Static Type Practice

1. Complete this table.
Answers to all items are in the Python typing and collections.abc document pages.

(*) In "Example use" column, assume that **x** refers to an object that provides the Type in left column. As an example, for Sized type:

```
# x refers to a Sized object
class MyStuff(Sized):
    def __len__(self):
        return 1
x = MyStuff()
```

Type	Provides methods	Example use (*)
	`__call__( )`	`x = MyCallable()` `x()`
Sized		`len(x)`

	`__iter__( )`	`my_iter = iter(x)`
		`while True:`
		`  print(next(my_iter))`
		`# more typical use`
		`for element in x:`
		`  print(element)`

choose the most basic type that has this behavior		`"apple" in x  # True or False`
		`len(x)`
		`[print(item) for item in x]`

2. Fill in the blanks with correct types. Use the most specific type that applies

```
n:_____ = 100
x:_____ = math.sqrt(2)
# the parameter to average must be a list of float or list of int
# average can return an int or float
Number = ______[    ,    ]
def average( items: _________) -> _______:
    return sum(items)/max(1, len(items))
# be more lenient: allow the parameter to be list, set, tuple, etc.
def average( item: ________________ ) _________________:
# get prices for all size of drinks
def prices( ) -> ___________:
     price_by_size = { "small": 25.0, "medium": 35.0, "large": 45.0 }
     return price_by_size
def get_total(size: ____, qty: ______) -> ________:
    return prices()[size] * qty
```

Add type hints to the code below.

```
class Product:
    """A kind of item that the store sells, e.g Nescafe Ice Coffee."""

    def __init__(self, product_id: ______, description: ______, price: ______):
        self.id: str = product_id
        self.description: str = description
        self.unit_price: float = price

class LineItem:
    """LineItem represents the purchase of a product, with a quantity"""

    def __init__(self, product: _______, quantity: ______):
        self.product = product
        self.quantity = quantity

    def get_total(self) _____________:
        return self.product.unit_price * self.quantity

    def __str__(self)  _____________:
        return self.product.description

class TwoForOneItem(LineItem):
    """A LineItem with buy-1-get-1-free pricing."""

    def __init__(self, product______, quantity______):
        super().__init__(product, quantity)

    def get_total(self) _____________:
        net_qnty = self.quantity - self.quantity//2
        return self.product.unit_price * net_qnty

class TaxCalc:
    # tax rate is a static (class) value - show it as static in UML
    TAX_RATE = 0.07

    @classmethod                          # show class methods as static in UML
    def get_tax(cls, amount: ______) _____________:
        """compute the tax on given amount"""
        return cls.TAX_RATE * amount

class Sale:
    """A sale of a collection of items"""
    def __init__(self):
        self.items: ____________________ = []

    def add_item(self, item: _____________):
        """Add a LineItem to this sale"""
        self.items.append(item)

    def total(self) _____________:
        total_price = sum( item.get_total() for item in self.items )
        tax = TaxCalc.get_tax(total_price)
        return total_price + tax
```