step 4: Square root both sides.`
	`# x + (-5/3) = (1/3)√19 or x + (-5/3) = -(1/3)√19`
	`# Step 5: Solve for x.`
	`# x = (-5/3) + (1/3)√19 or (-5/3) + (-1/3)√19`

`get_eqn_from_str(s)`	***This is a static method.***
	Returns a `QuadraticEquation` object by reading and processing from a string.
	e.g.
	`qe1 = QuadraticEquation.get_eqn_from_str(`
	`"(3/2)x^2 + 5x + 1 = 0")`

`gen_random_eqn()`	***This is a static method.***
	Generate a random quadratic equation with **rational** values of a, b and c.
	*You should limit the values to a reasonable range.
	e.g.
	`for _ in range(10):`
	`    qe = QuadraticEquation.gen_random_eqn()`
	`    print(qe)`
	
	`# (-5/8)x^2 + (9/11)x + (5/9) = 0`
	`# (-1/2)x^2 + (2/3)x + (-13/4) = 0`
	`# (4/5)x^2 + (8/3)x + (-5/4) = 0`
	`# (-3/7)x^2 + (12/13)x + -1 = 0`
	`# -6x^2 + (7/5)x + (11/5) = 0`
	`# (-2/3)x^2 + (12/5)x + (-1/2) = 0`
	`# (-5/11)x^2 + (11/8)x + 10 = 0`
	`# (-7/3)x^2 + (5/7)x + (-11/15) = 0`
	`# (-7/2)x^2 + -14x + -1 = 0`
	`# (3/13)x^2 + (-1/4)x + -15 = 0`

5. Menu

Design and implement a menu with the following options. Implement necessary input validation functions to ensure smooth execution of program.

```
Welcome to the Quadratic Equation Solver
1. Enter new equation.
2. Solve equation with roots in floating point form.
3. Solve equation with roots in exact form.
4. Solve equation using complete the square method.
5. Generate a series of random equations and save to a file.
6. Read form file, solve all equations, save solutions to solution file.
7. Exit
```

Option	**Description**
`5. Generate a series of random equations and save to a file.`	Take user input to determine number of equations to be generated. *Note: - File name for storing equations should be "`equations.txt`".
`6. Read form file, solve all equations, save solutions to solution file.`	When writing output to the solution file, replace "`\u221A`" with "`|`" as txt file does not support unicode. *Note: - File name for storing equations should be "`equations.txt`". - File name for storing solutions should be "`solutions.txt`".