MTA 98-381 LESSON 5 FUNCTIONS

FUNCTIONS

- A function is a collection of statements that are grouped together to perform an operation.
- Syntax:

```
def function_name (parameter1, parameter2=1):
  statements in function body
  return values
```

- A function contains a header and body.
- The header always begins with the `def` keyword, followed by function's name (compulsory), formal parameters (optional), and a colon (compulsory).
- A parameter may have be assigned a default value.
- A function may return a value using the `return` keyword.

FUNCTIONS (CONT.)

- Example (a function that returns the sum of integers from start to end):

```
def sum (start, end, step = 1):
  total = 0
  for i in range (start, end+1, step):
    total += i
  return total
```

- Example of invoking/calling `sum` function:

```
print ('sum(2,8,2) =', sum(2,8,2))
```

  - We pass 2, 8, and 2 to the formal parameters start, end, and step respectively. These values are referred to as actual parameters or arguments.

FUNCTIONS (CONT.)

- Function allows us to run a block of codes many times without having to write it many times.
- We can sum different sets of integers in one program by calling the `sum` function many times as follows:

```
sum(2,8,2)
sum(1,5)
sum(9,11)
```

- Otherwise we will have 3 sets of `for` loop to sum them.

PASSING ARGUMENTS BY KEYWORD

- By default, arguments are passed to formal parameters based on positions.
- However, Python allows us to pass arguments by specifying the keyword (formal parameters).
- For example, the following 2 lines produce same result.

```
print (sum(1,5,2))
print (sum(end=5,step=2,start=1))
```

PASS BY VALUE

- All arguments in Python are passed-by-value.
- For numbers and strings, the changes made to the formal parameters do not affect the arguments.
- Example.

```
def increment(x):
  x += 1          # Changes to parameter does not
                  # affect argument.
x = 10
print("x =", x)   # 10
increment(x)
print("x =", x)   # Still 10, why?
```

RETURN STATEMENT

- If you want to carry the changes of parameters or variables after the end of the function, return parameters or variables .
- Example.

```
def increment(x):
  x += 1
  return x        # return the new value of x.

x = 10
print("x =", x)  # 10
x = increment(x) # x receives a new value.
print("x =", x)  # 11
```

RETURNING MULTIPLE VALUES

- Example:

```
def min_first (a, b):
    if a < b:
        return a, b
    else:
        return b, a

a, b = 3, 5
a, b = min_first(a, b)
print (a, b)                # 3 5
a, b = min_first(4, 2)
print (a, b)                # 2 4
```

FUNCTION EXERCISE 1

- Write a function named `get3coorinates` that collects 3 coordinates from users, return the 3 coordinates.
- Write a function named `calculate_length` that accepts 2 coordinates, and calculates and returns the length of the 2 coordinates.
- Write a function named `calculate3lengths` that accepts 3 coordinates, use the above `calculate_length` function to obtain the 3 lengths among the 3 coordinates, return the 3 lengths.
- Write a function named `sort3numbers` that accepts 3 numbers, return the 3 numbers in ascending order.
- Print the 3 lengths in ascending order.

FUNCTION EXERCISE 2

- Write a function that returns the reverse of a string. Use 'for' loop.