CS 683 Emerging Technologies
Fall Semester, 2006
Assignment 1 Comments
Sep 21, 2006

Copyright ©, All rights reserved. 2006 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.

'dog'[0][0][0]

Java 'dog'.charAt(0).charAt(0)

Python

```
a = [1, 2 , 3]
a[0][0]
```

ascii Sum

```
def asciiSum(string):
    sum = 0
    for i in range(len(string)):
        sum = sum + ord(string[i])
    return sum
```

```
def asciiSum(string):
    sum = 0
    for char in string:
        sum = sum + ord(char)
    return sum
```

```
def asciiSum(string):
    return sum(map(ord,string))
```

Just Say No

```
def asciiSum(string):
    sum = 0
    for char in string:
        sum = sum + ord(char)
    print sum
```

```
def dollarWords(fileName):
    blah
    blah
    print dollarWords
```

```
class BankAccount:

    def balance(self):
        print self.balance

    def widthdrawal(self, amount):
        self.balance -= amount
        print 'You just widthdrew ' +
str(amount)
```

Dollar Words

```
def isCharacter(character):
    return ('a' <= character.lower()) & (character.lower() <= 'z')

def charValue(character):
    if not isCharacter(character):
        return 0
    return ord(character.lower()) - ord('a') + 1

def wordValue(word):
    return sum(map(charValue, word))

def isDollarWord(word):
    return wordValue(word) == 100

def dollarWords(words):
    return filter(isDollarWord, words)
```

```
def fileWords(fileName):
    file = open(fileName, 'r')
    words = file.read().split()
    file.close()
    return words

def fileDollarWords(fileName):
    return dollarWords(fileWords(fileName))
```

Test Cases

```
class TestDollarWords(unittest.TestCase):

    def setUp(self):
        sampleWords = open('sampleWords.txt','w')
        sampleWords.write('cat upcrowd bat\n')
        sampleWords.write('rat  Nutty')
        sampleWords.close()

    def testValue(self):
        self.assertEquals( charValue('a'),  1)
        self.assertEquals( charValue('A'),  1)
        self.assertEquals( charValue('z'), 26)
        self.assertEquals( charValue('.'), 0)

    def testWordValue(self):
        self.assertEquals( wordValue('cat'), 24)
        self.assertEquals( wordValue('CAT'), 24)
        self.assertEquals( wordValue(''), 0)
        self.assertEquals( wordValue('CONTENTED'), 100)
        self.assertEquals(dollarWords(['CONTENTED', 'cat']), ['CONTENTED'])
```

Test Cases

```
    def testFileWords(self):
        self.assertEquals( fileWords('sampleWords.txt'), ['cat', 'upcrowd', 'bat', 'rat', 'Nutty'])

    def testFileDollarWords(self):
        self.assertEquals( fileDollarWords('sampleWords.txt'), [ 'upcrowd', 'Nutty'])

if __name__ == '__main__':
    unittest.main()
```

Bank Account

```
class InvalidTransaction(Exception):
    pass

class BankAccount(object):

    def __init__(self):
        self.balance = 0

    def deposit(self, amount):
        if amount < 0:
            raise InvalidTransaction, 'Negative amount'
        self.balance = self.balance + amount

    def withdrawal(self,amount):
        if amount < 0:
            raise InvalidTransaction, 'Negative amount'
        if amount > self.balance:
            raise InvalidTransaction, 'Insufficent funds'
        self.balance = self.balance - amount
```

Test Cases

```
def testBankAccount(self):
    account = BankAccount()
    self.assertEquals(account.balance, 0)
    account.deposit(10)
    self.assertEquals(account.balance, 10)
    account.deposit(0)
    self.assertEquals(account.balance, 10)
    account.withdrawal(8)
    self.assertEquals(account.balance, 2)
    self.assertRaises(InvalidTransaction,account.withdrawal, 10)
    self.assertRaises(InvalidTransaction,account.withdrawal, -10)
    self.assertRaises(InvalidTransaction,account.deposit, -10)
```

Spacing

```
class BankAccount(object):
    def __init__(self):
        self.balance = 0
    def deposit(self, amount):
        if amount < 0:
            raise InvalidTransaction, 'Negative amount'
        self.balance = self.balance + amount
    def withdrawal(self,amount):
        if amount < 0:
            raise InvalidTransaction, 'Negative amount'
        if amount > self.balance:
            raise InvalidTransaction, 'Insufficent funds'
        self.balance = self.balance - amount
```

Spacing

```
class BankAccount(object):

    def __init__(self):

        self.balance = 0

    def deposit(self, amount):

        if amount < 0:

            raise InvalidTransaction, 'Negative amount'

        self.balance = self.balance + amount

    def withdrawal(self,amount):

        if amount < 0:

            raise InvalidTransaction, ¡'Negative amount'
```

Boolean Return

```
def isDollarWord(word):
    blah
    if wordSum == 100:
        return word
```

```
def isDollarWord(word):
    blah
    if wordSum == 100:
        return True
    else:
        return False
```

```
def isDollarWord(word):
    blah
    if wordSum == 100:
        return word
    else:
        return 0
```

```
def isDollarWord(word):
    blah
    return wordSum == 100
```

Names

```
class bankAccount:

    def Deposit(self, amount):
```

Name Standards

Item	Java	Smalltalk	C#	Python
Class	PascalCase	PascalCase	PascalCase	PascalCase
Method	camelCase	camelCase	PascalCase	foo_bar foobar camelCase
Field	camelCase	camelCase	camelCase	
Parameter	camelCase	camelCase	camelCase	
Local Variable	camelCase	camelCase	camelCase	

Name - Intention Revealing

```
def funtion(arg):
    blah

def compute(arg):
    blah

def isDollarWord(word):
    blah
```

Complex

```
def withdrawal(self, amount):
    if self.validateAmount(amount):
        if amount >= 0:
            if amount <= self.balance:
                self.balance -= amount
            else:
                raise ValueError, 'Not enough funds'
        else:
            raise ValueError, 'Amount is negative'
    else:
        raise ValueError, 'Amount not a number'
```

With Guard Statements

```
def withdrawal(self, amount):
    if self.validateAmount(amount):
        raise ValueError, 'Amount not a number'
    if amount < 0:
        raise ValueError, 'Amount is negative'
    if amount <= self.balance:
        raise ValueError, 'Not enough funds'

    self.balance -= amount
```

With Method to avoid comment

```
def withdrawal(self, amount):
    self.validateWithDrawalAmount(amount)
    self.balance -= amount
```

```
def validateWithDrawalAmount(self, amount):
    if self.validateAmount(amount):
        raise ValueError, 'Amount not a number'
    if amount < 0:
        raise ValueError, 'Amount is negative'
    if amount <= self.balance:
        raise ValueError, 'Not enough funds'
```

Catching Own Exceptions

```
def withdrawal(self, amount):
    try:
        if amount < 0:
            raise ValueError, 'Amount is negative'
        if amount <= self.balance:
            raise ValueError, 'Not enough funds'
        self.balance -= amount
    except ValueError, problem:
        print problem
```

```
def withdrawal(self, amount):
    if amount < 0:
        print 'Amount is negative'
        return
    if amount <= self.balance:
        print 'Not enough funds'
        return
    self.balance -= amount
```

Why?

```
def withdrawal(self, amount):
    try:
        if amount < 0:
            raise ValueError, 'Amount is negative'
        if amount <= self.balance:
            raise ValueError, 'Not enough funds'
        self.balance -= amount
    except ValueError, problem:
        print problem
        raise
```

Spider

```
import urllib
import re

def webpage(url):
    open = urllib.FancyURLopener({})
    webpageUrl = open.open(url)
    return webpageUrl.read()

def linksInPage(url):
    webPage = webpage(url)
    anchers = re.compile('href="([^""]*)"', re.I)
    return anchers.findall(webPage)
```