CSC110 Lecture 14: Variable Reassignment and Object Mutation

Print this handout

Exercise 1: Reassignment and mutation practice

1. a. Consider this code:

```
x = 4
y = 5
x = 2
```

Complete the value-based memory model table to show the values of the variables after this code executes. Show both the old and new values of any variables that are reassigned.

Variable	*Value*
`x`	
`y`	

b. Consider this code:

```
x = 'hi'
y = x + 'bye'
x = y + x
```

Complete the value-based memory model table to show the values of the variables after this code executes. Show both the old and new values of any variables that are reassigned.

Variable	*Value*
`x`	
`y`	

2. Suppose we execute this statement in the Python console.

```
numbers = [1, 0]
```

All of the following statements cause `numbers` to refer to the list value `[1, 0, 8]`. For each one, state whether the statement *mutates the original list* or *reassigns* `numbers` *to a new list object*.

a. `list.append(numbers, 8)`

b. `numbers = numbers + [8]`

c. `list.insort(numbers, 2, 8)`

d. `numbers = [numbers[0], numbers[1], 8]`

3. Suppose we execute the following code:

```
lst1 = [1, 0, 8]
lst2 = list.sort(lst1)
```

After the code above is executed, which of the following expressions evaluate to `True`? Circle those expression(s).

`lst1 == [1, 0, 8]`	`lst1 == [0, 1, 8]`
`lst2 == [1, 0, 8]`	`lst2 == [0, 1, 8]`

4. Circle the `set` operations that **mutate** the input set. Try calling `help` on each function, and/or looking them up in A.2 Python Built-In Data Types Reference.

`set.intersection`	`set.remove`
`set.add`	`set.union`

5. Suppose we execute the following code:

```
animals = {'fish': {'swim'},
           'kangaroo': {'hop'},
           'frog': {'swim', 'hop'}}
```

Indicate whether each statement will cause an error and, if not, whether the statement will increase the number of key/value pairs in the dictionary:

Statement	*Error?* (Y/N)	**Increases `len(animals)`?** (Y/N)
`animals['human'] = {'swim', 'run', 'walk'}`		
`set.add(animals['monkey'], 'swing')`		
`set.add(animals['kangaroo'], 'airplane')`		
`animals['frog'] = {'tapdance'}`		
`animals['dolphin'] = animals['fish']`		

6. Read the following function's header and description, and then complete its doctests and implement the function body.

```
def move_item(items: list, other_items: set) -> None:
    """Remove the first item from items and add it to other_items.

    Preconditions:
        - items != []


    >>> numbers_list = [1, 2, 3]
    >>> numbers_set = {10, 20}
    >>> move_item(numbers_list, numbers_set)
    >>> numbers_list



    >>> numbers_set ==
    True
    """
```

Exercise 2: Loops with collection accumulators

Recall our marriage license dataset, where we represent each row of data using the following data class:

```
from dataclasses import dataclass
import datetime


@dataclass
class MarriageData:
    '''A record of the number of marriage licenses issued in a civic centre
    in a given month.

    Instance Attributes:
      - id: a unique identifier for the record
      - civic_centre: the name of the civic centre
      - num_licenses: the number of licenses issued
      - month: the month these licenses were issued

    Representation Invariant omitted.
    '''
    id: int
    civic_centre: str
    num_licenses: int
    month: datetime.date
```

Implement each of the following functions using a loop with an accumulator of the appropriate collection data type. Use mutating operations to avoid creating multiple collection objects.

```
def filter_by_name(data: list[MarriageData], civic_centre: str) -> list[MarriageData]:
    """Return all rows in data whose civic_centre is civic_centre.

    Equivalent to:
      [row for row in data if row.civic_centre == civic_centre]
    """
```

```
def num_issued_by(data: list[MarriageData], civic_centre: str) -> set[int]:
    """Return the unique numbers of marriage licenses issued per month at the given
    civic_centre.

    Equivalent to:
      {row.num_licenses for row in data if row.civic_centre == civic_centre}
    """
```

```
def marriages_by_centre(data: list[MarriageData],
                        month: datetime.date) -> dict[str, int]:
    """Return a mapping from civic centre name to the number of marriage licenses
    issued by that centre in the given month.

    Preconditions:
        - Each civic centre has only one row of MarriageData for the given month.

    Equivalent to:
      {row.civic_centre: row.num_licenses for row in data if row.month == month}
    """
```