Lab 13

We encourage you to work together on the Pre Lab. The Pre Lab is not graded but will help you prepare for your lab session. If you have any questions on the material of the Pre lab, first check the book and recitation slides. If you continue to have any doubt about it, please email your recitation TA or the course instructor.

This Prelab gives some examples about the use of tuples, sets and dictionaries.

Here is the PreLab13

Environment Setup

Go to your working directory in “data.cs.purdue.edu” and create a directory “cs177/lab13”. Refer to the first lab material if you need to remember the steps to do so. Then start the IDLE Python Interpreter.

About this lab

So far you have learned many of the features of Python and have acquired a very good level of programming logic. You have been able to solve projects that involve data collections. Today's Lab objective is to improve your proficiency dealing with the different types of data collection featured by Python. Specifically, you will work with lists, tuples, sets and dictionaries.

Please find the skeleton of the lab here. We strongly encourage you to follow the skeleton without modifying the comments. You are expected to fill the blanks between comments and follow the logic provided.

Deliverable: Change the name of the skeleton from lab13_skeleton.py to lab13.py and deliver the latter.

Exercise 1: Tuples vs Lists

This exercise is very short and part of the code is provided to you. The idea is to highlight the difference between mutable and immutable data collections since you will use this knowledge to complete the other exercises. As you already know lists are mutable while tuples are not. This affect the way we manipulate them, specially when we want to add new elements to the data collection. In this exercise you will have the opportunity to explore the difference.

Your task here is to create two functions exercise1() and doOperations(). The function exercise1() calls doOperations(). The skeleton of the these two functions is provided below.
#Receive two values x and y and apply a list of operations
#Store same result in two different data collections (notice the difference)
#@param x: type integer
#@param y: type integer
#@return tuple that contain a list of the results and a tuple of the same results: type tuple
def doOperations(x, y):

    #Create empty list
    myList =

    #Create empty tuple
    myTuple =

    #Calculate addition x + y and store value in both data collections
    myList.append(x+y)
    myTuple = myTuple + (x+y,)

    #Calculate substraction x - y and store value in both data collections
    myList.append(x-y)
    myTuple =

    #Calculate multiplication x * y and store value in both data collections
    #Calculate integer division x // y and store value in both data collections

    #Return tuple of both collections
    return (myList, ...)

#Define two values, apply arithmetic operations to the them and print results stored in both a list and a tuple
#@param None
#@return None
def exercise1():

    #Define variables
    x = 10
    y = 5

    #Call doOperations with x and y and store the returned tuple
    results =

    #Print results in list
    print('List : ' + str(results[0]))

    #Print results in tuple
    print('Tuple : ' ...)
**Code of exercises 2 and 3**

```python
def main():
    print('Exercise 1: ')
    exercise1()

    print('Exercise 2: ')
    exercise2()

    print('Exercise 3: ')
    exercise3()

main()
```

Your specific tasks are:

**In exercise1()**

1. Define the two variables `x = 10` and `y = 5`
2. Call function `doOperations()` and receive the returned value in the `results` variable
3. Print results according to the sample output

**In `doOperations()`**

1. Create an empty list
2. Create an empty tuple
3. Apply the four arithmetic operations (+-*/), to `x` and `y` as indicated
4. Store the result of each operation twice: one in the list and the other in the tuple
5. Understand the reason of the difference
6. Return both the list and the tuple as a tuple (list, tuple)

In order to complete this exercise you only need to follow the provided pattern. As you can see you treat tuples in the exact way as you treat strings. Every time you want to add a new element you practically concatenate the existing tuple with the new element and assign the resulting new tuple to the same variable. This is so because both tuples and strings are immutable.

On the other hand, since lists are mutable you can modify the same data collection (the list) using one the available methods (in this case `append()`).

**Sample output:** Function `exercise1()` called from `main()` as indicated in the skeleton

| Exercise 1: | List : [15, 5, 50, 2] | Tuple : (15, 5, 50, 2) |
Exercise 2: Sets

In this exercise you are also required to code two functions exercise2() and applyOperations(). As in exercise 1, function exercise2() calls applyOperations. The basic idea of the entire exercise is to create two sets A and B and apply some set operations available in Python. The two sets you will work will contain the following elements:

1. \[ A = \{\text{'hello'}, 1, (3,4), 56, 78, 9, 22, 45, \text{'Purdue'}, (5,6,7,8)\} \]
2. \[ B = \{\text{'hello'}, 1, (3,4), 56, 78, 44, 22, 45\} \]

Notice that all the elements of the sets are immutable. Sets cannot contain mutable elements such as lists. Once the sets have been created you are required to apply the following operations:

1. issubset: \[ A.\text{issubset}(B) \]
2. issuperset: \[ A.\text{issuperset}(B) \]
3. intersection: \[ A.\text{intersection}(B) \]
4. union: \[ A.\text{union}(B) \]
5. difference: \[ A.\text{difference}(B) \]
6. symmetric difference: \[ A.\text{symmetric_difference}(B) \]

---

Code of exercise 1
---

```python
#Receive two sets A and B and apply a list of operations using A as reference
#@param A: type set
#@param B: type set
#@return tuple of the different results (issubset, issuperset, intersection, union, difference, symmetric difference, update): type tuple
def applyOperations(A,B):
    #Create an empty tuple called results (remember how initialize an empty tuple from exercise 1)
    results =

    #Operation: Is A subset of B? (remember how to update tuples from exercise 1)
    results = results + (A.issubset(B),)

    #Operation: Is A superset of B?
    results =

    #Operation: A intersection B
    results =

    #Operation: A union B
    results =
```

---

http://courses.cs.purdue.edu/
```python
results =

#Operation: A difference B
results =

#Operation: A symmetric difference B
results =

return results

#Define sets to be used in the exercise, apply set operations to the sets and print results
#@param None
#@return None
def exercise2():

    #Create sets A and B
    A =
    B =

    #Apply operations to A and B (call the applyOperation function and assign the result to a variable)
    results =

    #Print results according to the output sample
    print('Is A subset of B? ' + str(results[0]))
    print(...
    print(...
    print(...
    print(...
    print(...

- -

Code of exercise 3
- -

def main():

    print('Exercise 1: ')
exercise1()

    print('\nExercise 2: ')
exercise2()

    print('\nExercise 3: ')
exercise3()

main()
```

Your specific tasks are:
In exercise2()

1. Create the two specified sets
2. Call applyOperations() function with both sets and receive the returned value in the results variable
3. Print results according to the sample output

In applyOperations()

1. Create an empty tuple called results to store the results of all the operations
2. Apply the set operations over the two sets as indicated
3. Store the result of each operation in the tuple called results (remember procedure to add elements to a tuple from exercise 1)
4. Return the tuple results containing the result of each operation

**Sample output:** Function exercise2() called from main() as indicated in the skeleton

<table>
<thead>
<tr>
<th>Exercise 2:</th>
<th>Is A subset of B? <strong>False</strong></th>
<th>Is A superset of B? <strong>False</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A intersection B:</td>
<td>{1, 45, 78, 22, 56, 'hello', (3, 4)}</td>
<td></td>
</tr>
<tr>
<td>A union B:</td>
<td>{1, 9, 44, 45, 78, 'Purdue', (5, 6, 7, 8), 22, 56, (3, 4), 'hello'}</td>
<td></td>
</tr>
<tr>
<td>A difference B:</td>
<td>{'Purdue', 9, (5, 6, 7, 8)}</td>
<td></td>
</tr>
<tr>
<td>A symmetric difference B:</td>
<td>{9, 44, 'Purdue', (5, 6, 7, 8)}</td>
<td></td>
</tr>
</tbody>
</table>

**Exercise 3: Dictionaries**

In this exercise your task consists of coding two functions exercise3() and modifyDate(). Once again exercise3() calls modifyDate(). These functions change the format of the provided list of dates. The dates provided are in the format 12/25/2012 (mm/dd/yyyy). Your functions must change this format to:

1. Format 1: December/25/2012
2. Format 2: Dec/25/2012

For this task is imperative to use dictionaries provided in the skeleton below. As you can appreciate, with an educational purpose we are using two different ways to define the dictionaries

```python
#Receive a date of the form 12/25/2012 and change it to December/25/2012
#@param date: type string
#@return tuple of new formats (format1,format2): type tuple
def modifyDate(date):
    #First way to declare a dictionary
    monthDictNum = {1:'January', 2:'February', 3:'March', 4:'April', \
```

http://courses.cs.purdue.edu/
5: 'May', 6: 'June', 7: 'July', 8: 'August', \

```python
# Second way to declare a dictionary
monthDictAbbr = dict([("January", 'Jan'), ('February', 'Feb'),
                     ('March', 'Mar'), ('April', 'Apr'),
                     ('May', 'May'), ('June', 'June'), ('July', 'July'),
                     ('August', 'Aug'),
                     ('September', 'Sept'), ('October', 'Oct'), ('November', 'Nov'),
                     ('December', 'Dec')])

# TODO

# Return tuple
return (format1, format2)
```

```python
# Define list of dates to be used in the exercise, change their format and print results
@param None
@return None
def exercise3():
    # List of dates
    
    # Iterate over date and print it along with their equivalent formats
    for date in ...
        # Call the corresponding function and store the result in newFormats
        newFormats =

        # Print date and its corresponding equivalences according the sample output
        print(date + ...)
```

```python
def main():
    print('Exercise 1: ')
exercise1()
    
    print('Exercise 2: ')
exercise2()
    
    print('Exercise 3: ')
exercise3()
main()
```
Your specific tasks are:

In exercise3():

1. Iterate of each date of the list
2. For each date call the function modifyDate() and store the result in the variable newFormats
3. Print the provided date along with the new two formats (contained in the variable newFormats) as specified in the sample output below.

In modifyDate():

1. Split the string date using character '/' and save the resulting list with the name items.
2. Use only the first element of the resulting list (item[0]) as key of the dictionary monthDictNum.
3. Remember change the type of items[0] to be an integer (since the key of mothDictNum is of type integer)
4. Retrieve the corresponding value v1 from the dictionary and use it for two purposes.
5. Purpose 1: Form the format1 string by concatenating v1 + items[1] + items[2]
6. Purpose 2: Use v1 as key of the dictionary mothDictAbbr and obtain the corresponding value v2
7. Form the forma2 string by concatenating v2 + items[1] + items[2]
8. Return tuple(format1, format2)

Sample output: Function exercise3() called from main() as indicated in the skeleton

<table>
<thead>
<tr>
<th>Exercise 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/7/1986 = December/7/1986</td>
</tr>
<tr>
<td>3/19/1982 = March/19/1982</td>
</tr>
<tr>
<td>6/26/1972 = June/26/1972</td>
</tr>
<tr>
<td>4/29/1957 = April/29/1957</td>
</tr>
<tr>
<td>5/18/1954 = May/18/1954</td>
</tr>
</tbody>
</table>

Turnin Instructions

Run putty and login to data.cs.purdue.edu. Turn in your lab by typing:

$ cd cs177
$ turnin –c cs177=COMMON –p lab13 lab13

To verify that you did not submit a wrong file or an empty one, run the following command:

$ turnin –v -p lab13

Grading Criteria

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise 1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Exercise 2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Exercise 3</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>