Functions and Modules

August 2017

1) Write a function sum that takes two integer parameters a and b and returns the sum of squares of all the integers n such that a <= n <= b.

```
def sum(a,b):
    total = 0
    for i in range(a,b+1):
        total += i**2
    return total

# tests
print("example output")
print("sum(1,3) is",sum(1,3)) #should be 14
print("sum(4,10) is",sum(4,10)) #should be 371
```

```
## example output
## sum(1,3) is 14
## sum(4,10) is 371
```

2. Write a function triangle that takes three real parameters which represent the lengths of three sides of a triangle. The function should return True if the triangle is right-angled and False otherwise.

```
def triangle(a,b,c):
    #right angled triangles satisfy a^2+b^2=c^2
    #for some combination of a,b,c
    right_angled = False
    if a**2 + b**2 == c**2 or\
        b**2 + c**2 == a**2 or\
        c**2 + a**2 == b**2:
        right_angled = True
    return right_angled

# tests
print("example output")
print("triangle(1,2,3) =",triangle(1,2,3)) #false
print("triangle(3,4,5) =",triangle(3,4,5)) #true
```

```
## example output
## triangle(1,2,3) = False
## triangle(3,4,5) = True
```

3. **a)** Write a function factorial that takes a non negative integer parameter and returns its facorial; we define 0! = 1. **b)** The number e can be defined as the limit of the infinite sum 1/0! + 1/1! + 1/2! + 1/3! + 1/4! +... Write a function compute_e which takes a positive integer parameter. This parameter indicates the number of terms that are to be summed (i.e. compute_e(3) should produce the sum 1/0! + 1/1! + 1/2!). Sum the terms and return the approximate value of e. Call/use the function you wrote in part a in your solution.

```
#a
def factorial(n):
    fact = 1
    for i in range(1,n+1):
        fact *= i
```

```
    return fact
#b
def compute_e(n):
    e_val = 0
    for j in range(0,n):
        e_val += 1./factorial(j)
    return e_val

# test
print("example output")
print("factorial(5) is", factorial(5)) #120
print("factorial(10) is", factorial(10)) #3628800
print("compute_e(3) is", compute_e(3)) #2.5
print("compute_e(11) is", compute_e(11)) #2.71828180114638454
```

```
## example output
## factorial(5) is 120
## factorial(10) is 3628800
## compute_e(3) is 2.5
## compute_e(11) is 2.7182818011463845
```

4. You have already written a program that calculates all prime numbers less than 100. Write a function primes that takes one integer parameter and returns the number of primes less than that integer.

```
def primes(n):
    if n==2:
        return [2]
    else:
        primes=[2]
        for p in range(3,n,2):
            prime = True
            for c in primes:
                if p % c == 0:
                    prime=False
                    break
            if prime:
                primes.append(p)
        return primes
#tests
print("example output")
print("primes(10) is", primes(10)) #2,3,5,7
print("primes(50) is", primes(50)) #2,3,5,7,11,13,17,19,23,29,31,37,41,43,47
```

```
## example output
## primes(10) is [2, 3, 5, 7]
## primes(50) is [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

5. Write a function solve_quadratic that takes three real parameters a, b, c (in that order). The function should return the roots of the quadratic equation ax**2 + bx + c = 0 as a tuple. If the roots are imaginary, the function should return "No real solutions"

```
def solve_quadratic(a,b,c):
    from math import sqrt
    d = b**2 - 4*a*c
    if d<0:
        return("No real solutions")
```

```
    else:
        return (-b - sqrt(d))/(2*a), (-b + sqrt(d))/(2*a)

# tests
print("example output")
print("solve_quadratic(1,-2,1) is ",solve_quadratic(1,-2,1))# should be [1,1]
print("solve_quadratic(1,0,-1) is ",solve_quadratic(1,0,-1))# should be [-1,1]
print("solve_quadratic(1,1,1) is ",solve_quadratic(1,1,1))# "no real solutions"
```

```
## example output
## solve_quadratic(1,-2,1) is  (1.0, 1.0)
## solve_quadratic(1,0,-1) is  (-1.0, 1.0)
## solve_quadratic(1,1,1) is  No real solutions
```

6. Write a function plot_polynomial that takes two integers and a list as parameters. The list contains the coefficients of the polynomial (with the first corresponding to the coefficient of the term of the highest degree). The two integers are the bounds of the domain of the polynomial. The function should produce a graph of the polynomial. (Figure **??** is an example of this graph.)

```
def plot_polynomial(a, b, L):
    import numpy as np
    import pylab as pl
    x_values = np.arange(a, b + 0.01, 0.01)
    y_values = np.zeros_like(x_values)

    for j,element in enumerate(x_values):
        y_value = 0
        for i in range(0, len(L)):
            y_value += L[i] * element ** (len(L) - 1 - i)
        y_values[j] = y_value

    pl.plot(x_values, y_values)
    pl.ylabel('y') ; pl.xlabel('x')

# example
import pylab as pl
from rmdplot import figsize, savefig
pl.figure(figsize=figsize
plot_polynomial(-1,1,[4,1,1]) #4x^2+x+1
plot_polynomial(-1,1,[3,0,0,2]) #3x^3+2
pl.legend(["$4x^2+x+1$","$3x^3+2$"])
savefig("figure/poly.pdf","Example Polynomial \label{fig:poly}")
```

File “”, line 20 plot_polynomial(-1,1,[4,1,1]) #4x^{2+x+1} SyntaxError: invalid syntax

7. Write a function lissajous that takes four real parameters a, b, c, d. Let a be the amplitude of oscillation in the x direction, b be the angular frequency of oscillation in the x direction, c be the amplitude of oscillation in the y direction, d be the angular frequency of oscillation in the y direction. Assume the oscillations are in phase and described by cosine waves. The function should display the Lissajous figure for this motion on the time interval [0, 1000]. (Figure 1 is an example of this plot.)

```
def lissajous(a,b,c,d):
    import numpy as np
    import pylab as pl
    time = np.arange(0,100,0.1)
```

```
    x = a*np.cos(b*time)
    y = c*np.cos(d*time)
    pl.plot(x,y)
    pl.xlabel('x'); pl.ylabel('y')
# example
import pylab as pl
from rmdplot import figsize, savefig
pl.figure(figsize=(6,4))
pl.subplots_adjust(wspace=0.3)
pl.subplot(121)
lissajous(1,0.1,3,1.0)
pl.title("lissajous(1,0.1,3,1)")
pl.subplot(122)
lissajous(1,.1,3,0.2)
pl.title("lissajous(1,0.1,3,0.2)")
savefig("figure/lissajous.pdf","Example Lissajous \label{fig:lissa}")
```

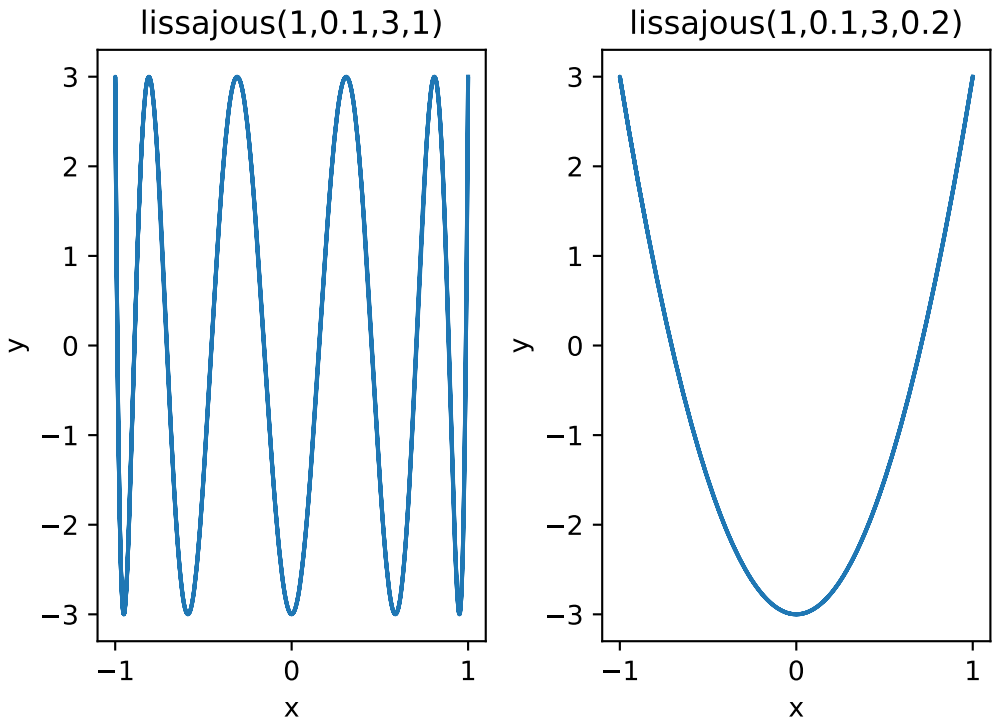

Figure 1: Example Lissajous

8. A quick introduction to recursive functions. A function that calls itself is called recursive. Here is an example of recursion to calculate a factorial, which you should have already done using a loop:

```
def factorial(n):
    if n <= 1: return 1
    else: return n * factorial(n-1)
```

Here we have a base case (n = 0 or n = 1) and the function calls itself to reduce any other case to the base case. Write a function sum() that takes a single integer parameter n. It should return the sum of all the positive integer s up to and including n (i.e. sum(5) = 1 + 2 + 3 + 4 + 5 = 15). Use recursion.

```python
def sum(n):
    if n==1:
        return 1
    else:
        return n+sum(n-1)

#tests
print("example output")
print("sum(10)=",sum(10)) # 55
print("sum(100)=",sum(100)) # 5050
```

```
## example output
## sum(10)= 55
## sum(100)= 5050
```