Dictionaries

Introduction to Computer Programming

Dr. Paul Vrbik

October 30, 2018

Here is a deliberately vague question.

Question

Write a function that returns the character frequency of a string, ignoring case.

For example, the character frequency of “Hello World” would encode that there is one “h”, three “ℓ”’s and so on.

What are some viable representations?

Two Lists

```
[ [‘h’, ‘e’, ‘l’, ‘o’, ‘ ’, ‘w’, ‘r’, ‘d’],
 [1, 1, 3, 2, 1, 1, 1, 1] ]
```

One-to-one Encoding

```
 a b c d e f g h i j k l m n o p q r s t u v w x y z
[0,0,0,1,1,0,0,1,0,0,0,3,0,0,2,0,0,1,0,0,0,0,1,0,0,0]
```

Hash Function

In both cases we provided a way to map a key (character) to a value (number).

$$\texttt{str} \to \texttt{int}$$

$$\texttt{h} \mapsto 1$$

$$\texttt{e} \mapsto 1$$

$$\texttt{k} \mapsto 3$$

$$\vdots$$

This mapping is called a hash function.

Question

What are the hash functions for the two list and one-to-one encoding?

```
def hash_1(key:str, L1:List[str], L2:List[str]) -> int:
    ‘‘‘Two list encoding.’’’
    return L2[L1.index(key)]


def hash_2(key:str, xs:List[int]) -> int:
    ‘‘‘One-to-one encoding.’’’
    pos_in_alpha = chr(ord(key)-ord(‘a’))
    return xs[pos_in_alpha]
```

Generally speaking we can index a list by any type of key given some hash-function which takes keys to values.

Python has a “magic” hash function that will index anything appropriately. The result is implemented as the `dictionary` data-type.

Dictionaries have no ordering because the keys can be of mixed type.

Example

For example, the “Hello World” example encoded using a `dictionary` is

```
>>> xs = {
    ‘d’:1,
    ‘o’:1,
    ‘ ’:1,
    ‘r’:1,
    ‘w’:1,
    ‘e’:1,
    ‘l’:3,
    ‘h’:0
    }
```

Question

Refactor the character frequency code to use dictionaries and use the following header.

```
def char_freq( xs:str ) -> Dict[str, int]:
```

Dictionaries

Definition (Dictionary)

A dictionary in Python is a data-structure that stores `tuple(key, value)` pairings. Python has a “magic” hash function to find a key among the tuples fast.

Dictionaries are not ordered and are mutable.

Hash tables are a candidate for the most important/useful data-structure.

items

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }

>>> H.items()
dict_items([(‘red’, [‘apple’, ‘firetrucks’, ‘cars’]),
(‘yellow’, [‘banana’, ‘cars’]), (‘blue’, [‘sky’, ‘cars’])])
```

values

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }

>>> H.values()
dict_values([[‘apple’, ‘firetrucks’, ‘cars’], [‘banana’, ‘cars’],
[‘sky’, ‘cars’]])
```

keys

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }
>>> H.keys()
dict_keys([‘red’, ‘yellow’, ‘blue’])
>>> H["blue"]
["sky", "cars"]
>>> H["green"]
KeyError: ‘green’
>>> H["green"] = ["leaves"]
# Is fine because we’re assigning not retrieving.
```

clear & copy

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }
>>> F = H.copy()
>>> H.clear()
>>> H["blue"]
KeyError: ‘blue’
>>> F["blue"]
[‘sky’, ‘cars’, ‘windex’]
```

copy

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }
>>> F = H.copy()
>>> H["blue"].append("windex")
>>> H["blue"]
[‘sky’, ‘cars’, ‘windex’]
>>> F["blue"]
[‘sky’, ‘cars’, ‘windex’]
```

shallow copy

get

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }
>>> H.get("red")
["apple", "firetrucks", "cars"]
>>> H["green"]
KeyError: ‘green’
>>> H.get("green")
None
```

safer because no error is thrown here

setdefault

```
>>> dict.setdefault("green", [])

>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }

>>> H["green"]
[]
```

pop & popitem

```
>>> H = {
    "red":["apple", "firetrucks", "cars"],
    "yellow":["banana", "cars"],
    "blue":["sky", "cars"]
    }
>>> H.popitem()
("blue", ["sky", "cars"])
>>> H
{‘red’: [‘apple’, ‘firetrucks’, ‘cars’],
‘yellow’: [‘banana’, ‘cars’]}
```

Question

Write a function that when given a string, returns a dictionary whose keys are characters and values are the positions of these characters in the string.

```
def char_index( xs:str ) -> Dict[str, List[int]]:
    """
    >>> char_index("")
    {}
    >>> char_index("aaAAbbBB")
    {‘a’:[0, 1], ‘b’[4, 5]:, ‘A’:[2, 3], ‘B’:[6, 7]}
    """
```

Question

Complete the following function.

```
def combine( d1:Dict[int, List[int]],
             d2:Dict[int, List[int]] ) -> Dict[int, int]:
    '''Return the dictionary where each key is a key
    that is in both d1 and d2.

    The value associated with each key in the new
    dictionary is the sum of all the integers associated
    with that key in d1 and d2.

    >>> combine({1:[2], 4:[5, 6]}, {4:[8]})
    {4:19}
    '''
```

Question

Write a function with the following contract that returns a dictionary mapping artists in the file to the number of albums they have authored.

```
def count_albums(albums:TextIO) -> Dict[str, int]:
```

Question

Write a function with the following contract that does a reverse lookup which returns all keys corresponding to an item.

```
def reverse_lookup(d:Dict, item) -> list:
    ‘‘‘Returns all keys such that d[keys] == item
    >>> reverse_lookup({1:‘paul’, 19:‘vrbik’, -31:‘paul’}, ‘paul’)
    [1, -31]
    reverse_loopup({1:‘paul’, 19:‘vrbik’, -31:‘paul’}, ‘irene’)
    []
    ’’’
```

Question

Write a function which inverts the keys and items of a dictionary.

```
def invert(d:Dict) -> Dict:
    ‘‘‘Return the inverted version of d.
    >>> returned = invert({1: 10, 2: 10})
    >>> returned == {10: [1, 2]} or returned == {10: [2, 1]}
    True
    ’’’
```

Next Time

1. STUDY.
2. Whatever we did not finish today.
3. More dictionaries.