Discussion 12: Dictionaries & Recursion in Python SOLUTIONS

Planning Your Phase 1

1. In the table below, write Python code to execute the listed commands on `class_dict`.

`class_dict = {'Math':'1A', 'English':'R1A'}`

Add the key 'CS' with the value '10'	`class_dict['CS'] = '10'`
Access the value of 'Math'	`class_dict['Math']`
Change the value of 'Math' to '1B'	`class_dict['Math'] = '1B'`
Check if 'UGBA' is a key in class_dict	`'UGBA' in class_dict`
Check if '10' is a value in class_dict	`'10' in class_dict.values()`
Get a list of the keys in class_dict	`list(class_dict)`

****Note: by default, when you try to perform an operation on a dictionary (e.g., iterating through it, checking whether it contains a key, converting it to a list), Python will look *only* at the keys. So `'UGBA' in class_dict` is actually checking whether 'UGBA' is present in the *keys* of the `class_dict`, and `list(class_dict)` generates a list of *only keys*. We could also write `'UGBA' in class_dict.keys()` and `list(class_dict.keys())`, but this is redundant and unnecessary.**

2. Can you access a key, value pair in a dictionary by its index?

No, the pairs in a dictionary are unordered and thus do not have indices.

3. Are keys and/or values in a dictionary returned in a predictable order?

No, the items in a dictionary are unordered. See CS61B for the reason why.

4. Can dictionaries have duplicate keys? What about duplicate values?

A dictionary may not contain duplicate keys. If you try to insert a duplicate key, the value assigned to the corresponding original key will simply be overwritten.

Duplicate values are fine.

Dictionary Practice

```
fav_numbers = {'Schuyler': 18, 'Brendan': 12, 'Mansi': 7, 'Aaron': 152}
```

1. On the lines below, write Python code that increments each person's favorite number by the length of their name.

```
for person in fav_numbers:
  fav_numbers[person] += len(person)
```

2. On the lines below, write Python code that prints a list of all people from `fav_numbers` whose favorite numbers are even.

```
names = []
for person in fav_numbers:
     if fav_numbers[person] % 2 == 0:
          names.append(person)
print(names)
```

3. Write a function `merge_dicts` that takes two dictionaries as input, and returns a new dictionary that contains all entries from both input dictionaries. Your function should not modify

the inputs. You can assume that both input dictionaries have strings as keys and numbers as values. For any keys present in both input dictionaries, the corresponding value in the output dictionary should be the sum of the values in the inputs.

```
>>> dict1 = {‘Schuyler’: 10, ‘Brendan’: 15}
>>> dict2 = {‘Aaron’: 5, ‘Mansi’: 20, ‘Schuyler’: -10}
>>> merge_dicts(dict1, dict2)
{‘Schuyler’: 0, ‘Aaron’: 5, ‘Mansi’: 20, ‘Brendan’: 15}
```

```
def merge_dicts(d1, d2):
    new_dict = {}
    for key in d1:
        new_dict[key] = d1[key]
    for key in d2:
            if key in new_dict:
                new_dict[key] += d2[key]
            else:
                new_dict[key] = d2[key]
    return new_dict
```

4. Assume we have defined `food_dict` in the Python interpreter, as below. What will be displayed after each of the following lines executes? If the result is an error message, just write "Error." **Assume that the commands are executed independently, NOT sequentially.**

```
>>> food_dict = {"fruit": "apple", "veggie": "carrot", "beverage": "water",
"grain": "rice"}
```

```
>>> len(food_dict)
```

```
4
```

```
>>> list(food_dict)
```

```
['beverage', 'fruit', 'grain', 'veggie']
```

```
Note: You are not expected to write these items in any particular order.
```

```
>>> food_dict[0]
```

Error

```
>>> (‘fruit’ in food_dict) and (‘apple’ in food_dict)
```

False

```
>>> ("fruit" in food_dict.keys()) and ("apple" in food_dict.values())
```

True

```
>>> for food in food_dict:
...     food += "s"
>>> food_dict
```

{'beverage': 'water', 'fruit': 'apple', 'grain': 'rice', 'veggie': 'carrot'}

Notes: Again, order doesn’t matter. The keys in the dictionary don’t change because we are only adding ‘s’ to the food variables, not the actual keys in the dictionary. Remember that strings are immutable, as in Snap!.

```
>>> def recursion_is_fun(dict1, dict2): ...
if dict2 == {}:
...         return dict1
...     dict2.pop(list(dict2)[0])
...     return recursion_is_fun(dict1, dict2)
>>> copy = food_dict
>>> recursion_is_fun(food_dict, copy)
```

{}

Notes: Dictionaries are mutable objects in Python, and thus behave exactly like lists. To make a copy of a dictionary in Python, you must call the DICT_NAME.copy() method.

```
>>> more_food = {"protein" : "chicken"}
>>> food_dict["more food"] = more_food
```

```
>>> food_dict

{'beverage': 'water', 'more food': {'protein': 'chicken'}, 'fruit': 'apple',
'grain': 'rice', 'veggie': 'carrot'}
```

Note: Again, order doesn't matter.

Recursion in Python

1. In the table below, translate the Snap! code into Python. These Python snippets will be useful for writing recursive functions.

	`my_list[1:]`
 Note: This block doesn't actually exist in Snap!	`my_list[:-1]`
	`my_list[1:-1]`
	`[5] + my_list`
	`my_list + your_list`
	`my_list.append(5)`
	`my_list.pop()`

Note: There are multiple ways to perform most of the operations above. The solutions given are simply our preferences, not the only "correct" expressions.

2. Is it possible to apply the 'all but first of' and 'all but last of' python expressions above to a string instead of a list? If so, how would we need to modify these expressions?

Yes, and you don't need to change the expressions at all.

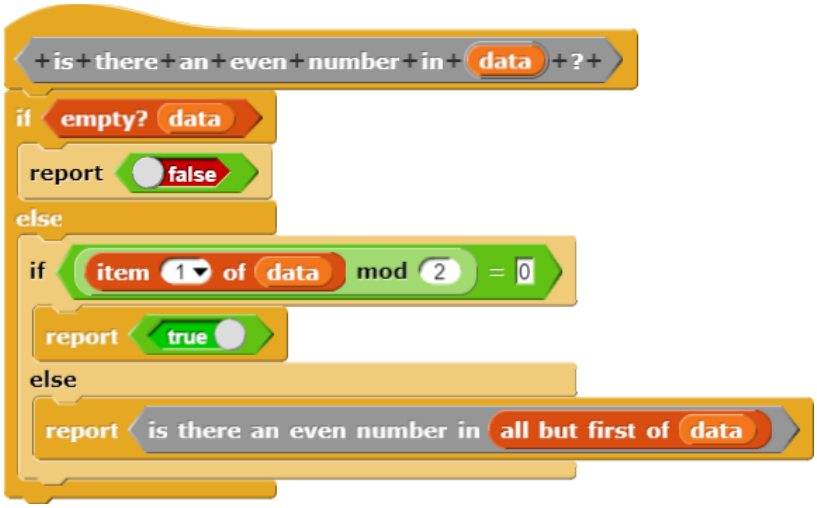
3. Is it possible to apply the 'all but first of' and 'all but last of' python expressions above to a dictionary instead of a list? If so, how would we need to modify these expressions?

No. Items in dictionaries are unordered and thus the notion of "first" and "last" items doesn't make sense.

4. Which of the above python expressions, if any, actually modify the original list?

Only append and pop. All other expressions return a new list without modifying the original.

5. Translate the following code from Snap! into Python:




```
def is_even_in(data):
      if len(data) == 0:
            return False
      elif data[0] % 2 == 0:
            return True
      else:
            return is_even_in(data[1:])
```

6. Write a recursive function in Python that removes a given item from a list, as illustrated by the examples below. It should create and return a new list, not mutate the input list.

```
>>> delete_elements(4, [4, 5, 6])
[5, 6]
```

```
>>> delete_elements(7, [4, 5, 6])
[4, 5, 6]
>>> delete_elements(4, [4, 5, 6, 4])
[5, 6]
```

```
def delete_elements(item, lst):
    if len(lst) == 0:
        return lst
    elif lst[0] == x:
        return delete_elements(x, lst[1:])
    else:
        return [lst[0]] + delete_elements(x, lst[1:])
```