National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2024/2025

Tutorial 5
Data Abstraction

Release date: 16th September 2024
Due: 22nd September 2024, 23:59

General Restrictions

- No importing packages unless explicitly allowed to do so.
- Use only `tuple` as your compound data structure. No `list`, `set`, `dict` etc.

Questions

1. (a) In this question, you will implement a representation of line segments in a 2D plane. Some sample executions are provided for you to test your implementations but you are **strongly encouraged** to create your own test cases.

   i. A point can be represented as a pair of numbers: the x coordinate and the y coordinate.

   Based on this representation of a point

   - Implement a point constructor `make_point` by using a **tuple**.
   - Implement a selector `x_point` which returns the x coordinate of a given point.
   - Implement a selector `y_point` which returns the y coordinate of a given point.

   To test your functions, you may find the following function (that prints a given point argument) useful:

```
def print_point(p):
    print("(", x_point(p), ",", y_point(p), ")")
```

   This is an example of why data abstraction is useful. `print_point` only uses the selectors for your new compound data object, it does not need to know anything about how your object is implemented. Furthermore, there is no need to modify `print_point` when you decide to modify your object's implementation.

   Sample Execution:

```
p1 = make_point(2, 3)
print(x_point(p1)) #expected printout: 2
print(y_point(p1)) #expected printout: 3
print_point(p1) #expected printout: ( 2 , 3 )
```

ii. A line segment has two endpoints. It can be represented by a pair of points: a starting point and an ending point.

Based on this representation of a line segment, implement a line segment constructor `make_segment`.

Implement a selector `start_segment` which returns the starting point of a given line segment.

Implement a selector `end_segment` which returns the ending point of a given line segment.

Sample Execution (continued from part previous):

```
p2 = make_point(5, 7)
s = make_segment(p1, p2)
print_point(start_segment(s)) #expected printout: ( 2 , 3 )
print_point(end_segment(s)) #expected printout: ( 5 , 7 )
```

iii. Finally, using the selectors and constructors which you have implemented, implement a function `midpoint_segment` that takes a line segment as argument and returns its midpoint. (The midpoint of a line segment is the point whose coordinates are the averages of the coordinates of the endpoints of the line segment.)

Sample Execution (continued from previous):

```
m = midpoint_segment(s)
print_point(m) #expected printout: ( 3.5 , 5.0 )
```

(b) Implement a representation for a rectangle in a 2D plane. In terms of your constructors and selectors, create functions that compute the perimeter and the area of a given rectangle.

(Optional) Now implement an alternative representation for rectangles. Can you design your system with suitable abstraction barriers, so that the same perimeter and area functions will work using either representation?

2. In this question, your task is to help a convenience store manage inventory of the goods.

   (a) Your first task is to design a `Product` data type, which serves to model various perishable goods in the convenience store. The `Product` data type supports the following functions:

   - `make_product(name, shelf_life)` takes the name of the product (**a string**), and number of days (**an integer**) that the product can remain on the shelf before expiring. It returns a data type representing a `product`.
   - `get_name(product)` takes in a `product`, and returns its name.
   - `get_shelf_life(product)` takes in a `product`, and returns its shelf life.

   Decide on a data structure **using tuples** to represent a record, and implement the functions `make_product`, `get_name`, and `get_shelf_life`.

   Sample Execution:

```
>>> milk = make_product("milk", 2)
>>> bread = make_product("bread", 3)
>>> get_name(milk)
'milk'
>>> get_shelf_life(milk)
2
>>> get_name(bread)
'bread'
>>> get_shelf_life(bread)
3
```

(b) An `Inventory` is a collection of `Product` objects in the convenience store. The `Inventory` data type supports the following functions:

- `new_inventory()` returns a data type representing an `Inventory` that does not contain any product.
- `add_product(inv, product)` takes in an `Inventory` and a `Product`, and returns a new inventory with a product added to it.
- `add_one_day(inv)` takes in an `Inventory` and returns a new inventory with the same products having sat one more day on the shelf.
- `get_expired(inv)` takes in an `Inventory` and returns a tuple of products that have sat on the shelf longer than their shelf life. Note that the products returned should be **equivalent to the products** added to the inventory. See sample execution for details.

Decide on a data structure **using tuples** to represent an inventory. Provide an implementation for the functions `new_inventory`, `add_product`, `add_one_day` and `get_expired`.

Sample Execution:

```
>>> inv = new_inventory()
>>> inv = add_product(inv, bread)  # add bread
>>> inv = add_one_day(inv)  # day 1
>>> inv = add_product(inv, milk)   # add milk
>>> inv = add_one_day(inv)  # day 2
>>> inv = add_product(inv, milk)  # add another milk
>>> inv = add_one_day(inv)  # day 3
>>> get_expired(inv)
()     # no expired products yet

>>> inv = add_one_day(inv)  # day 4
# bread has expired, first milk has expired
>>> expired = get_expired(inv)
>>> len(expired)
2
>>> bread in expired
True
>>> milk in expired
True

>>> inv = add_one_day(inv)  # day 5
# second milk now expires
>>> len(get_expired(inv))
3   # there are two expired milk products and one bread
```