Introduction to Python Dictionaries

Oct 20, 2015

ACT2-4

- Do Task 1 – Practice spotting errors in functions

Debugging Programs

Function	Example
`addTwo(x,y)`	`addTwo(2,0)`
`subtractTwo(x,y)`	`subtractTwo(2,0)`
`multiplyTwo(x,y)`	`z = multiply(2,0)`
`divideTwo(x,y)`	`divideTwo(2,0)`
`addList(myList)`	`myList([2,0])`

Debugging Programs

Function	Example
`addTwo(x,y)`	`addTwo(2,0)` – output is wrong
`subtractTwo(x,y)`	`subtractTwo(2,0)` – any input causes an error
`multiplyTwo(x,y)`	`z = multiply(2,0)` – What is z after this assignment?
`divideTwo(x,y)`	`divideTwo(2,0)` – What happens when I run this? Use an "if" to catch the error and print a message to the screen.
`addList(myList)`	`myList([2,0])` – any input causes an error

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`		
`(1 <= 2) or ((1 == 1) and (1 != 2))`		
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2))`		
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2))`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`		
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2))`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`	`True or True`	`True`
`y = 100` `not(not(y == 100))`		

Boolean Expressions on Numbers

Expression	Evaluation	Final
`(100 > 101) and (-1 != -1)`	`False and False`	`False`
`(1 <= 2) or ((1 == 1) and (1 != 2))`	`True or (True and True)`	`True`
`x = 1` `not(x > 2) or (x <= x+1)`	`True or True`	`True`
`y = 100` `not(not(y == 100))`	`not(False)`	`True`

Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	**Example**	**Result**
Equality	`'a' == 'b'`	`False`
Inequality	`'a' != 'b'`	`True`

Boolean Expressions on Strings

Boolean Operators on Strings		
Operator	**Example**	**Result**
Equality	`'a' == 'b'`	`False`
Inequality	`'a' != 'b'`	`True`

```
>>> 'apple' == 'apple'
True
>>> 'apple' == 'Apple'
False
>>> 'apple' == 'apple!'
False
```

The Big Picture

Overall Goal
Build a Concordance of a text
- *Locations* of words
- *Frequency* of words

Today: Summary Statistics
- Get the vocabulary size of Moby Dick (Attempt 1)
  - Write test cases to make sure our program works
- Think of a *faster* way to compute the vocabulary size

Save ACT2-4.py and MobyDick.txt to the *same* directory

Writing a `vocabSize` Function

```
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniquelist)
```

Writing a `vocabSize` Function

```
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniquelist)
```

```
def noReplicates(wordList):
    '''takes a list as
argument, returns a list free
of replicate items.  slow
implementation.'''
```

```
def isElementOf(myElement,myList):
    '''takes a string and a list
and returns True if the string is
in the list and False
otherwise.'''
```

Writing a `vocabSize` Function

```
def vocabSize():
    myList = readMobyDickShort()
    uniqueList = noReplicates(myList)
    return len(uniquelist)
```

```
def noReplicates(wordList):
    '''takes a list as
argument, returns a list free
of replicate items.  slow
implementation.'''
```

```
def isElementOf(myElement,myList):
    '''takes a string and a list
and returns True if the string is
in the list and False
otherwise.'''
```

```
def testNoReplicates():
```

```
def testIsElementOf():
```

Writing *test cases* is important to make sure your program works!

Slow Implementation

```
def isElementOf(myElement,myList):
    '''Takes a string and a list and returns
       True if the string is in the list and False otherwise.'''
    for e in myList:
        if e == myElement:
            return True
    return False
```

```
def noReplicates(wordList):
    '''Takes a list as argument,
      returns list free of replicates'''
    newList = []
    for w in wordList:
        if isElementOf(w, newList) == False:
            newList = newList + [w]
    return newList
```

Many list traversals!

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Summary Statistics

- Get the vocabulary size of Moby Dick (Attempt 1)
  - Write test cases to make sure our program works
- Think of a *faster* way to compute the vocabulary size

What does `slow implementation` mean?

- Replace `readMobyDickShort()` with `readMobyDickAll()`
- Now, run `vocabSize()`
  - Hint: Ctrl-C (or Command-C) will abort the call.

What does `slow implementation` mean?

- Replace `readMobyDickShort()` with `readMobyDickAll()`
- Now, run `vocabSize()`
  - Hint: Ctrl-C (or Command-C) will abort the call.

- *Faster* way to write `noReplicates()`
  - What if we can sort the list?

```
['a','a','a','at','and','and',…,'zebra']
```

Sorting Lists

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

Sorting Lists

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

```
>>> myList = [0,4,1,5,-1,6]
>>> myList.sort()
>>> myList
[-1, 0, 1, 4, 5, 6]
```

Sorting Lists

Preloaded Functions			
Name	**Inputs**	**Outputs**	**CHANGES**
`sort`	List		Original List!

```
>>> myList = [0,4,1,5,-1,6]
>>> myList.sort()
>>> myList
[-1, 0, 1, 4, 5, 6]
>>> myList = ['b','d','c','a','z','i']
>>> myList.sort()
>>> myList
['a', 'b', 'c', 'd', 'i', 'z']
```

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Summary Statistics

- Get the vocabulary size of Moby Dick (Attempt 1)
  - Write test cases to make sure our program works
- Think of a *faster* way to compute the vocabulary size

Homework

- Sort your original list
- Make a new list, with initially only the first element of the original list

- For each element in the original list (from the second element on):
  - If that element is not the same as the previous
    - Add to the new list

Only *one* list traversal!

Remember what we're doing…

- Doing text analysis

- Introducing you to *computer programming*
  - In Python!
  - … and we are introducing these concepts *swiftly*

- Takes *practice*!
- Questions / office hours meetings are expected
  - We're here to help

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies

- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

Word Frequency: Inputs and Outputs

```
The cat had a hat. The cat sat on the hat.
```

I want to write a `wordFreq` function

Word Frequency: Inputs and Outputs

```
The cat had a hat. The cat sat on the hat.
```

I want to write a `wordFreq` function

- What is the input to `wordFreq`?

Word Frequency: Inputs and Outputs

```
The cat had a hat. The cat sat on the hat.
```

I want to write a `wordFreq` function

- What is the input to `wordFreq`?
- What is the output of `wordFreq`?

Word Frequency: Inputs and Outputs

```
The cat had a hat. The cat sat on the hat.
```

I want to write a `wordFreq` function

- What is the input to `wordFreq`?
- What is the output of `wordFreq`?

Word	Freq.
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

Word Frequency: Inputs and Outputs

```
The cat had a hat. The cat sat on the hat.
```

I want to write a `wordFreq` function

- What is the input to `wordFreq`?
- What is the output of `wordFreq`?
- We *could* do this with a list. How?

Word	Freq.
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies

- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

A New Data Structure

- A *Data Structure* is simply a way to store information.
  - **Lists** are a type of data structure
    - We can have **lists** of integers, floats, strings, booleans, or any combination.
    - Organized **linearly** (indexed by a range of integers)

A New Data Structure

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

A New Data Structure

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

Associate each **word** with the **frequency**.

A New Data Structure

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

Associate each **word** with the **frequency**.

Keys

Values

A New Data Structure

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

Associate each **word** with the **frequency**.

Keys Values

Key-Value Pairs

Key	Value
‘the’	3
‘cat’	2

A New Data Structure: Dictionaries

Keys can be *almost any* type or data structure.
Values can be *any* type or data structure.

Key Type	Value Type	Example Key	Example Value

A New Data Structure: Dictionaries

Keys can be *almost any* type or data structure.
Values can be *any* type or data structure.

Key Type	Value Type	Example Key	Example Value
`String`	`Integer`	`'the'`	`3`
`String`	`Integer`	`'cat'`	`2`

A New Data Structure: Dictionaries

Keys can be *almost any* type or data structure.
Values can be *any* type or data structure.

Key Type	Value Type	Example Key	Example Value
`String`	`Integer`	`'the'`	`3`
`String`	`Integer`	`'cat'`	`2`
`String`	`String`	`'Geisel'`	`'078-05-1120'`
`String`	`String`	`'Whitcher'`	`'552-38-5014'`

A New Data Structure: Dictionaries

Keys can be *almost any* type or data structure.
Values can be *any* type or data structure.

Key Type	Value Type	Example Key	Example Value
String	Integer	'the'	3
String	Integer	'cat'	2
String	String	'Geisel'	'078-05-1120'
String	String	'Whitcher'	'552-38-5014'
Float	String	1.0	'one point oh'
Float	String	2.8	'two point eight'

A New Data Structure: Dictionaries

Keys can be *almost any* type or data structure.
Values can be *any* type or data structure.

Key Type	Value Type	Example Key	Example Value
String	Integer	'the'	3
String	Integer	'cat'	2
String	String	'Geisel'	'078-05-1120'
String	String	'Whitcher'	'552-38-5014'
Float	String	1.0	'one point oh'
Float	String	2.8	'two point eight'
Integer	List	1638	[2, 3, 3, 7, 13]

A New Data Structure: Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

```
>>> freq = {}
>>> freq
{}
>>>
```

Initialize a Dictionary

A New Data Structure: Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

```
>>> freq = {}
>>> freq
{}
>>> freq['the'] = 3
>>> freq
{'the': 3}
>>>
```

Initialize a Dictionary

Key = 'the'
Value = 3

A New Data Structure: Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

```
>>> freq = {}
>>> freq
{}
>>> freq['the'] = 3
>>> freq
{'the': 3}
>>> freq['cat'] = 2
>>> freq
{'the': 3, 'cat': 2}
>>>
```

Initialize a Dictionary

Key = 'the'
Value = 3

Key = 'cat'
Value = 2

A New Data Structure: Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

```
>>> freq2 = {'the':3,'cat':2}
>>> freq2
{'the': 3, 'cat': 2}
>>>
```

Initialize a dictionary
with two key-value pairs

A New Data Structure: Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

```
>>> freq2 = {'the':3,'cat':2}
>>> freq2
{'the': 3, 'cat': 2}
>>>
>>> freq2['cat']
2
>>> freq2['the']
3
```

Retrieve a **value** using the **key**

Redefining things in the dictionary

```
The cat had a hat. The cat sat on the hat.
```

Word	Frequency
the	3
cat	7
had	1
a	2
hat	2
sat	1
on	1

```
>>> freq2 = {'the':3,'cat':2}
>>> freq2
{'the': 3, 'cat': 2}
>>>
>>> freq2['cat'] = 7
>>> freq2['a'] = freq2['a']+1
```

Assign a new value to a key!

The Big Picture

Overall Goal
Build a Concordance of a text
- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies
- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of keys	`>>> freq2.keys()` `['cat', 'the']`

- Keys Are Unique!
- Assigning/getting any value is very fast

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of keys	`>>> freq2.keys()` `['cat', 'the']`
`values()`	None	List of values	`>>> freq2.values()` `[2, 3]`

- Keys Are Unique!
- Assigning/getting any value is very fast

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of keys	`>>> freq2.keys()` `['the', 'cat']`
`values()`	None	List of values	`>>> freq2.values()` `[3, 2]`
`<key> in <dict>`	Key	Boolean	`>>> 'zebra' in freq2` `False`
`<key> in <dict>`	(same as above)		`>>> 'cat' in freq2` `True`

- Keys Are Unique!
- Assigning/getting any value is very fast

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of keys	`>>> freq2.keys()` `['the', 'cat']`
`values()`	None	List of values	`>>> freq2.values()` `[3, 2]`
`<key> in <dict>`	Key	Boolean	`>>> 'zebra' in freq2` `False`
`<key> in <dict>`	(same as above)		`>>> 'cat' in freq2` `True`
`del(<dict>[<key>])`	Dict. Entry	None	`>>> del(freq2['cat'])`

- Keys Are Unique!
- Assigning/getting any value is very fast

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of	`>>> freq2.keys()` `'cat']`
`values()`	None		`2.values()`
`<key> in <dict>`	Ke		`'a' in freq2`
`<key> in <dict>`	(same as above)		`>>> 'the' in freq2` `True`
`del(<dict>[<key>])`	Dict. Entry	None	`>>> del(freq2['cat'])`

Do Tasks 1 and 2

Let's try it!

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies

- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

Python Dictionaries

Function (All operate on Dictionaries)	Input	Output	Example
`keys()`	None	List of keys	`>>> freq2.keys()` `['the', 'cat']`
`values()`	None	List of values	`>>> freq2.values()` `[3, 2]`
`<key> in <dict>`	Key	True or False	`>>> 'zebra' in freq2` `False`
`<key> in <dict>`	(means same as above)		`>>> 'cat' in freq2` `True`
`del(<dict>[<key>])`	Dict. Entry	None	`>>> del(freq2['cat'])`

Python Dictionaries

```
The cat had a hat. The cat sat on the hat.
```

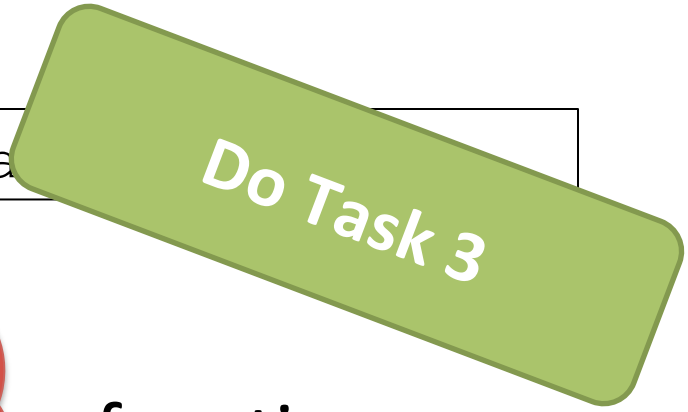
I want to write a `wordFreq` function

- What is the input to `wordFreq`?
- What is the output of `wordFreq`?

Word	Freq.
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

Python Dictionaries

```
The cat had a hat. The cat sa
```



I want to ...q function

- What is the inpu... ...rdFreq?
- What is the output of `wordFreq`?

Word	Freq.
the	3
cat	2
had	1
a	1
hat	2
sat	1
on	1

The Big Picture

Overall Goal

Build a Concordance of a text

- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies

- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

Building a Concordance

```
The cat had a hat. The cat sat on the hat.
0   1   2  3  4    5   6   7   8  9   10
```

Word	List of Positions	Frequency
the	[0,5,9]	3
cat	[1,6]	2
had	[2]	1
a	[3]	1
hat	[4,10]	2
sat	[7]	1
on	[8]	1

The Big Picture

Overall Goal
Build a Concordance of a text
- *Locations* of words
- *Frequency* of words

Today: Get Word Frequencies
- Define the inputs and the outputs
- Learn a new *data structure*
- Write a function to get word frequencies
- Go from word frequencies to a concordance (finally!)

This will be part of your next HW