COMP10001 Foundations of Computing
Semester 2, 2016

Tutorial Questions: Week 5

1. Given the assignment `d = {"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`, evaluate the following expressions, and determine: (a) the value the expression evaluates to; and (b) the final value of `d`. Assume that `d` is reset to its original value for each sub-question:

   (a) `d["R"]`

   A: *(a)* `0`*, (b)* `{"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (b) `d.pop("R")`

   A: *(a)* `0`*, (b)* `{"G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (c) `d["R"] = 255`

   A: *(a)* `None`*, (b)* `{"R": 255, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (d) `d["H"]`

   A: *(a)* `None`*, with* `KeyError`*, (b)* `{"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (e) `d.keys()`

   A: *(a)* `['B', 'R', 'other', 'G']`*,*
   (b) `{"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (f) `d["other"]["blur"] = 0.1`

   A: *(a)* `None`*, (b)* `{'B': 0, 'R': 0, 'other': {'opacity': 0.6, 'blur': 0.1}, 'G': 255}`

   (g) `d[["H","S","L"]] = [120,98,5]`

   A: *(a)* `None`*, with* `TypeError`*, (b)* `{"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

   (h) `d["R","B","G"]`

   A: *(a)* `None`*, with* `KeyError`*, (b)* `{"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`

2. Write a program that prints the keys of a dictionary in descending order of their values. For example, for a dictionary `fruit_prices = {"apple": 0.5, "banana": 19, "durian": 7}`, your program should print:

```
banana
durian
apple
```

   A:

```python
fruit_prices = {"apple": 0.5, "banana": 19, "durian": 7}
items = []
for key, value in fruit_prices.items():
    items.append((value, key))
for value, key in sorted(items, reverse=True):
    print(key)
```

3. Both lists and dictionaries have a `pop` method, with the important distinction that it can be called without any argument for lists, but can for dictionaries. What does `pop` do in each case, and what is the reason for this difference between the two types?

   A: *With lists,* `pop()` *removes + returns the last value in the list. With dictionaries, there is no meaningful ordering to the keys, so it doesn't make sense to talk about the "last value"; instead,* `pop(KEY)` *removes + returns the value associated with* `KEY`*.*

4. What is the output of the following code:

```python
def foo(x, y):
    print(x**y)

exp = foo(2,2)
print(exp)
```

A:

```
4
None
```

The issue is that `foo` prints rather than returns the value of the calculation, and doesn't have a return value, meaning that it defaults to returning `None`; hence, the assignment to the return value results in a value of `None`, rather than `4`

5. Write a function `largest_item(lst)` that takes a single list `lst` as an argument and returns the largest item in the list.

   A:

   ```
   def largest_item(lst):
       if not lst:
           return
       largest = lst[0]
       for item in lst[1:]:
           if item > largest:
               largest = item
       return(largest)
   ```

6. What is the output of the following code:

   ```
   def mutate(x, y):
       x = x + "--The End--"
       y.append("The End")
       print(x)
       print(y)

   mystr = "It was a dark and stormy night."
   mylist = mystr.split()
   mylist2 = mylist
   mutate(mystr, mylist2)
   print(mystr)
   print(mylist)
   ```

   A:

   ```
   It was a dark and stormy night.--The End--
   ['It', 'was', 'a', 'dark', 'and', 'stormy', 'night.', 'The End']
   It was a dark and stormy night.
   ['It', 'was', 'a', 'dark', 'and', 'stormy', 'night.', 'The End']
   ```

7. Write a function `letter_overlap(s1, s2)` that takes two string arguments (`s1` and `s2`), and returns the number of unique letters that are present in both strings.

   A:

   ```
   def letter_overlap(s1,s2):
       letter_dict = {}
       for char in s1:
           letter_dict[char] = True
       overlap = 0
       for char in s2:
           if char in letter_dict:
               overlap += 1
               letter_dict.pop(char)
       return(overlap)
   ```

Optional Extension Questions for Self-study

1. Write a function `common_letter(string)` that takes a single argument `string` (a string) and returns the most common letter(s) in `string` (as a sorted list of strings), and how many times they occur.

A:

```python
def common_letter(string):
    letter_dict = {}
    for letter in string:
        if letter not in letter_dict:
            letter_dict[letter] = 0
        letter_dict[letter] += 1
    freq_letter = []
    freq_letter_count = 0
    for letter in letter_dict.keys():
        if letter_dict[letter] > freq_letter_count:
            freq_letter = [letter]
            freq_letter_count = letter_dict[letter]
        elif letter_dict[letter] == freq_letter_count:
            freq_letter.append(letter)
    return(sorted(freq_letter),freq_letter_count)
```