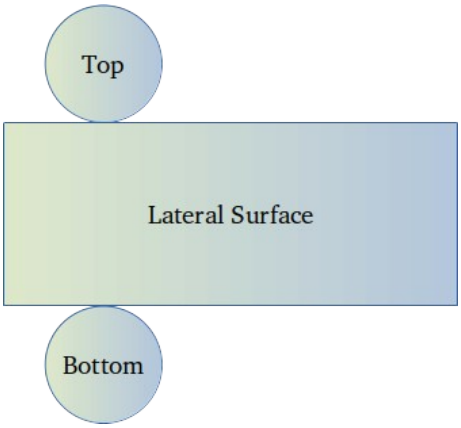
ICS3U: Calling Functions From Within Other Functions

We have already seen examples of nesting functions in Python, such as when typecasting user input to an integer as in `int(input("Enter a value: "))` or when calculating the sine of an angle in degrees, as in `math.sin(math.radians(30))`. We can do the same with functions that we write ourselves. For example, let's start with some simple functions that calculate some measurements for basic shapes. We have tested these out to ensure that they work as intended.

```
import math
def area_rectangle(length, width):
   area = length * width
   return area
def area_circle(radius):
   area = math.pi * radius**2
   return area
def circumference(radius):
   circ = 2 * math.pi * radius
   return circ
```

Now let's create another function that calculates the surface area of a cylinder. In the simplest case (a full cylinder) we could simply use the formula $SA_{cyl}=2\pi r(r+h)$. For our program, however, we are going to ask the user if they would like to include exclude either the top or bottom (or both) of the cylinder from the calculation. This will allow for other clyindrical shapes, like an open can or a tube. We can modify the mathematical formula to exclude these pieces, but an alternate method is to "build up" a cylinder instead, by adding together the three parts that make it up: circles for the top and bottom, and a rectangle for the lateral component. Note that this rectangle has dimensions equal to the height of the cylinder and the circumference of the top and bottom.




```
def surface_area_cylinder(height, radius, top, bottom):
   SA = area_rectangle(circumference(radius), height)
   if top in "Yy":
      SA += area_circle(radius)
   if bottom in "Yy":
      SA += area_circle(radius)
   return SA
```

The function works by first calculating the lateral surface area and assigning this to `SA`. If any of the two circles are required, their areas are added to `SA`. Note that we are making references to the basic functions `circumference`, `area_rectangle` and `area_circle` from *within* our `surface_area_cylinder` function. There is nothing preventing us from writing `surface_area_cylinder` without the use of these functions, but since we know that they have been tested and that they work, we can use them to do the cylinder calculations for us. Also note the nesting of functions in the call `SA = area_rectangle(circumference(radius), height)`.

The main program itself might look something like this.

```
height = float(input("Enter the height of the cylinder: "))
radius = float(input("Enter its radius: "))
top = input("Include top (Y/N)? ")
while top != "Y" and top != "y" and top != "N" and top != "n":
   top = input("You must enter Y or N: ")
bottom = input("Include bottom (Y/N)? ")
while bottom != "Y" and bottom != "y" and bottom != "N" and bottom != "n":
   bottom = input("You must enter Y or N: ")
SA = surface_area_cylinder(height, radius, top, bottom)
print("The surface area of the cylinder is", SA)
```

Note that the main program's purpose is to obtain user input and to call the functions that perform the actual calculations. As our programs get larger and more complex, this is generally the format that they will take: the main program will be used to delegate tasks to small, single-purpose functions that handle most of the logic involved. This is not only easier to maintain and to test, but it allows for a program to be written by a *team* of programmers rather than by a single person, as everyone can take on a set of tasks.

Here is another example of calling functions from others. The program below generates random "words" of 3-8 letters from a sequence of available characters, then displays them as a "paragraph" of 10 words. Of course, none of them are actual English words, unless you get very lucky and manage to generate one.

```
import random

def random_word(available_letters, word_length):
   word = ""
   for count in range(word_length):
      word += random.choice(available_letters)
   return word

def random_paragraph(available_letters, num_words):
   par = random_word(available_letters, random.randint(3,8))
   for count in range(num_words-1):
      word = random_word(available_letters, random.randint(3,8))
      par += (" " + word)
   return par + "."

paragraph = random_paragraph("ABCDEFG", 10)
print(paragraph)
```

Try running the program several times to test it out. Try using the debugger to trace through the program as it calls `random_word` from within `random_paragraph`, so that you can see when the program transfers execution from one function to the next.