Intro to Computing
ES112

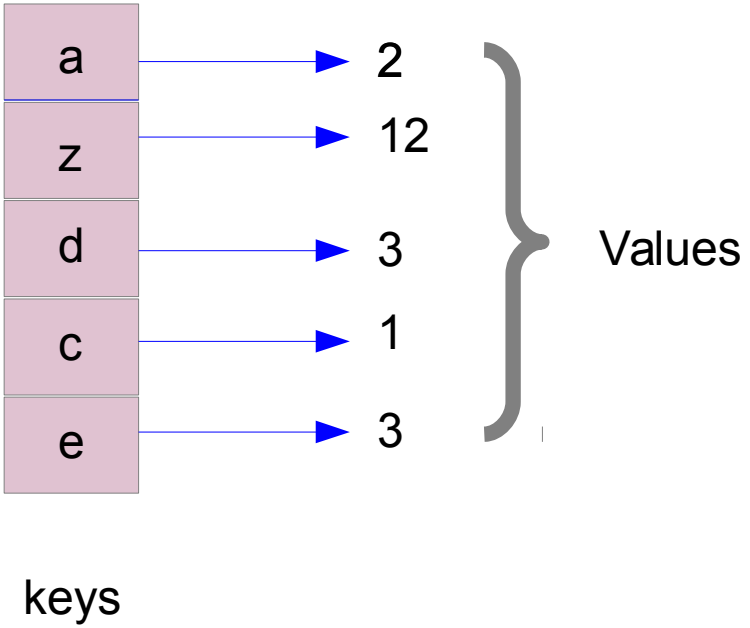
Lecture 5

Finding data values

- Suppose we have a list of student names and their grades
- We want to build a program such that
  - If anyone gives the program the name of a student, we can return their grades
  - Two students will not have exactly same names
- How can we do this?

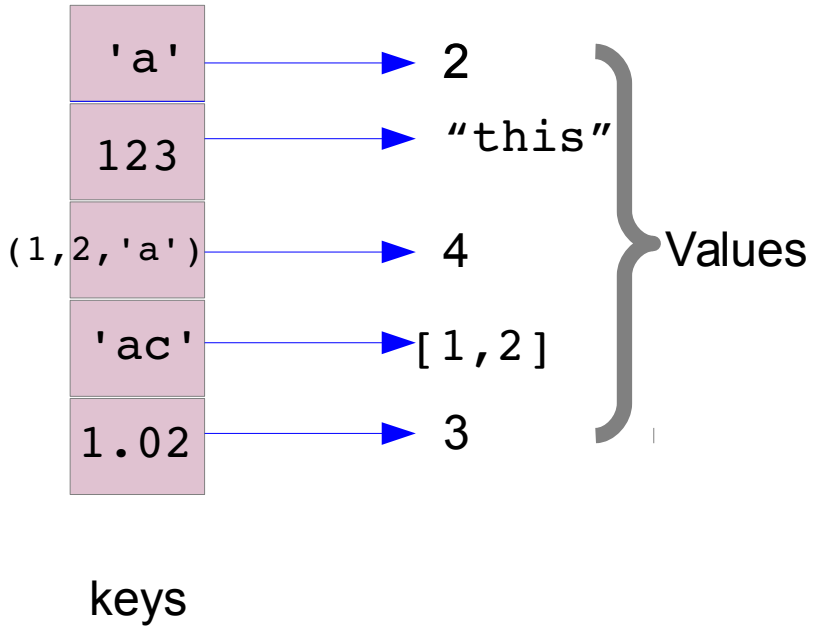
Dictionary

- Dictionary is key-value data structure, i.e. it can be used to store a set of keys and some values associated with them
  - The keys cannot be duplicate
  - Each key can have a value



Dictionary

- Dictionary is key-value data structure, i.e. it can be used to store a set of keys and some values associated with them
  - The keys cannot be duplicate
  - Each key can have a value
  - Values can be anything (int, float, list, string, dictionary...)
  - Keys can be anything “hashable”



Dictionary

- Dictionaries are data structures that have key-value pairs:
  - For each value of key, we store a value
- Creating a dictionary:
  - `example = {}`
  - `example = dict()`
  - `example={'first':1, 2:'second', 'third': 'another' }`
- Note that the types of keys and values can be different:
  - Values can be arbitrary
  - Keys can be numbers, strings, etc...
  - Anything “hashable” can be a key

Dictionary

- “Hashable” – in short, anything mutable is typically not hashable.
  - This behavior can be overridden (advanced)

- How to create and set a dictionary?

```
>>> a = {}
>>> a[1] = “this is the value”
>>> print a[1]
>>> a['cbe'] = 1
>>> print a['cbe']
```

Syntax of setting and accessing for a value is similar to that of a list

Dictionary

- Adding a new key-value pair to the dictionary
  - `example[k] = value`
  - `e.g. example[1]= “1”`
- Accessing an existing key-value:
  - `t = example[k]`
  - `print example[k]`
- Much like lists, except we need to provide a key instead of the index
- Notice there is a difference between `example[k]` and `example['k']`
- If `k` is not present as a key, then accessing `example[k]` will give KeyError

Checking a key

- We can check whether a key is present in a dictionary

```
>>> a = {}
>>> a[1] = “this is the value”
>>> 1 in a
True
>>> if(2 in a):
      print “2 is present”
    else:
      print “2 is not present”
2 is not present
>>>
```

- We have to use the in operator – returns True or False

More about dictionary

- Each key can only have one associated value
  - However, the value can be a list
- Getting the list of all keys:

```
ex={'first':1, 2:'second', 'third': 'another'}
allk = ex.keys()
print allk
```

- Getting list of all values:

```
allv = ex.values()
```

- Checking whether a particular key is in the dictionary
  - Using the keyword `in :` returns `True` or `False`

```
if (k in ex):
    print ex[k]
```

Dictionary

- Each key can have only one associated value
  - However, the value can be a list
- Obtaining the number of keys:
  - `len(ex)` or `len(ex.keys())`
- Suppose you are not sure that a key k is in dictionary:
  - `ex.get(k, defaultvalue)` : will return ex[k] is k is present in dictionary and will return defaultvalue else
  - `ex.get('first', 'Nothing')` returns 1
  - `ex.get('somekey', 'Nothing')` returns `'Nothing'`

Example

- Suppose we have a string that contains a sentence. We want to find out what is the histogram of words in the sentence
  - i.e. we want to output the list of unique words in the sentence, along with the count of number of times it appears

Creating a dictionary to count words

```
def histogram(s):
    d = dict()
    words = s.split()
    for c in words:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

- Steps are:
  - Create an empty dictionary
  - Split the string s (that contains the sentence) by whitespace
  - `words` contains the list of words
  - For each word, add it to the dictionary if not there
  - If already present, increment the count

What happens for this input?

"When you got nothing, you got nothing to lose"

Typical error

- Notice what would happen when we do this:

```
def histogram(s):
    d = dict()
    words = s.split()
    for c in words:
       d[c] += 1
    return d
```

Typical error

- Notice what would happen when we do this:

```
def histogram(s):
    d = dict()
    words = s.split()
    for c in words:
       d[c] += 1
    return d
```

- It gives a KeyError, since in order to increment, it is first accessing a key value that is not present in the dictionary

Dictionaries

- Important to note:
  - Suppose we add keys to the dictionary in a particular order
  - When we get the list of keys using `d.keys()`, the order need not be the same
- Lists **cannot be** keys in dictionary
  - Has to do with mutability of lists
  - Numbers, strings, tuples etc. will be the typical keys you use

Looping over a dictionary

- In order to write a loop over a dictionary keys we can just write

```
for k in d.keys():
  print k, d[k]
```

- In order to loop over the dictionary values:

```
for v in d.values():
   print v
```

Dictionary problem

- Suppose we have a list. We want to find out which are the distinct elements in the list. How to do this?

Dictionary problem

- Suppose we have a list. We want to find out which are the distinct elements in the list. How to do this?

- Create a dictionary of out of the elements of this list . Call it `d`

- Return `d.keys()`

Another example

- Suppose we have a dictionary d. We want to create another dictionary whose keys are the values of d and the values of d are the new keys
  - Note that the same value could be present in multiple keys

- What should be the keys and values of the new dictionary?

Reversing a dictionary

```
def invert_dict(d):
  inverse = dict()
  for key in d:
    val = d[key]
    if val not in inverse:
      inverse[val] = [key]
    else:
      inverse[val].append(key)
  return inverse
```

- Steps:
  - We will build a dictionary whose values contain lists
  - Go over each key in the dictionary
  - If not already added to the new dictionary, add it to the corresponding place