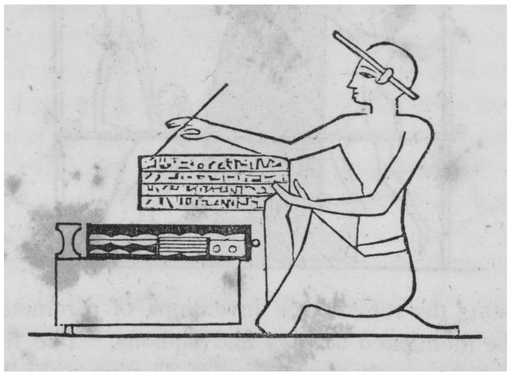

CPSC 5061—Programming I for Business

Worksheet 8

Functions

Concept

It is often convenient in programs to create little packets of code that perform a single task. These programming packets are called *functions*.

We've already seen and used functions provided by directly by Python, called *built-in functions*, and functions in libraries that we can import like `math` and `random`.

Here are some examples you should already be familiar with:

Function	Description	Example
`input`	a built-in function that prints the given prompt to the console and returns as a string whatever the user types into the console	`text = input('Name? ')`
`print`	a built-in function that converts the given data to strings and then prints them to the console ending with a newline	`print('total', total)`
`abs`	a built-in function that returns the absolute value of the given number	`magnitude = abs(distance)`
`randint`	a random library function that returns any one of the integers in the given range, inclusive, chosen at random	`secret = randint(1, 100)`
`random`	a random library function that returns a random float between 0.0 and 1.0	`guess = random() * 200 - 100`

Exercise

List several other functions you have already used:

Concept

Some functions have a single input and a single corresponding output. The input is called the *argument* and the output is called the function's *value*. These functions are like functions in mathematics and were the motivation behind them being called functions in programs, too. (In fact, there is a whole field of Computer Science that attempts to only use this pure type of function. This CS specialty is called *functional programming*.)

Concept Check

Only one of the examples given above is a pure function like this. Which one is it? Remember the criteria:

1. It takes a single argument.
2. It returns a single corresponding value for a given argument.

Evaluation Mechanism

So how does Python operate with these functions? When Python is asked to evaluate an expression that contains a function call, it carefully and exactly follows these steps:

1. **Stop and take note!** Python must take a look at everything it's doing right now and also take note of the current state of the machine. It needs to remember the value of every variable and also any other parts of the expression that it's in the middle of evaluating. Also, it must note exactly where it is in its current flow of control.
2. **Get ready to call the function!** Python then evaluates the argument to the function. The rule is that Python can only give the function one object, one datum, as the argument. So, if it is anything other than a literal value, it has to be evaluated and the result of the evaluation is then the value of the argument that will be passed into the function.
3. **Call the function!** Now Python completely pauses work on its current task and switches to the executing the function.
4. **Assign the argument to the function's parameter.** This step will be clearer in a moment when we start writing our own functions. This is where the result of determining the argument, that resulting datum, is made available to the function by assigning it to the parameter variable in the function.
5. **Execute the statements in the function.** This continues until the function states that it is done calculating its value with a `return` statement.
6. **Remember the return value.** This will be needed by the caller of the function. At the same time, Python also discards (i.e., forgets) anything to do with the function execution like any local temporary variables that were used (what we call its *context*).
7. **Return to the function caller's context.** This is why we had to remember everything as the first step! Restore all the state of the machine and now proceed. But this time Python *replaces* the function call in the expression that is being evaluated *with the returned value* from the function and uses that, instead, to finish evaluating the expression it's working on.

Example

Let's say we're executing the following line of code in Python. This is the *caller* of abs:

```
epsilon = abs(2 * a) * 1e-10
```

Python comes to this assignment statement and begins execution of it. First it needs the value of the expression on the right side of the **=**. That is a multiplication of the absolute value of *2a* and 10^{-10}. In order to accomplish that multiplication, Python first has to determine the absolute value of *2a*. For this it has a function, **abs**. But to call it and get the result and use it, it must follow the seven steps above.

1. Remember the current context: evaluating this assignment statement, just beginning to evaluate the right-hand side of the **=**, with variable **a** having the value -3.2.
2. Evaluate the argument, **2*a**, which we determine is -6.4.
3. Call the abs function. This means we tell the computer to pause working on the epsilon assignment and start executing the code for the abs function.
4. The very first thing there is to assign -6.4 to the abs function's parameter variable. Say the abs function has this code:
   ```
   def abs(n):
       if n < 0:
           n = -n
       return n
   ```
   then the variable n is assigned to -6.4.
5. The code in the abs function is evaluated until we get to **return n**.
6. The return value is +6.4 so Python remembers that but otherwise discards the context of the abs function execution.
7. Python replaces the function call with the returned value and proceeds in the original task. It's as though it now has this code:
   ```
   epsilon = 6.4 * 1e-10
   ```
   so, continues on by performing the multiplication and doing the assignment of 0.00000000064 to the variable **epsilon**.

Concept

Not all functions in programming match the pure mathematical idea of a function. Here are some of the ways they can differ:

- A function may take more than one argument (like **randint**), no arguments at all (like **random**), or even a varying number of arguments (like **print**). Python allows this by letting you separate multiple arguments with commas.
- A function may return a value that depends not just on its arguments, but also the state of the world (like **input**, **randint**, and **random**).
- A function may not return any value at all (like **print**). Python handles this by using a special placeholder value, **None**, as the return value for these functions.
- Unlike a pure function which only interacts solely via argument and return value, a programming function may also change the state of the world (like **print**).

Remember It

What functions that you have used take several arguments?

No arguments?

Varying number of arguments (try to think of at least 2)?

What functions that you have used change the state of the world?

What functions that you have used depend on the state of the world?

Defining a Function

A programmer can write functions of their own design. These are defined with the keywords `def` and `return`. The parameters are listed in the function header line enclosed in parentheses, mimicking the arrangement of the arguments in the callers. The body of the function is indented in a function body clause and ends with a return statement. So, in general, like so:

```
def function_name(parameter1, parameter2, and_so_on):
    """docstring"""
    statement2
    …
    return expression
```

An interesting note is that the function name becomes a variable just like any other. Its corresponding datum is of datatype **`function`** (add this to your datatype knowledge chart!). Because of this, the naming rules for functions are the same as those for variables. And also, you can't have a function and a variable of the same name simultaneously—this can be a tricky bug to track down, so beware!

Example Function Definition

Let's write a function that computes $3x^2 - 2x + 3$.

```
def f(x):
    y = 3 * x**2 - 2 * x + 3
    return y
```

And then let's use it (call it) in some code to produce a table of values.

```
x = -3.0
while x <= 3.0:
    print('| %6.2f | %6.2f |' % (x, f(x)))
    x += 0.25
```