Week 06 Tuple, List, Iterables

Part 1 Tuple

In this exercise, please try to come out with the answer without using IDLE/Python first. Then type in the expressions into IDLE to verify your answers. The objective is for you to understand why and how they work. If there is an error, specify the error and the cause of error.

Expressions	**Output**

`tup_a = (10, 11, 12, 13)`
`print(tup_a)`
`tup_b = (“CS”, 1010)`
`print(tup_b)`
`tup_c = tup_a + tup_b`
`print(tup_c)`
`print(len(tup_c))`	

`print(11 in tup_a)`
`print(14 in tup_b)`
`print(“C” in tup_c)`
`print(tup_b[1])`
`tup_d = tup_b[0]*4`
`print(tup_d)`
`print(tup_b[1] * 4)`	

`tup_e = tup_d[1:]`
`print(tup_e)`
`tup_f = tup_d[::-1]`
`print(tup_f)`
`tup_g = tup_d[1:-1:2]`
`print(tup_g)`
`tup_h = tup_d[-1:6:-2]`
`print(tup_h)`	

`tup_i = (1)`
`print(tup_i)`
`tup_j = (1,)`
`print(tup_j)`
`print(tup_i * 4)`
`print(tup_j * 4)`	

`print(min(tup_a))`
`print(max(tup_a))`
`print(min(tup_c))`
`print(max(tup_c))`
`print(min(tup_e))`
`print(max(tup_e))`	

`for i in tup_b:` `  print(i)`	
`for i in range(5):` `  print(i)`	
`for i in range(2,5):` `  print(i)`	
`for i in range(2,5,2):` `  print(i)`	
`for i in range(5,1,-1):` `  print(i)`	
`for i in range(5,6,-1):` `  print(i)`	

Part 2 List

In this exercise, please try to come out with the answer without using IDLE/Python first. Then type in the expressions into IDLE to verify your answers. The objective is for you to understand why and how they work. If there is an error, specify the error and the cause of error.

Expressions	**Output**

`lst_a = [“CS”, 1010]`
`print(lst_a)`
`lst_b = [“E”,(“is”, “easy”)]`
`print(lst_b)`
`lst_c = lst_a + lst_b`
`print(lst_c)`	

`tup_a = (“CS”, 1010)`
`tup_a[1] = 2030`
`lst_a[1] = 2030`
`print(lst_a)`	

`lst_a.append(“E”)`
`print(lst_a)`
`lst_a.extend(“easy”)`
`print(lst_a)`	

`cpy_b = lst_b[:]`
`print(cpy_b)`
`cpy_b[1] = “is hard”`
`print(cpy_b)`
`print(lst_b)`	

`lst_d = [1, [2], 3]`
`cpy_d = lst_d[:]`
`print(cpy_d)`
`print(lst_d)`
`lst_d[1][0] = 9`
`print(cpy_d)`
`print(lst_d)`	

`print(lst_d == cpy_d)`
`print(lst_d is cpy_d)`
`print(lst_d[1] == cpy_d[1])`
`print(lst_d[1] is cpy_d[1])`	

Part 3 List Mutation

Try the follow and notice the difference between the left and right column in each row.

`x = 1`
`y = x`
`x = 2`
`print(x)`
`print(y)`
	`lstx = [1,2,3]`
	`lsty = lstx`
	`lstx[0] = 999`
	`print(lstx)`
	`print(lsty)`

`a = 4`
 	
`def foo(x):`
`  x = x * 2`
`  print(x)`
 	
`print(a)`
`foo(a)`
`print(a)`
	`lsta = [1,2,3]`
	
	`def foo2(lst):`
	`  lst[0] = lst[0]*2`
	`  lst[1] = lst[1]*2`
	`  print(lst)`
	
	`print(lsta)`
	`foo2(lsta)`
	`print(lsta)`

Part 4 Burger Meal Order

We can use tuple/list to create a data that consists of mixed types. In this exercise, you are hired by a fast food franchise to implement their automatic ordering system. One order can contain a lot of burgers. For example, my order for my family can be:

```
>>> myOrder
('BVPB', 'BCPCPB', 'BPCBPCB')
```

So an "order" is a tuple of burger collections. You can download the file mealOrder.py to have a look at the implemented functions:

```
def makeEmptyOrder():
    return ()

def add_burgerToOrder(order,burger):
    return order + (burger,)
```

And we can create an order like the followings

```
>>> myOrder = makeEmptyOrder()
>>> myOrder = add_burgerToOrder(myOrder,'BVPB')
>>> myOrder = add_burgerToOrder(myOrder,'BCPCPB')
>>> myOrder
('BVPB', 'BCPCPB')
```

And noticed that the function `burgerPrice()` from previous tutorials are also provided in your file.

Your tasks are

- Write a function `enoughMoney(order, moneyInMyPocket)` to check if you have enough money to pay for your order, namely, returning `True` if `moneyInMyPocket` is more than or equal to the total price in the order, and `False` otherwise.

  ```
  >>> enoughMoney(myOrder,15)
  True
  >>> enoughMoney(myOrder,13)
  False
  ```

- Write a function `printReceipt(order)` to print a nice receipt for your order.

  ```
  >>> printReceipt(myOrder)
  Your orders:
  BVPB $3.2
  BCPCPB $5.6
  BPCBPCB $6.1
  --------------
  Total: 14.9
  ```

- Write a function `removeItem(order,item)` to remove an item in the order.

```
>>> myOrder = removeOrder(myOrder,'BVPB')
>>> myOrder
('BCPCPB', 'BPCBPCB')
>>> myOrder = removeOrder(myOrder,'BVVVVVPPB')
The item BVVVVVPPB is not in the order.
```

Further Challenge

The current implementation assumes the input and output are tuples. How different is it if we can use **lists** instead of tuples?

- Try to implement the whole system with list.
- Is it possible to allow the inputs to be either lists or tuples?