DICTIONARIES AND STRUCTURING DATA

CS 3080: Python Programming

University of Colorado
Colorado Springs

Question: Multiply each element of a list by a number

```
[1, 2, 3, 4, 5]   >> multiply by 5 >>     [5, 10, 15, 20, 25]
```

```
myList = [1, 2, 3, 4, 5]
myNewList = [i * 5 for i in myList]

print(myNewList)  # [5, 10, 15, 20, 25]
```

For example, in Matlab:
myList = myList .* 5

== vs is

```
>>> from copy import copy

>>> spam = [1, 2, 3]
>>> eggs = spam
>>> eggs == spam
True
>>> eggs is spam
True

>>> eggs = copy(spam)
>>> eggs == spam
True
>>> eggs is spam
False
```

is operator defines if both the variables point to the same object whereas **==** checks if the values for the two variables are the same

Dictionary

Also called associative arrays, maps, hashmaps, hashtables, etc.

- A **dictionary** is a collection which is unordered, changeable and indexed. They have keys and values defining *key-value pairs*.
  - `myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud’}`
  - *Keys >>> size, color, disposition*
  - *Values >>> fat, gray, loud*
- You can access these values through their keys:
  - `myCat[‘size’] >>> ‘fat’`
- You can use integers as keys too
- It is **mutable**

Add and remove items

```
spam = {}
spam[‘color’] =  ‘gray’   # spam = {‘color’: ‘gray’}


del spam[‘color’]   # Deletes item with key ‘color’
spam.pop(‘color)    # Remove item with key ’color’ and returns its value
spam.popitem()      # Removes an arbitrary item (the last item added) and
                    # returns its tuple
spam.clear()        # Removes all items
```

Dictionaries vs lists

- Items in dictionaries are unordered, so you cannot access them with an index value.
  - *The first item in a list named spam would be spam[0]. But there is no “first” item in a dictionary.*
- It does not matter in what order the key-value pairs are typed in a dictionary.
  - *Different order but same key-value pairs makes the same dictionary*
- Because dictionaries are not ordered, they can’t be sliced like lists
- KeyError for dictionary and IndexError for Lists.

Dictionaries vs lists

```
person1 = {}
person1['name'] = 'Phill’
person1['salary'] = 3500.0
person1['age'] = 22

person2 = {}
person2['age'] = 22
person2['salary'] = 3500.0
person2['name'] = 'Phill’

print(person1)
print(person2)
print(person1 == person2)
```

Dictionaries vs lists

```python
person1 = {}
person1['name'] = 'Phill’
person1['salary'] = 3500.0
person1['age'] = 22

person2 = {}
person2['age'] = 22
person2['salary'] = 3500.0
person2['name'] = 'Phill’

print(person1)  # {'name': 'Phill', 'salary': 3500.0, 'age': 22}
print(person2)  # {'age': 22, 'salary': 3500.0, 'name': 'Phill'}
print(person1 == person2)
```

Dictionaries vs lists

```
person1 = {}
person1['name'] = 'Phill’
person1['salary'] = 3500.0
person1['age'] = 22

person2 = {}
person2['age'] = 22
person2['salary'] = 3500.0
person2['name'] = 'Phill’

print(person1)  # {'name': 'Phill', 'salary': 3500.0, 'age': 22}
print(person2)  # {'age': 22, 'salary': 3500.0, 'name': 'Phill'}
print(person1 == person2)  # True
```

Dictionaries vs lists

```
list1 = []
list1.append(0)
list1.append(10)
list1.append(20)

list2 =[]
list2.append(10)
list2.append(20)
list2.append(0)

print(list1)  # [0, 10, 20]
print(list2)  # [10, 20, 0]
print(list1 == list2) # False
```

Dictionary methods

- .keys(), .values() and .items()
- The results of these methods are *dict_keys, dict_values,* and *dict_items* data types.
  - *It can be used in for loops!*

- You can use the multiple assignment trick in a for loop to assign the key and value to separate variables when using .items().

```
for key, value in myDict.items():
  print(key, value)
```

in and not in

- Like lists, you can use in and not in operators to check if a key or value exists in a dictionary.
  - *spam = {'name': 'Zophie', 'age': 7}*

    'name' in spam.keys()   >>> True

    'Zophie' in spam.values()*   *>>> ?

    ‘color' in spam.keys()   >>> ?

    'color' not in spam.keys()   >>> ?

    'color' in spam   >>> ?

in and not in

- Like lists, you can use in and not in operators to check if a key or value exists in a dictionary.
  - *spam = {'name': 'Zophie', 'age': 7}*

```
'name' in spam.keys()          >>> True
'Zophie' in spam.values()      >>> True
'color' in spam.keys()         >>> ?
'color' not in spam.keys()     >>> ?
'color' in spam                >>> ?
```

in and not in

- Like lists, you can use in and not in operators to check if a key or value exists in a dictionary.
  - *spam = {'name': 'Zophie', 'age': 7}*

    'name' in spam.keys() *>>> True*

    'Zophie' in spam.values() *>>> True*

    ‘color' in spam.keys() *>>> False*

    'color' not in spam.keys() *>>> ?*

    'color' in spam *>>> ?*

in and not in

- Like lists, you can use in and not in operators to check if a key or value exists in a dictionary.
  - *spam = {'name': 'Zophie', 'age': 7}*

```
'name' in spam.keys()             >>> True
'Zophie' in spam.values()         >>> True
‘color' in spam.keys()            >>> False
'color' not in spam.keys()        >>> True
‘name' in spam                    >>> ?
```

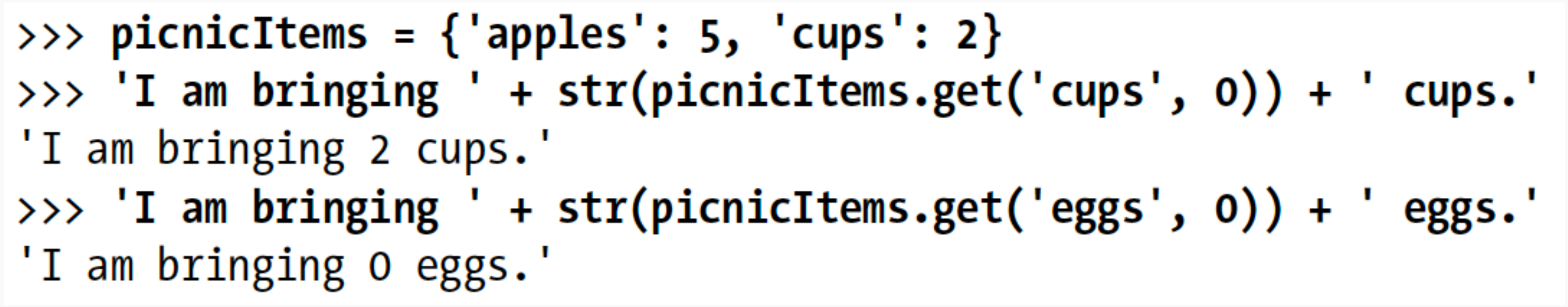
in and not in

- Like lists, you can use in and not in operators to check if a key or value exists in a dictionary.
  - *spam = {'name': 'Zophie', 'age': 7}*

'name' in spam.keys()	*>>> True*
'Zophie' in spam.values()	*>>> True*
‘color' in spam.keys()	*>>> False*
'color' not in spam.keys()	*>>> True*
‘name' in spam	*>>> True, same as **‘name' in spam.keys()***

.get(‘key’, 0)

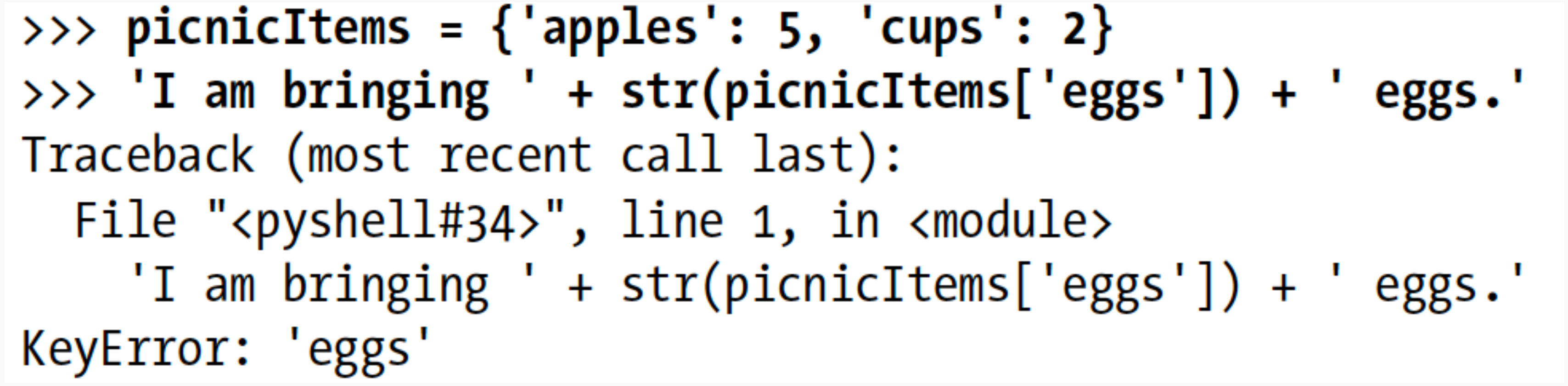
- Takes one or two arguments:
  - *the key of the value to retrieve. If key not found, the method will return **None**. >>> get(‘key’)*
  - ***Optional argument:** fallback value to return if that key does not exist. >>> get(‘key’, 0). If key not found, the method will return 0.*



```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

myDict[‘key’]

- We can also access the value of a pair through myDict[‘key’]
  - *But if the key is not found, it will give us an error (note the difference between the .get(‘key’) method)*



```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs'
```

.setdefault(‘key’,’value’)

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black'
```

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

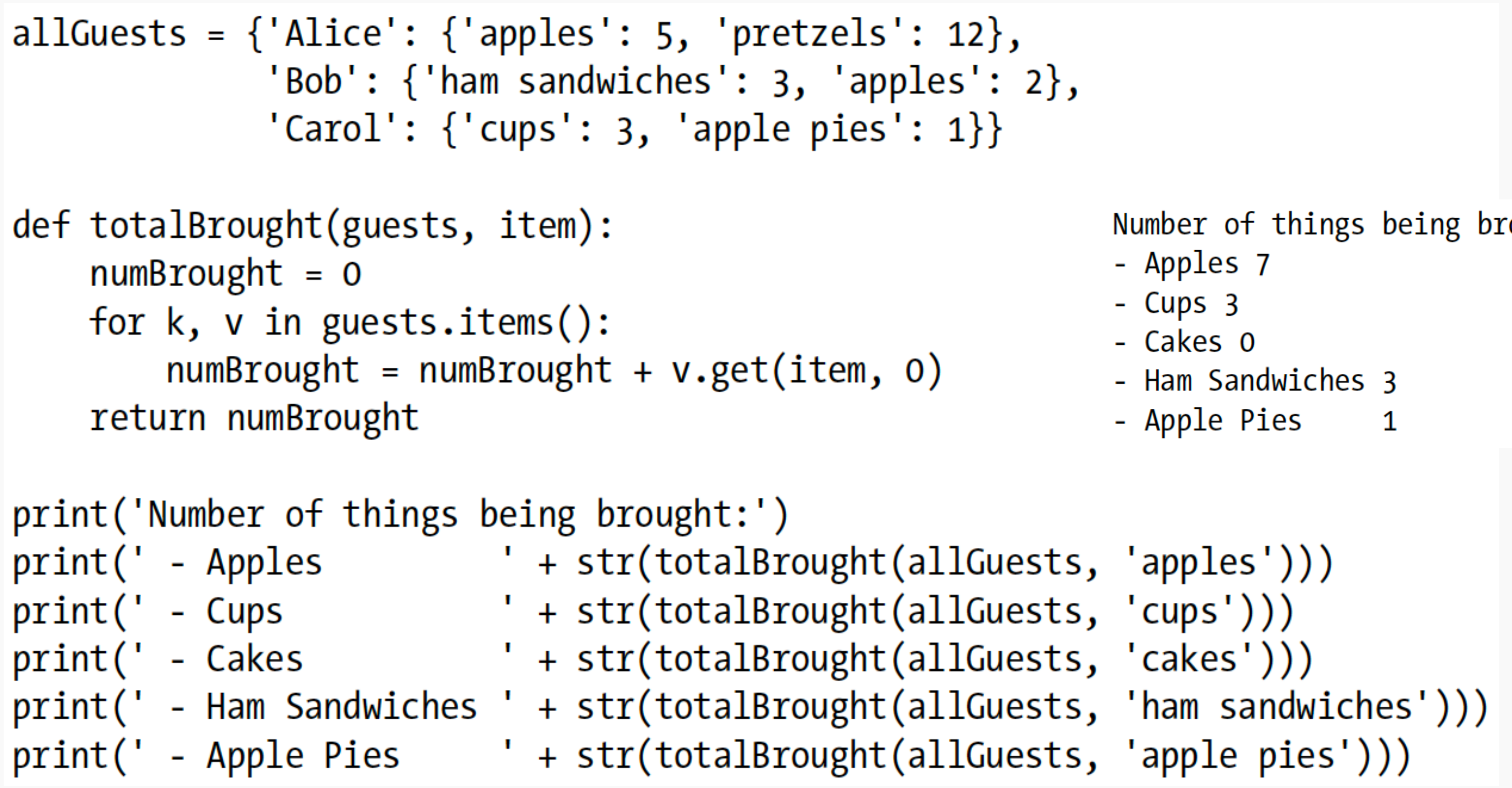
Nested Dictionaries and Lists

```python
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'ham sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple pies': 1}}

def totalBrought(guests, item):
    numBrought = 0
    for k, v in guests.items():
        numBrought = numBrought + v.get(item, 0)
    return numBrought

print('Number of things being brought:')
print(' - Apples         ' + str(totalBrought(allGuests, 'apples')))
print(' - Cups           ' + str(totalBrought(allGuests, 'cups')))
print(' - Cakes          ' + str(totalBrought(allGuests, 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests, 'ham sandwiches')))
print(' - Apple Pies     ' + str(totalBrought(allGuests, 'apple pies')))
```

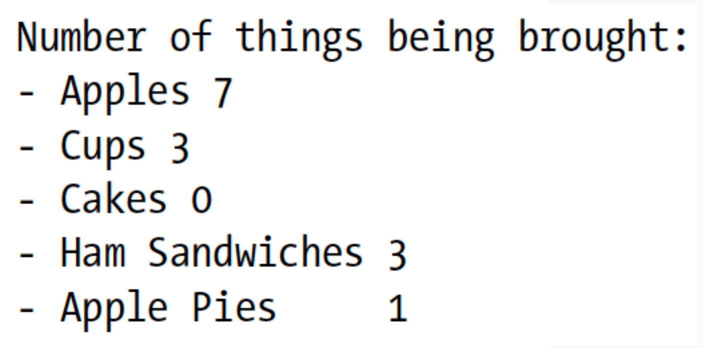
Nested Dictionaries and Lists



```python
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'ham sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple pies': 1}}

def totalBrought(guests, item):
    numBrought = 0
    for k, v in guests.items():
        numBrought = numBrought + v.get(item, 0)
    return numBrought

print('Number of things being brought:')
print(' - Apples         ' + str(totalBrought(allGuests, 'apples')))
print(' - Cups           ' + str(totalBrought(allGuests, 'cups')))
print(' - Cakes          ' + str(totalBrought(allGuests, 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests, 'ham sandwiches')))
print(' - Apple Pies     ' + str(totalBrought(allGuests, 'apple pies')))
```



```
Number of things being brought:
- Apples 7
- Cups 3
- Cakes 0
- Ham Sandwiches 3
- Apple Pies     1
```

Remember lists comprehensions?

```python
comp_list = [x ** 2 for x in range(7) if x % 2 == 0]
print(comp_list)
# [0, 4, 16, 36]
```

Dict comprehensions

- As list comprehension, we can create dict comprehensions
- Curly braces for dicts, brakets for lists

```
dict_comp = {x: chr(65+x) for x in range(1, 11)}

type(dict_comp)   # <class 'dict'>
print(dict_comp)
# {1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8:
# 'I', 9: 'J', 10: 'K'}
```