Variables

The values in the expressions that we are working with often have more meaning than the numbers alone reveal. For example, the expression

```
0.5 * 10 * 5
```

might be the area of a triangle, where the value 10 is the base and 5 is the height. To associate a name with those values, we use variables. The general form of an assignment statement is

```
variable = expression
```

Assign the value 10 to a variable named base and the value 5 to a variable height. Now rewrite the expression

```
0.5 * 10 * 5
```

so that it uses variables base and height. Next, assign the expression to a variable named area. Reassign variables base and height the values 12 and 17, respectively. What is the value of area now? Type area and the return key to see area's value. Verify the current variable values by typing each variable in the shell followed by the return key. Now recalculate the area (using the new values of the variables) and check it's value.

Functions and Modules

In the previous section, we calculated the area of a triangle. Let's define a function, called calculate_area, so that we can reuse the code we wrote. The general form of a function is:

```
def function_name(parameters):
    body
```

In the editor section of Wing, define a function calculate_area, parameters: triangle_base and triangle_height (note:any legal variable names would do, but it's best to choose something meaningful).

The function should calculate the triangle's area (recall area = 0.5*base*height) for the given base and height, and return the area. Save the file: File -> Save. A dialog box will appear. In the text field labelled File name: replace **`untitled-1.py`** with **`triangle.py`**. In the Save in: drop down menu, select your own directory: the line that starts iwth your user id and continues with 'on CCentre Student Server...'. Create a new folder called cscb20s. Open cscb20s, and create a folder LabPractice. Open LabPractice. Click the Run button. In the shell, call the function as follows:

```
calculate_area(10, 12)
calculate_area(3.4, 5.2)
calculate_area(base, height)
```

Explore Module `math` using `dir` and `help`

Complete the tasks below using functions from Python's math module. Use **`dir(math)`** to find out which functions math contains and use **`help(math.function)`** (where function is replaced by the name of a function from math) to find out more about a particular function. To see detailed information about all math functions, call **`help(math)`**. In the shell, import math and write code to do the following:

a. Calculate 2 * pi.
b. Take the floor and ceiling of 56.4.
c. Calculate the square root of the logarithm of 244.
d. Calculate the base 2 logarithm of 256.

Manipulating type `str`

The Python type **`str`** stands for string. A string is a sequence of characters. We can iterate over (walk through) a str in a loop and examine each character. In the Python shell, do the following: Assign the string "superconductivity" to a variable s. (If your fingers are up to it, you may use the string "supercalifragilisticexpialidocious".) For the following print statements, recall that "+" combines strings (concatenates them), and "," can be used to separate items in a print statement.

a. Write a loop that prints each character in s on a separate line.
b. Write a loop that prints each character in s on the same line with a space after each character. Notice where the **`>>>`** prompt appears.
c. Write a loop that prints each character in s on the same line, with a space and a comma after each character: s , u , p , e , r , c , etc.
d. Write a loop similar to the previous but without a space between each character and the comma that follows it: s, u, p, e, r, c, etc. Notice that there is a comma after the last character. We won't worry about this now.

Using `str` Methods

Methods act much like functions. The only difference is that a method is a function that belong to specific data types and is designed to make use of the specific value it is 'called on'. For example, **`'abc'.upper()`**, a str method, uses the value 'abc' to return its uppercase equivalent, 'ABC'. What methods a value has depends on its type: 'abc' has a method upper() because 'abc' is a str. Explore the str methods using dir and help, and use them to complete the tasks below. Create strings in the Python shell and do the following:

a. Check whether or not a string ends with the letter "h".
b. Check whether or not a string contains only numbers.
c. Replace every instance of the character "l" (ell) in a string with a "1" (the number one).
d. Given a string composed only of numbers, pad it with zeros so that it is exactly 6 digits long.
e. Check whether or not a string contains only lowercase letters.
f. Remove all leading zeroes from a string composed only of numbers.

Functions involving strings

This section asks you to write the code for the functions specified in the table below. First, a word about correctness: Function correctness: Before you start writing any of the code for the functions, you need to think about how you will make sure your code is correct. You have to know in your own mind, for every function, what the expected output (or return value) is for all possible values of the parameter. That way, after you've written your functions, you can check if they work as you expect.

In most cases, you can't list every possible argument value and outcome. You have to limit your prediction and checking to a relatively small set of values, so that you can do your testing in a reasonable amount of time. The best way to do this is to pick some situations that are representative of others, and also to introduce both "usual" and "unusual" situations. For example, predicting what your function will do with the empty string "" as a parameter, is an unusual test case, since this situation is often unspecified in the function docstring. A representative test case might be the string 'e' (or 'K') standing for all single-character strings.

string Function Specifications:

For this section, do not use any of Python's str methods. However, you may use **`__builtins__`** functions:

- **`longer(str, str)`** Given two strings, return the length of the longer string.
- **`earlier(str, str)`** Given two strings made up of lowercase letters, return the string that would appear earlier in the dictionary.
- **`count_letter(str, str)`** Given a string and a single-character string, count all occurrences of the second string in the first and return the count.
- **`remove_digits(str)`** Return a new string that is the same as the given string, but with digits removed.
- **`repeat_character(str, str)`** Given a string and a single-character string, return a string consisting of the second, single character string repeated as many of times as it appears in the first string.
- **`where(str, str)`** Given a string and a single-character string, return the index of the first occurrence of the second string in the first. For example, **`where("abc", "b")`** should return 1. If the second string is not in the first, return -1.

Lists

In this section, you will write short functions or statements that involve lists. In some of the exercises, you will be expected to use list methods so that you can become familiar with the tools available to you.

Remember to use the Python functions dir and help to get information about methods.

Do you remember how to slice strings? We can do the same thing with lists. If we have a list

```
names = ['Bob', 'Ho', 'Zahara', 'Amitabha', 'Dov', 'Maria']
```

then **`names[1:4]`** returns a new list equal to

```
['Ho', 'Zahara', 'Amitabha']
```

There are several list methods that you may find helpful: **`append, count, extend, index, insert, pop, remove, reverse, sort`**. Use help to learn more about them.

1. Type this assignment statement into the Python shell:

   ```
   names = ['Bob', 'Ho', 'Zahara', 'Amitabha', 'Dov', 'Maria']
   ```

   For the following steps, use names and slice notation:
   a. Write an expression that produces this new list: ['Zahara', 'Amitabha', 'Dov']
   b. Write an expression that produces this new list: ['Bob']
   c. Write an expression that produces this new list: ['Amitabha', Dov', 'Maria']
2. Given a list L and a value v, using list methods (not a loop),write an expression that removes the first occurrence of v from L.
3. Write an expression that adds the string "How are you?" to the front of the list ["I am fine, thank you"].
4. Write code that turns [2, 4, 99, 0, -3.5, 86.9, -101] into [99, 86.9, 4, 2, 0, -3.5, -101]. You should use just two method calls. (See if you can do the transformation in one statement!
5. Write a function every_third that takes a list as a parameter and returns a new list that contains every third element of the original list, starting at index 0. Do not use slice notation.
6. Write a function every_ith that takes a list L and an integer i as parameters and returns a list consisting of every ith element of L, starting at index 0. Don't use slice notation.

`while` loops

Use while loops to complete the following functions. (Do not use for loops.) Verify your work. As always, we give the type of the parameter in parentheses. Do not actually use list as the name of a parameter or variable.

```
display_list(list)
```
Print the elements of the given list.
```
display_list_even(list)
```
Print the elements of the given list that occur at even indices.
```
display_list_reverse(list)
```
Print the elements of the given list from the end of the list to the front.
```
sum_elements(list)
```
Sum the elements of the given list of ints, starting from the front of list, until the total is over 100 or the end of the list is reached, and return the sum at that point (as an int).
```
duplicates(list)
```
Return True if the given list contains adjacent elements with the same value, and return False otherwise.

Nested Lists

List elements may be lists themselves. When this happens it is called a nested list or a list of lists:

```
pets = [["Shoji", "cat", 18], ["Hanako", "dog", 15], ["Sir Toby", "cat", 10],
["Sachiko", "cat", 7], ["Sasha", "dog", 3], ["Lopez", "dog", 13]]
```

We can access each element of list pets using its index:

```
>>> pets[3]
["Sachiko", "cat", 7]
```

We can access each element of list pets using its index: We can also access elements of the inner lists. For example, since pets[3] refers to a list, we can use pets[3] to access the element at position 2:

```
>>> pets[3][2]
7
```

This is saying that element 2 of element 3 of the list pets (the age of the cat named \Sachiko") is 7.

Write the following loops and functions and call each function to verify your work. You may use for loops.

1. Write a loop that prints each list from list pets on a separate line.
2. Write a loop that prints the second element of each inner list in list pets on a separate line.
3. Write a loop that examines list pets and computes the number of dogs in the list.
4. Write a loop that examines list pets and computes the sum of the ages of the animals in the list. Ages are the third element of the inner lists.
5. Write a function nested_lengths that takes a list L as a parameter and returns a list of the lengths of the sublists. More formally: for each element e in L, the returned list contains a corresponding element, e, that represents the number of elements in e.

Text Files: the Basics

In this section, you will practice opening, reading, and writing text files. Use dir and help to get information on file methods provided by Python. Before you begin, download the text file data.txt and starter code file basics.py from the Labs section of the course website on the intranet. Open file_basics.py in Wing and implement the three functions according to their docstring descriptions.

```
# Instead of using the terms "url" or "file", we use the more general
# term "reader" to indicate an open file or webpage.

def display_lines(r):
    '''Display the lines of the file/webpage (with leading and trailing
    whitespace removed) from open reader r.'''

def display_lines_with_text(r, text):
    '''Display the lines of the file/webpage (with leading and trailing
    whitespace removed) from open reader r that contain the str text.'''

def copy_file(r):
    '''Write the lines of open reader r to the file copy.txt.'''

if __name__ == '__main__':

    data_file = open('data.txt')
    display_lines(data_file)
    data_file.close()

    data_file = open('data.txt')
    display_lines_with_text(data_file, 'ene')
    data_file.close()

    data_file = open('data.txt')
    copy_file(data_file)
    data_file.close()
```