CPS 109 - Lab 8

Agenda 1

Today, we'll start by solving an example problem together.

This will show you how you might want to dissect the kinds of problems you may see on your exam.

Agenda 1

Today, we'll start by solving an example problem together.

This will show you how you might want to dissect the kinds of problems you may see on your exam.

Problem:

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palinedrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s) :
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Technique Summarized

1. What is the problem asking of me?
2. What are the input(s) to my function?
3. What are the output(s) of my function?
4. Are there any special instructions?
5. What do the unittests tell me?

Step 1: What is required?

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palindrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s) :
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Step 2: Inputs?

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palinedrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s):
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Step 3: Outputs?

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palinedrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s) :
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Step 4: Special Instructions?

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palinedrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s) :
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Step 5: Unittests?

28 lines (23 sloc) | 794 Bytes

Raw Blame

```
import unittest
'''
Assume that s is a string. Check to see if s is a palindrome.
If it is, return True. Else, return False. Recall that a
palinedrome is a string that is the same string if reversed.

You must use RECURSION to solve the problem.

For example,
recursive_palindrome('racecar') is True
recursive_palindrome('blue') is False

Three test caases have been included. Add two more,
both of which should be edge cases.
'''
def recursive_palindrome(s) :
    pass

class PalinTests(unittest.TestCase):
    def test1(self):
        self.assertTrue(recursive_palindrome('racecar'))
    def test2(self):
        self.assertFalse(recursive_palindrome('blue'))
    def test3(self):
        self.assertTrue(recursive_palindrome('madam'))

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

Recursion Technique

(This is taken from the exam review slides)

1. Make sure your function has a clear goal
2. Write out (then type up) your 1 or more base cases.
3. Design your recursive step.

Agenda 2

Today you're gonna learn how to write a program *properly*.

Whether you like it or not.

Writing a Program

Why do we bring this up now? You've been writing programs since the beginning of the semester.

Correction: you've been writing bad programs for the majority of the semester.

Writing a Program

Well that’s not very nice :(

A Word On Locality

A lot of you seem to still struggle with the concept of locality.

Recall: locality is the idea that variables themselves are used and destroyed within specific code blocks. You need to pass data around to local variables in order to maintain that data.

A Word On Locality

Passing around data? Understanding how functions use arguments?? Why can’t I just have variables declared in the ether or use *global* to declare global variables?

A Word On Locality

The issue with globals and ether variables (as I call them) is simple:

Say you have a large program, consisting of several files. Troubleshooting issues involving variables like that are now a nightmare and next to impossible.

Bad Programming Example

```
list = input("Enter a bunch of words seperated by spaces.").split()

for i in list:
    print(i)


if __name__ == "__main__":
    pass
```

Good Programming Example

```
def print_list(ls):
    for i in ls:
        print(ls)

if __name__ == "__main__":
    my_list = input("Please enter a series of words seperated by spaces:\n").split()
    print_list(my_list)
```

Bad Programming Example

```
a = input("iunput string:")
b = input("inpoot strung:")

def func():
    def func_two(a):
        print(a)
    def func_three(b):
        print(b)
    func_two()
    func_three()

func_two(a)
func_three(b)
```

Good Programming Example

```
def print_these_strings(str_one, str_two):
    print(str_one)
    print(str_two)

if __name__ == "__main__":
    a = input("Please input your first string:\n")
    b = input("Please input your second string:\n")
    print_these_strings(a, b)
```

Designing A Program

1. What is expected of me?
2. What is the functionality I need?
3. What functions and classes do I need to accomplish this?
4. Create skeleton for your code.
5. Write tests
6. Fill in functions and classes.
7. Write your main.

Designing A Program

Problem: I need a program that keeps track of employees. This should include basic information about them, their jobs, their departments, etc. I need to be able to add and delete employees, look up employees and give employees promotions.