PROGRAMMING BOOTCAMP

Data structures pt II: Dictionaries

Lesson 5 – 9/20/16

Today's schedule

1. File writing
2. Dictionaries

1. File writing

File writing

Opening an output file is almost identical to input, with a small difference:

```
var = open(fileName, 'w')
```

Example:

```
outFile = open("seqs.txt", 'w')
```

File writing

Opening an output file is almost identical to input, with a small difference:

```
var = open(fileName, 'w')
```

Important: opening a file in 'w' mode will overwrite the file if it already exists.

Example:

```
outFile = open("seqs.txt", 'w')
```

Writing to an output file

Once the output file is opened, we use:

```
var.write(someStr)
```

Example:

```
outFile.write("This is output!\n")
```

Writing to an output file

Once the output file is opened, we use:

```
var.write(someStr)
```

Example:

```
outFile.write("This is output!\n")
```

Don't forget the newline!
Unlike `print`, `.write()` does not
insert this for you.

Simple example

Code

```
fileName = "output.txt"
outFile = open(fileName, 'w')
outFile.write("This is me,")
outFile.write("printing to \n a file.")
outFile.close()
```

output.txt

```
This is me,printing to
 a file.
```

Note the spacing and newline

Only strings can be printed

Code

```
fileName = "output.txt"
outFile = open(fileName, 'w')
outFile.write(25)
outFile.close()
```

Error:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    outFile.write(25)
TypeError: expected a character buffer object
```

Only strings can be printed

Code

```
fileName = "output.txt"
outFile = open(fileName, 'w')
outFile.write(str(25))
outFile.close()
```

A simple fix.

output.txt

```
25
```

Reading and writing can be done at the same time (as long as it's to different files)

Code

```
infile = "genes.txt"
outfile = "output.txt"
inFile = open(infileName, 'r')
outFile = open(outfileName, 'w')
for line in inFile:
    line = line.rstrip('\n')
    outFile.write("Found " + line + "\n")
outFile.close()
inFile.close()
```

genes.txt

```
uc007zzs.1
uc009akk.1
uc009eyb.1
uc008wzq.1
uc007hnl.1
```

output.txt

```
Found uc007zzs.1
Found uc009akk.1
Found uc009eyb.1
Found uc008wzq.1
Found uc007hnl.1
```

2. Dictionaries

Lists vs Dictionaries

Two main differences:

1. You retrieve elements from a dictionary using a "key", rather than an index
2. Dictionaries are unordered

1. Indexing by keys

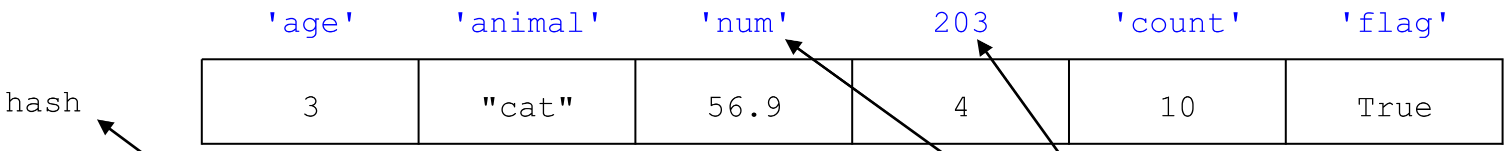
A dictionary is similar to a list, except instead of accessing elements by their index, you access them by a name ("key") that you pick.

	'age'	'animal'	'num'	203	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

```
>>> print dict["animal"]
cat
```

1. Indexing by keys

A dictionary is similar to a list, except instead of accessing elements by their index, you access them by a name ("key") that you pick.



	'age'	'animal'	'num'	203	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

```
>>> print dict["animal"]
cat
```

Dictionaries are similar to what other languages call "hash tables". So I might call them that sometimes.

Keys can be strings or numbers. You can use single quotes or double quotes around the strings; doesn't matter

2. Unordered

Lists are all about keeping elements in some order. Though you may change the ordering from time to time, it's still in *some* order.

You should think of **dictionaries** more like magic grab bags. You mark each piece of data with a key, then throw it in the bag. When you want that data back, you just tell the bag the key and it spits out the data assigned to that key.

2. Unordered

Lists are all about keeping elements in some order. Though you may change the ordering from time der.

You shoul e like magic grab bags ta with a key, then throw it in the bag. When you want that data back, you just tell the bag the key and it spits out the data assigned to that key.

Technicality:
Ok, so in reality, there *is* an order to your dictionary. But it is an order that Python picks that obeys complex rules and is essentially unpredictable by us. So for all intents and purposes, it may as well be unordered. Don't worry about it too much... just treat it like a magic grab bag and all will be well.

Practice with dictionary keys

	'age'	'animal'	'num'	205	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

What will this code print?

```
    print hash['count']
```

Practice with dictionary keys

	'age'	'animal'	'num'	205	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

What will this code print?

```
print hash['num']
```

Practice with dictionary keys

	'age'	'animal'	'num'	205	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

What will this code print?

```
print hash[age]
```

Practice with dictionary keys

	'age'	'animal'	'num'	205	'count'	'flag'
hash	3	"cat"	56.9	4	10	True

What will this code print?

```
var = 'animal'
print hash[var]
```

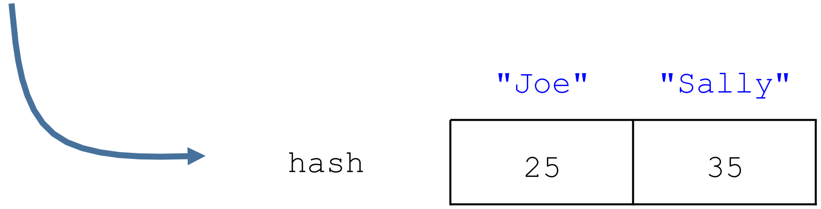
Creating a dictionary

Create an empty dictionary:

```
hash = {}
```

Create a dictionary with elements:

```
hash = {"Joe": 25, "Sally": 35}
```



"Joe" "Sally"

hash 25 35

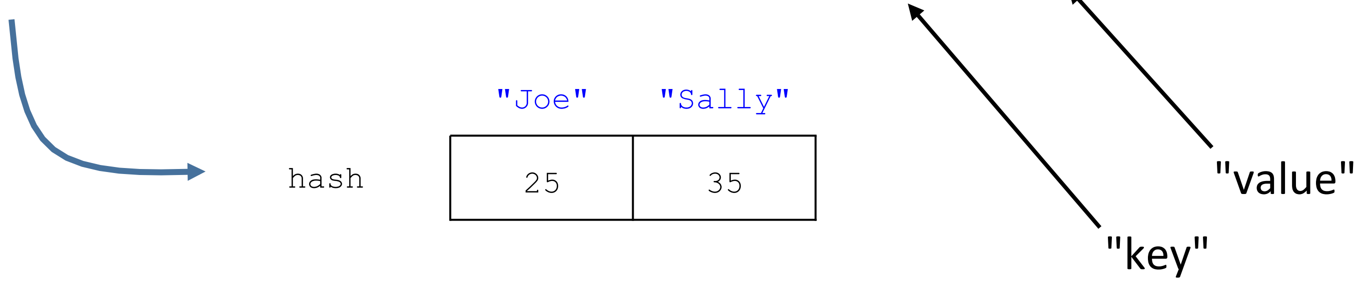
Creating a dictionary

Create an empty dictionary:

```
hash = {}
```

Create a dictionary with elements:

```
hash = {"Joe": 25, "Sally": 35}
```



"Joe" "Sally"

hash 25 35

"value"

"key"

Adding to a dictionary

Add entry:

```
hash[newKey] = newVal
```

Example:

```
>>> hash = {}
>>> hash["Joe"] = 25
>>> hash["Bob"] = 39
>>> print hash
{'Bob': 39, 'Joe': 25}
```

Adding to a dictionary

Add entry:

```
hash[newKey] = newVal
```

Example:

```
>>> hash = {}
>>> hash["Joe"] = 25
>>> hash["Bob"] = 39
>>> print hash
{'Bob': 39, 'Joe': 25}
```

Note that Python printed them in a different order than we entered them.

Removing from a dictionary

Delete entry:

```
del hash[existingKey]
```

Example:

```
>>> hash = {"name": "Joe", "age": 35, "job": "plumber"}
>>> print hash
{'job': 'plumber', 'age': 35, 'name': 'Joe'}

>>> del hash["age"]
>>> print hash
{'job': 'plumber', 'name': 'Joe'}
```

Phonebook example

Code:

```
phonebook = {}
phonebook["Joe Shmo"] = "958-273-7324"
phonebook["Sally Shmo"] = "958-273-9594"
phonebook["George Smith"] = "253-586-9933"

name = raw_input("Lookup number for: ")
print phonebook[name]
```

Output example:

```
Lookup number for: <we enter>Sally Shmo
958-273-9594
```

Phonebook example

Code:

```
phonebook = {}
phonebook["Joe Shmo"] = "958-273-7324"
phonebook["Sally Shmo"] = "958-273-9594"
phonebook["George Smith"] = "253-586-9933"

name = raw_input("Lookup number for: ")
print phonebook[name]
```

Notice that we can store the name of a key in a variable, and then use that variable to access the desired element. In this case, name holds the name that we input in the terminal, Sally Shmo.
What would happen if we entered a name that was not in the phonebook?

Output example:

```
Lookup number for: <we enter>Sally Shmo
958-273-9594
```

Checking if something is in the dict

This is the same as with a list. Use `in`:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

if "Joe" in ages:
    print "Yes, Joe is in the dictionary"
else:
    print "No, Joe is not in the dictionary"
```

Result:

```
Yes, Joe is in the dictionary
```

Dictionary methods

Here are some useful dictionary methods:

- `dict.keys()` - returns a **list** of the keys only
- `dict.values()` - returns a **list** of the values only
- `dict.items()` - returns a **list** of key-value pairs

Example:

```
>>> colors = {"apple": "red", "banana": "yellow", "grape": "purple"}
>>> colors.keys()
['grape', 'apple', 'banana']
>>> print colors.values()
['purple', 'red', 'yellow']
>>> print colors.items()
[('grape', 'purple'), ('apple', 'red'), ('banana', 'yellow')]
```

Using .keys()

Code:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for name in ages.keys():
    print name, "is in the dictionary."
```

Output:

```
Sally is in the dictionary.
Joe is in the dictionary.
George is in the dictionary.
```

Once again, notice that things are printed in a seemingly random order.

Using `.keys()`

Code:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for name in ages.keys():
    print name, "is", ages[name]
```

This gets the value associated with the name

Output:

```
Sally is 36
Joe is 35
George is 39
```

Using .keys()

Code:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for name in ages:
    print name, "is", ages[name]
```

Note that in a `for` loop, you can actually leave off the `.keys()`, because this is what python loops over by default when a dict is the iterable.

Output:

```
Sally is 36
Joe is 35
George is 39
```

Using .values()

Code:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for age in ages.values():
    print "There is a person who is", age
```

Output:

```
There is a person who is 36
There is a person who is 35
There is a person who is 39
```

The order is still random-seeming, but note that it's the same order as when we printed the keys.

Using `.items()`

Code:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for (name, age) in ages.items():
    print name, "is", age
```

`.items()` returns two variables each time it is called: a key and its value. This is why we can simultaneously assign the result to two variables

Output:

```
Sally is 36
Joe is 35
George is 39
```

Sorting a dictionary

You can **not** sort a dictionary. However, you can emulate sorting in the following way:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for name in sorted(ages.keys()):
    print name, "is", ages[name]
```

Output:

```
George is 39
Joe is 35
Sally is 36
```

Sorted based on person's name

Sorting by values

Occasionally, you'll also want to sort the keys of your dictionary based on their *value*, rather than the key itself. Here's one way to do it:

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39

for name in sorted(ages, key=ages.get):
    print name, "is", ages[name]
```

Output:

```
Joe is 35
Sally is 36
George is 39
```

← Sorted based on age rather than name

Nested Dictionaries

You can get creative with the values that are stored in dictionaries. For example, you can even have dictionaries as values!

```
peeps = {}
peeps["Joe"] = {}
peeps["Sally"] = {}
peeps["Joe"]["age"] = 35
peeps["Joe"]["color"] = "purple"
peeps["Sally"]["age"] = 36
peeps["Sally"]["color"] = "chartreuse”
print(peeps)
```

Output:

```
{'Sally': {'color': 'chartreuse', 'age': 36}, 'Joe': {'color':
'purple', 'age': 35}}
```

Terminology quiz

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39
```

"Joe" is most accurately referred to as...

a. an element

b. an index

c. a key

d. a value

Terminology quiz

```
ages = {}
ages["Joe"] = 35
ages["Sally"] = 36
ages["George"] = 39
```

35 is most accurately referred to as...

a. an element

b. an index

c. a key

d. a value

Terminology quiz

```
ages = []      #this is a list
ages[0] = 35
ages[1] = 36
ages[2] = 39
```

0 is most accurately referred to as...

a. an element

b. an index

c. a key

d. a value

Terminology quiz

```
ages = []      #this is a list
ages[0] = 35
ages[1] = 36
ages[2] = 39
```

39 is most accurately referred to as...

a. an element

b. an index

c. a key

d. a value