Lab 5

User-defined Modules

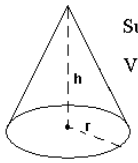
We have used several built-in modules or libraries like **math**, **turtle**, and **sys**. In this lab you will build several user-defined modules or libraries. A ***module*** or ***library*** allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. In its simplest form, a module is a file consisting of Python code. A module can define ***functions***, ***variables***, and ***classes*** (we will look at classes in the last third of CS8). A module can also include runnable code!

Part A

Create a module named cone.py. Define three functions with arguments r (radius) and h (height): volume(r, h), surface_area(r, h), and printstats(r, h) to calculate the volume and surface area of a cone with radius r and height h. printstats(r, h) displays to two decimal places both the volume and surface_area. Don't forget to document each function by describing its purpose (this is used by Python's `help()` function to display information about a function). For example to document the volume() function do as follows,

```
def volume(r, h):
    '''
    volume(r,h) computes the volume of a right-circular cone with radius r and height h.
    '''
    …
```

Now test the cone module as illustrated in the Python Shell. Note what gets displayed when `help(volume)` is called.



$$\text{Surface area} = \pi r \sqrt{r^2 + h^2}$$

$$\text{Volume} = \frac{1}{3}\pi r^2 h$$

```
>>> from cone import *
>>>
>>> help(volume)
    Help on function volume in module cone:

    volume(r, h)
        volume(r,h) computes the volume of a right-circular cone with radius r and height h.

>>> volume(4,7)
    117.28612573401894
>>>
>>>
>>> help(surface_area)
    Help on function surface_area in module cone:

    surface_area(r, h)
        surface_area(r,h) computes the surface area of a right-circular cone with radius r and height h.

>>> surface_area(10,2)
    320.38084488828184
>>>
>>>
>>> help(printstats)
    Help on function printstats in module cone:

    printstats(r, h)
        printstats(r,h) displays a summary of a right-circular cone's volume and surface area.

>>> printstats(7,11)
    volume = 564.44
    surface area = 286.73
```

Part B

Create a module named specialsums.py. Define three functions with one argument n, an integer: gauss(n), sumofsqs(n), and sumofcubes(n). Note that the caller provides the value for n.

gauss(n) computes the sum:

$1 + 2 + 3 + ... + n$

sumofsqs(n) computes:

$1 + 2^2 + 3^2 + ... + n^2$

sumofcubes(n) computes:

$1 + 2^3 + 3^3 + ... + n^3$

Make sure your functions work properly, that is, returns 0 when n is 0 or a negative integer. Now test the specialsums module as illustrated in the Python Shell.

```
>>> import specialsums
>>>
>>> # find the sum 1 + 2 + 3 + ... + 99 + 100
>>> specialsums.gauss(100)
    5050
>>>
>>>
>>> # find the sum 1*1 + 2*2 + 3*3 + ... + 99*99 + 100*100
>>> specialsums.sumofsqs(100)
    338350
>>>
>>>
>>> # find the sum 1*1*1 + 2*2*2 + 3*3*3 + ... + 99*99*99 + 100*100*100
>>> specialsums.sumofcubes(100)
    25502500
>>>
>>>
>>> help(specialsums.gauss)
    Help on function gauss in module specialsums:

    gauss(n)
        Carl Fredrick Gauss found a formula for 1 + 2 + 3 + ... + n in 2nd grade.
        We honor him by calling this sum, Gauss' Sum.

>>>
```

By placing a description at the beginning of this module (use ''' document here ''') anyone using it can ask for `help(specialsums)` to display each function and their description.

```
>>> help(specialsums)
    Help on module specialsums:

    NAME
        specialsums

    DESCRIPTION
        Module: specialsums contains functions that sum consecutive integers, sum squares
        of consecutive integers, and sum cubes of consecutive integers.

    FUNCTIONS
        gauss(n)
            Carl Fredrick Gauss found a formula for 1 + 2 + 3 + ... + n in 2nd grade.
            We honor him by calling this sum, Gauss' Sum.

        sumconsecutive(m, n)
            This function computes the sum:
                    m + (m+1) + (m+2) + ... + n, when 1 <= m < n

                    1 + 2 + 3 + ... + n, when m == n >= 1

                    n + (n+1) + (n+2) + ... + m, when 1 <= n < m.

        sumofcubes(n)
            This function computes the sum of the cubes of the integers from 1 to n.

        sumofsqs(n)
            This function computes the sum of the squares of the integers from 1 to n.

    FILE
        /Users/novacky/Courses/cs8/labs/lab 6/Modules lab/specialsums.py
```

Exercise: Implement the fourth function `sumconsecutive(m,n)` described above.

The dir( ) Function

The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in the specified module. Here is an example for you to run.

dir.py - /Users/novacky/cs8/CS8 Syllabus Docs/class10/dir.py (3.4.3)

```
import cone
import specialsums

print("Names associated with the cone module:\n", dir(cone))
print()
print()

print("Names associated with the specialsums module:\n", dir(specialsums))
print()
print()
```

Ln: 11 Col: 0

Here is the output produced:

```
names associated with the cone module:
 ['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'pi', 'printstats', 'sqrt', 'surface_area', 'volume']


['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'gauss', 'sumconsecutive', 'sumofcubes', 'sumofsqs']
```

Timing Code in Python

In computer science, one way to test the efficiency of a function or program segment is to time the execution of the function call or program segment on different argument values and/or on different sets of data.

The time() function returns the time (in seconds) from January 1, 1970 (This is when time began, because the Unix operating system was created then). Here is an outline of how function calls and program segments are timed.

```
import time
start = time.time()

###  code to be timed goes here (place function call here)

stop = time.time()
print('%1.2f seconds' % (stop - start))
```

Part C

Two ways to compute the cube of a number **u** is to use the exponentiation operator to form u**3 or to use repeated multiplication to form u*u*u. Time the function sumofcubes(N) using these two ways of calculating a cube of a number, for the following values of N.

sumofcubes(N) with u**3

N	Time (sec)
5000000	
10000000	
15000000	
20000000	
25000000	

sumofcubes(N) with u*u*u

N	Time (sec)
5000000	
10000000	
15000000	
20000000	
25000000	