Programming Exercises

Problem 1

Modify the recursive Fibonacci program given in the chapter so that it prints tracing information. Specifically, have the function print a message when it is called and when it returns. For example, the output should contain lines like these:

Computing fib(4)
...
Leaving fib(4) returning 3

Use your modified version of fib to compute fib(10) and count how many times fib(3) is computed in the process.

Solution

First, copy and paste the program as it’s written on the top of page 442 in your textbook. Second, you will need to add a print statement in three separate places: before you run the recursion conditions, beneath the condition if n < 3, and beneath the else statement. Remember to cast n into a character using the str function. Thirdly you will need to keep track of the number of times you run fib(3). This is not that hard. Simply declare a variable outside the function (I call mine numberOfThrees) and assign it the value of 0. In the function, insert a condition to check if n is equal to 3 (implying we’re running fib(3)). If so, add one to numberOfThrees. But remember to include the line global numberOfThrees at the beginning of fib so it knows to use that global variable instead of some local variable named numberOfThrees and it will cause an UnboundLocalError. Finally, below the fib functions definition, run fib(10) and print out numberOfThrees.

```python
numberOfThrees = 0
def fib(n):
    global numberOfThrees
    print("Computing fib(" + str(n) + ")")
    if n is 3:
        numberOfThrees +=1
    if n < 3:
        print("Leaving fib(" + str(n) + ") returning 1")
        return 1
    else:
        output = fib(n-1) + fib(n-2)
        print("Leaving fib(" + str(n) + ") returning " + str(output))
        return output

fib(10)
print(numberOfThrees)
```
Problem 3

A palindrome is a sentence that contains the same sequence of letters reading it either forwards or backwards. A classic example is: "Able was I, ere I saw Elba." Write a recursive function that detects whether a string is a palindrome.

Solution

I shortened the instructions to the basic idea.

The very first thing you should do is check to make sure there aren’t any punctuation marks, spaces, or capital letters to mess up your calculations. Use `string.punctuation` and remove every instance of a punctuation mark from the text using the `replace` function. Next, use `replace` to remove all spaces, and then use `lower` to turn all letters into lowercase (because, for example, a != A). Make sure you only do this once by checking if the entire text is made of alphabet characters using `not(quote.isalpha())`.

There are two base cases for this recursive function: if the remaining text is of length 1, and if the remaining text is of length 2. If it’s length 1, return True (because that’s how palindromes work). If it’s length 2, then check if both letters are the same then return True if so. Return False otherwise.

Finally, create a function that takes manual input from a user and determines if it is a palindrome.

```python
def isPalindrome(quote):
    # cleanup string
    if not(quote.isalpha()):
        for punc in string.punctuation:
            quote = quote.replace(punc, "")
        quote = quote.replace(" ", "")
        quote = quote.lower()
        print(quote)

    # base case
    if len(quote) is 1:
        return True
    if len(quote) is 2:
        if quote[0] is quote[1]:
            return True
        else:
            return False
    if quote[0] is quote[-1]:
        print(quote[1:-1])
        return isPalindrome(quote[1:-1])
    else:
        return False

def main():
    string = input("Give me something on the next line, and I will determine if it is a palindrome.")
    return isPalindrome(string)

main()
```
Problem 4

Write and test a recursive function max to find the largest number in a list. The max is the larger of the first item and the max of all the other items.

Solution

The wording of the exercise is somewhat vague and max is a built-in function, so I'm going to change it to maximum instead. What this exercise is trying to tell you is the following:

\[
\text{maximum}(\text{list}) = \text{whatever is greater: list[0] or maximum[the list minus the first entry]}
\]

The first thing you must do is find the base case, which in our case is...

\[
\text{len(alist)} == 2:
\text{return whichever item in alist is greater}
\]

Now we look for a general case, which will be

\[
\text{else:}
\text{calculate the maximum of the list - without the first element. set it to the variable a.}
\text{if the first element of the list is greater than a, return the first element else, return a.}
\]

def maximum(alist):
    if len(alist) == 2:
        if alist[0] > alist[1]:
            return alist[0]
        else:
            return alist[1]
    else:
        a = maximum(alist[1:])
        if alist[0] > a:
            return alist[0]
        else:
            return a
Problem 5

Computer scientists and mathematicians often use numbering systems other than base 10. Write a program that allows auger to enter a number and a base and then prints out the digits of the number in the new base. Use a recursive function `baseConversion(num, base)` to print the digits.

Hint: Consider base 10. To get the rightmost digit of a base 10 number, simply look at the remainder after dividing by 10. For example, 153%10 is 3. To get the remaining digits, you repeat the process on 15, which is just 153/10. This same process works for any base. The only problem is that we get the digits in reverse order (right to left).

Write a recursive function that first prints the digits of `num//base` and then prints the last digit, namely `num%base`. You should put a space between successive digits, since bases greater than ten will print out multi-character digits. For example, `baseConversion(245, 16)` should print out 15 5.

Solution

I know this seems incredibly complicated, but it’s not that hard. Remember when you had to learn about binary numbers? Those are numbers that use a base of 2 and only use the numbers 0 and 1 to count things. 01 is equal to 1, 1111 is equal to 15, etc. This question expands on that idea by illustrating an important fact: you can make a base out of any amount of numbers. The most frequently used ones are base 10 (digits), base 2 (binary), base 8 (octal), and base 16 (hexadecimal).

But what you’re being asked is to create a generalized function that can take any base 10 number and convert it into a different base. It takes a bit of thinking, but once you understand how the conversion works, you can make a recursive function.

The base case for this recursive function is when `num` has a value less than the `base` you wish to translate to. When this happens, you just return `str(num)`. I am using the `str()` function because although these are numbers, I want to output them in an order that’s not normal. I can try using a list of numbers and outputting those separated by spaces, but that’s a little too cumbersome for such a simple equation.

The general case for this recursive function is when `num` has a greater value than `base`. When that occurs, we have to split `num` into two different values. If I refer to the hint provided in the textbook, the first portion will pass `num // base` into `baseConversion`, and the second portion will pass `base%num` into `baseConversion`. // probably looks unfamiliar to you. It’s division that removes anything after the decimal point. So essentially `floor(base / num)`. It’s always a good idea to check your work by running through a recursive algorithm on pen and paper, because recursion is not that intuitive. However, it is amazingly elegant and simple, as you’ll see below.

```python
def baseConversion(num, base):
    if num < base:
        return str(num)
    else:
        return baseConversion(num // base, base) + " " + baseConversion(num % base, base)
```

```bash
7
```
Problem 6

Write a recursive function to print out the digits of a number in English. For example, if the number is 153, the output should be "One Five Three." See the hint from the previous problem for help on how this might be done.

Solution

This answer resembles the answer to the previous question. First, create a list of strings corresponding to the English words representing numbers. For instance:

```python
```

What’s nice about that list above is that each word corresponds to its index. `conversion[0]` is "Zero", `conversion[1]` is "One", etc.

Next, you need to convert your code to automatically handle base 10 numbers. Copy and paste the code from the previous problem into a new script. Anytime you see `base`, replace it with `10`. The other thing you must do is change the base case. Instead of returning the `str` format of a number, you are going to return its word in `str` format.

Also note that I initialized my list `conversion` outside of the function definition. In order for `numberInWords` (the name I used for this function) to work properly, I’ll need to use the `global` keyword to indicate that I’m using a variable from outside of the script.

```python
conversion = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]

def numberInWords(number):
    global conversion
    if number < 10:
        return conversion[number]
    else:
        return numberInWords(number // 10) + " " + numberInWords(number % 10)
```
Problem 10

Automated spell checkers are used to analyze documents and locate words that might be misspelled. These programs work by comparing each word in the document to a large dictionary (in the non-Python sense) of words. If the word is not found in the dictionary, it is flagged as potentially incorrect.

Write a program to perform spell-checking on a text file. To do this, you will need to get a large file of English words in alphabetical order. If you have a Unix or Linux system available, you might poke around for a file called `words`, usually located in `/usr/dict` or `/usr/share/dict`. Otherwise, a quick search on the Internet should turn up something usable.

Your program should prompt for a file to analyze and then try to look up every word in the file using binary search. If a word is not found in the dictionary, print it on the screen as potentially incorrect.

Solution

Problem 10 wants you to find a dictionary and use it to spell check words using binary search. Not an easy task.

Start by finding a dictionary. I chose to make a separate function to handle this called `uploadDictionary`, so I can keep this task separate from other tasks in the program. Mac users will find the dictionary in `/usr/share/dict/words`, which I have uploaded using the built-in `open` function. I have found an equivalent for both Mac and Windows users at the following location:

http://www-01.sil.org/linguistics/wordlists/english/wordlist/wordsEn.txt

Next, run a spell check to find every word in a text file using binary search. This is simpler than it seems because strings can use boolean operators like `>` and `<` to compare words alphabetically. For instance, "a" < "b" returns `True` because "a" is first in alphabetical order. And it works on entire words, too. For instance, "alphabet" > "boogie" returns `False`. I named my function `spellCheck`. An important note is that the instructions do not ask for a recursive implementation of binary search, so I used an iterative implementation instead. I’ve copied the version found on page 432 of your textbook, with some modifications to account for the switch from numbers to strings.

In the final function, which I have named `problem10`, we’ll upload a text file, check its words, and stick any problem words into a list. We’ll then output all the words that we think might be misspelled. Remember to remove all punctuation and turn all words into lower-case to prevent any mistakes in spell checking.

For simplicity’s sake, I’m turning every word in the dictionary into lowercase. Remember that "a" is not the same value as "A". Also somewhat surprisingly they do not include plural words (like "words"), so do not worry if they appear in your results.
```python
import string

def uploadDictionary():
    textfile = open("/usr/share/dict/words", "r")
    words = textfile.read()
    dictionary = []
    for eachWord in words.split():
        dictionary.append(eachWord.lower())
    return dictionary

def spellCheck(word, alist):
    low = 0
    high = len(alist) - 1
    #print("next word: " + word)
    while low <= high:
        mid = (low + high) // 2
        item = alist[mid]
        #print(" item: " + item)
        if word == item:
            return True
        elif word < item:
            high = mid - 1
        elif word > item:
            low = mid + 1
    return False #occurs only when we’re done inside the loop

def problem10(textFile):
    problemWords = []
    doc = open(textFile, "r")
    text = doc.read()
    for punc in string.punctuation:
        text = text.replace(punc, "")
    text = text.lower()
    dictionary = uploadDictionary()
    print(text.split())
    for eachWord in text.split():
        if not(spellCheck(eachWord, dictionary)):
            problemWords.append(eachWord)
    if len(problemWords) > 0:
        print("Your document has " + str(len(problemWords)) + " spelling errors.")
        for eachWord in problemWords:
            print(str(problemWords.index(eachWord) + 1) + " ", end="")
            print(eachWord)
    else:
        print("Your document has no spelling errors!")
```