lec7

August 12, 2016

In [ ]:
```
#Today we will see more examples of functions
#calling each other.
#You have already seen an example of this
#in lecture 6.
#And at the very end, we will cover a new topic
#called recursion which we will cover tomorrow
#and next week. Read lecture 4 handout
#before class tomorrow.
#It will probably be confusing at first
#because it is a new topic.
#But you will understand it over time.
```

In [ ]:
```
#some other things that you might have forgotten.
#You can have a list of lists
#e.g.
x=[1,2,3,4,5] #what is x[0]
x=['a','b','c','d']
#how about now?
x=[[1,2],3,[4,5,6,7]]
#what is x[0]? [1,2]
#what is x[1]? 3
#what is x[2]? [4,5,6,7]
#what is len(x)? 3
#What is x[0][0]?
#  1 because x[0] is [1,2] and [1,2][0] is 1.
#what is x[0][1]? 2.
#what is len(x[0][0])?
#  Error because len(1) cannot use len on int.
#what about len(x[0])? 2
```

In [104]:
```
#Say you want to swap the values of a and b.
#.e.g.
a=5
b=6

#a=b #now a is 6
#b=a #b=a now b is also 6
#new_a=a #new_a is 5
```

```
#a=b #now what is a? a is 6
#b=new_a #noew what is b? 5.

#and you want to have a=6 and b=5
#How would you do this?


a,b=b,a
print a
print b
```

```
6
5
```

In [ ]:
```
#Say you want to swap the values of a and b.
#.e.g.
a=5
b=6
#and you want to have a=6 and b=5
#How would you do this?
new_a=b
b=a
a=new_a

#an easy way to do this in python is
a,b=b,a
```

In [ ]:
```
#As we have seen before, one function can call another function.
#What does this function do?
#what is printed?
#Having more than one function
def passing_grade(h):
    if h>50:
        #print 'good'#True  <--Some people are confused by print Vs. return
        return True
    else:
        #print 'bad'#False
        return False

def candy_for_grade(g):
    if passing_grade(g):
        return 'candy'
    else:
        return 'no_candy'
y=candy_for_grade(51)
#print y

#what is the value of y?
```

```
In [ ]: #As we have seen before, one function can call another function.
        #What does this function do?
        #what is printed?
        #Having more than one function
        def passing_grade(h):
            if h>50:
                print 'good'#True  <--Some people are confused by print Vs. return
                return True
            else:
                print 'bad'#False
                return False

        def candy_for_grade(g):
            if passing_grade(g):
                return 'candy'
            else:
                return 'no_candy'
        y=candy_for_grade(51)
        print y

        #what is the value of y? Now? Rembmer
        #whether or not we print something, this
        #does not change
        #the value of why.

In [ ]: #In lab2, you have looked at the function
        #isPalindrome which
        #returns True if a given string is a palindrome.
        #I.e., isPalindrome('abcde') returns False.
        #returns False and isPalindrome('abcba')
        #returns True.
        #You have also seen a function called reverse
        #that returns the reversed verion of a string.
        #I give you a function called reverse. It takes in
        #a string, and returns the reversed version of
        #the string. So reverse('abcba') returns abcba.
        #reverse('12345') returns '54321'.
        #reverse(12345) gives an error.
        #can you think of a way to create the function
        #isPalindrom by using the function reverse?
        #Like I said in lecture before,
        #first try to write down in English
        #how the function would work.
        #.....

In [109]: def Reverse(x):
              y=''
              for n in range(len(x)):
```

```
            y += x[len(x)-n-1]
        print y
        return y


#def isPalindrome(x):
#    if Reverse(x)==x: #I am saying if None==x:
#        return True
#    else:
#        return False
#isPalindrome('abcba')

def isPalindrome(x):
    return Reverse(x)==x
isPalindrome('abcba')
```

```
Out[109]: True
```

```
In [111]: 5==5
```

```
Out[111]: True
```

```
In [46]: def isPalindrome(x):
             if x==reverse(x):
                 return True
             else:
                 return False

         isPalindrome('123210')
```

```
Out[46]: False
```

```
In [115]: #Another way
          def isPalindrome(x):
              x=str(x)
              return x==reverse(x)
          #this does the same thing as the function
          #isPalindrome above.
          #why?
          #if I give isPalindrome('12345') I want False
          #if I give isPalindrome(12345) I want False
          #if I give isPalindrome('abcba') I want True
          #if I give isPalindrome(12321) I want True
          #isPalindrome('12345')
```

```
Out[115]: False
```

```
In [48]: #Ok so now we want to have a function isPalindrome.
         #However, we want it
```

```
#to be able to have a string or an int as an input.
#the function we wrote above can only take a string.
#what is the output of the code below?

def reverse(x):
    y=''
    for n in range(len(x)):
      y += x[len(x)-n-1]
    return y

def isPalindrome(x):
    #x=str(x) #add this to use ints.
    if x==reverse(x):
        return True
    else:
        return False

isPalindrome(123210)
```

In [50]:
```
#We get a type error. Why? Because reverse only takes a string.
#where exactly is the error?
#1. We cannot use len on an int
#2. We cannot index into an int like this:
#       if x is an int we cannot say x[0]
#or x[1] etc...
#So how can we still have a function
#that takes a string or an int and returns
#whether or not that string or int is a palindrome?
#There are many ways of
#doing this. I will give you a few minutes to
#think of a few ways.
```

In [120]:
```
#One way to do this is to have 2 functions.
#One checks if a string is
#a palindrome. Another one checks if an int is
#a palindrome.
def reverse(x):
    '''
    This function takes a string. And returns
    the reversed version of the string.
    '''
    y=''
    for n in range(len(x)):
      y += x[len(x)-n-1]
    return y

def isStrPalindrome(x):
    '''
```

```
        This function takes a string and returns
        True if the string is a palindrome. It returns
        False otherwise.
        '''
        #return x==reverse(x)
        #same as
        if x==reverse(x):
            return True
        else:
            return False

    def isIntPalindrome(x):
        '''
        This function takes an int. And returns
        True if the int is a plandrome. It returns
        False otherwise.
        '''
        #what is I did x==reverse(x)
        #return str(x)==reverse(str(x))
        if str(x)==reverse(str(x)):
            return True
        else:
            return False

    def isPalindrome(x):
       if type(x)==str:
          #isStrPalindrome takes a string
          #and returns True if the string is
          #a palindrome. False if not.
          return isStrPalindrome(x)

       elif type(x)==int:
          #isStrPalindrome takes a int
          #and returns True if the int is
          #a palindrome. False if not.
          return isIntPalindrome(x)
       else:
          print 'Wrong data type'
          return

    y=isPalindrome(12.56)
    print y
```

```
Wrong data type
None
```

```
In [ ]: #Finally, we are going to cover recursion.
        #This might be a hard concept to understand
```

```
        #in the beginning. But with a little bit of
        #practice,
        #you'll get it.
```

In [ ]:
```
        #Lets start with an example.
        #consider the Fibonacci sequence
        #1, 1, 2, 3, 5, 8, 13, . . ..
        #This sequence is defined by the 0th and 1st
        #Fibonacci numbers both being 1,
        #and subsequent Fibonacci numbers being
        #the sum of the previous two.

        #The Fibonacci sequence is very well known in
        #Math and
        #appears frequently in nature.
        #It also has applications
        #in various computer science algorithms.

        #We will not discuss the Fibonacci sequence
        #in detail
        #in this class but if you are interested in
        #learning
        #more you can read the Wikipedia page here:
        #https://en.wikipedia.org/wiki/Fibonacci_number
```

In [ ]:
```
        #The Fibonacci sequence is defined by the 0th
        #and 1st
        #Fibonacci numbers both being 1,
        #and subsequent Fibonacci numbers being
        #the sum of the previous two
        #F(i) is 1 if i=0 or i=1
        #
        #F(i) is F(i−1) + F(i−2) otherwise

        #So what is F(0)?
        #    1 because i is 1
        #What is F(1)?
        #   1 because i is 1
        #What is F(2)
        #   It is F(i-1) + F(i-2)
        #   which is 1+1=2
        #F(3) is F(i-1)+F(i-2)=F(3-1)+F(3-2)
        #   which is F(2)+F(1)=2+1=3
        #F(4) is F(4-1)+F(4-2)=F(3)+F(2)=3+2=5
        # ....And it continues this way
```

In [ ]:
```
        ##Now let us make a function that takes i and
        #returns the ith Fibonacci number.
```

```
        #We can do it the way we have learned how to.
        #First we write in English how we want to create
        #the function.
```

In [84]:
```
        #Now let us make a function that takes i and
        #returns the ith Fibonacci number.
        #We can do it the way we have learned how to.
        #lets call the function fibonacci
        def fibonacci(i):
            fib_i=1
            fib_i_prev=1
            fib_i_prev_prev=1

            if i==0 or i==1:
                return fib_i
            for x in range(2,i+1):
                fib_i = fib_i_prev + fib_i_prev_prev
                fib_i_prev_prev=fib_i_prev
                fib_i_prev=fib_i
            return fib_i
        fibonacci(3)
```

Out[84]: 3

In [60]:
```
        #Although we could implement the Fibonacci sequence
        #in a very complicated way without using recursion,
        #the following function using recursion
        #is very simple and easier to understand
        def fibonacci(i):
            if i<2:
                return 1
            return fibonacci(i-1) + fibonacci(i-2)
        fibonacci(5)

        #compare this function to the function we
        #wrote above
        #which is more complicated and hard to understand
```

Out[60]: 8

In [18]:
```
        #Here is another example of a function that can
        #be done recursively. Multiplying 2 numbers
        a=5
        b=3
        a*b
        #a*b gives 15 as you all know
```

Out[18]: 15

In [31]:
```
#We can implement the multiplication of 2 numbers
#a and b in as recursive function only using
#addition.
#when you multiply 5*3, what do you do?
#5*3 is 5+5+5. The function below adds 5 to itself
#3 times. In genera, the function below takes
#inputs a and b and adds a to itself b times.
#This is the same as performing a*b.
def multiplyNumbers(a,b):
    print a,b
    if b == 0:
        return 0
    else:
        return a + (multiplyNumbers(a,(b - 1)))
multiplyNumbers(5,3)
#we can print a and b to see how many times
#multiplyNumbers is called and what the inputs(a,b)
#are each time it is called.
#Remember printing something
#is different from returning something. We are
#printing all of these values to
#figure out what the function is doing.
#This does not change what the
#function is returning.
```

```
5 3
5 2
5 1
5 0
```

Out[31]: 15