**Exercise 1: Playing around with Geometry.**

To implement the classes representing: equilateral triangles, circles, rectangles, squares and pentagons with the following characteristics/properties/capabilities.

1. they should understand the calculate\_area() and calculate\_perimeter() messages with the obvious meaning
2. the state must be private
3. a list of geometric shapes must be sortable by area and by perimeter (not at the same time, of course)
4. to add an hexagon class should maintain all the capabilities of the existing classes and correctly interact with them
5. to write a iterator that permits to return the elements of a list of geometric shapes sorted by increasing areas.

**Exercise 2: Playing around with Algebra.**

A **monoid** is an algebraic structure consisting of a set together with a single associative binary operation and an identity element. E.g., the set of booleans with the or operator is a monoid whose identity is false.

A **group** is an algebraic structure consisting of a set together with an operation that combines any two of its elements to form a third element. To qualify as a group, the set and the operation must satisfy a few conditions called group axioms, namely closure, associativity, identity and invertibility. E.g., the set Z (the integers with sign) with the + operator is a group but with the \* is not since inverting the operation break the closure property.

A **ring** is an algebraic structure consisting of a set together with two binary operations (usually called addition and multiplication), where each operation combines two elements to form a third element. To qualify as a ring, the set together with its two operations must satisfy certain conditions, namely, the set must be an abelian (i.e., commutative) group under addition and a monoid under multiplication such that multiplication distributes over addition.

Write the Monoid, Group and Ring classes implementing the monoids, groups and rings algebraic structures respectively. Each class must be general, in the sense that the operations and the sets are not a priori defined and they should implement methods to check the properties characterizing the algebraic structure.

Test your implementation with the following examples (S is the set, add=additive operation, mul=multiplicative operation, i=identity):

**Monoids**

* S = {True, False} add = or i = False
* S = Zn={0,1,...,n-1} add = + where a+b=(a+b)%n i = 0

**Groups**

* S = itertools.permutations('RGB'), given a a function swaps 1st with 2nd element in the permutation, and b a function that swaps 2nd with 3rd element add = function composition i = a(a(x))
* S = Q-{0} add = \* i = 1

**Rings**

* S = {0} add = + where 0+0=0 mul = \* where 0\*0=0 i = 0
* S = Z add = + mul = \* i = 0
* S = Z4 = {0,1,2,3} add = + where a+b=(a+b)%4 mul = \* where a\*b=(a\*b)%4 i = 0

**Note** infinite sets can be implemented through iterators; property checks on infinite sets should be on a finite subset.

**Exercise 3: Playing around with Arithmetic (the Polish Calculator).**

Write a PolishCalculator class that implements a stack-based calculator that adopts polish notation for the expressions to be evaluated.

**Polish Notation** is a prefix notation wherein every operator follows all of its operands; this notation has the big advantage of being unambiguous and permits to avoid the use of parenthesis. E.g., (3+4)\*5 is equal to 3 4 + 5 \*.

An instance of the PolishCalculator class will understand the following API:

* \_\_init(self)\_\_ with the obvious meaning
* \_\_str(self)\_\_ which will format the expression in the corresponding infix notation
* eval(str) which will evaluate the expression contained in the string str and written in the polish notation

The recognized operators are +, - (both unary and binary), \*, /, \*\* over integers and floats, or, and and not over booleans. At least a space ends each operands and operators, T and F respectively represent True and False.

**Hint.** The evaluation/translation can be realized by pushing the recognized elements on a stack.

**Exercise 4: Playing around with Extensions.**

Python's dictionaries do not preserve the order of inserted data nor store the data sorted by the key. Write an extension for the dict class whose instances will keep the data sorted by their key value. **Note** that the order must be preserved also when new elements are added.