Writing functions facilitates

- abstraction, by which we focus on relevant information while ignoring implementation details;
- top-down design, which breaks a task into easier subtasks, each solved by its own function;
- clarity of code, since it is easier to read a function name than the code it summarizes;
- correctness of code, since it is easy to test, debug, and optimize a short function; and
- code reuse, since we write, test, and debug a function once but call it many times.

Defining a function

A function definition has this form (here UPPERCASE indicates a placeholder):

def FUNCTION_NAME(PARAMETERS):
    """OPTIONAL_DOCUMENTATION_STRING""" # line 1: summary, 2: blank, 3+: details
    BODY

e.g.

def letterhead(): # define 'letterhead()' with no parameters
    """Prints a UW building address."""
    print('1300 University Avenue')
    print('Madison WI 53706')

def standardize(x, mu=0, sigma=1): # define 'standardize()' with 3 parameters
    """Returns difference between x and mu, in sigma's."""
    print(f'standardize(x={x}, mu={mu}, sigma={sigma})') # debugging output
    z = (x - mu) / sigma
    return z

letterhead()          # call letterhead()
standardize(x=10, mu=6, sigma=2) # call standardize()

A note on testing numeric functions

- Do not use a == b for real numbers (float in Python). e.g. \((49 \times (1 / 49)) == 1\)
- Instead, use np.isclose(a, b, rtol=1e-05, atol=1e-08) # relative, absolute tolerance
- assert CONDITION stops if CONDITION is not True, e.g. assert (0 < 1), assert (2 < 1) # error
- assert np.isclose(a=f(ARGUMENTS), b=answer) checks f(ARGUMENTS) \approx answer. e.g.
  assert np.isclose(a=standardize(x=10, mu=6, sigma=2), b=2)
Calling a function

1. Execution jumps to first line of function upon seeing the call, \texttt{FUNCTION\_NAME(ARGUMENTS)}.

2. Function’s \texttt{PARAMETERS} receive copies of \textit{references} to caller’s \texttt{ARGUMENTS}.
   
   (a) \texttt{PARAMETERS} may contain parameters with default values of the form \texttt{NAME=EXPRESSION}.
   
   (b) In the call, \textit{positional} arguments must precede \textit{keyword arguments} (\texttt{NAME=EXPRESSION}).
   
   (c) Assignment to a function’s parameters and local variables doesn’t affect caller’s variables, but a mutable object can be changed via a parameter reference.

3. Code in function is executed until \texttt{return EXPRESSION} or last line.

4. Execution returns to caller; if caller assigned a variable to function call, it gets \texttt{EXPRESSION} from function’s \texttt{return EXPRESSION} (or None if there is no \texttt{return}).
   
   (a) \texttt{return} and \texttt{print()} are different. \texttt{return} returns a value to which the caller can assign a variable. \texttt{print()} writes text but doesn’t affect the state of the program. Most functions should use \texttt{return}, not \texttt{print()}, to provide output.

Study “Calling a function” and these tiny examples (run at \url{pythontutor.com}):

```python
z = 3
print(f'{standardize(x=10, mu=6, sigma=2)}')
print(f'z={z}') # shows 2c

z = standardize(4, mu=6) # shows 2, 2a
print(f'z={z}') # z changed from "z = standardize(...)", not "z = (x - mu) / sigma"
standardize(mu=6, 4) # error; shows 2b

address = letterhead()
print(f'address={address}') # shows 4, 4a

def parameters_are_reference_copies(list1, list2):
    list1 = [0, 1, 2] # shows 2c: assignment to list1 does not change first_list,
    list2[0] = 23 # shows 2c: but can change second_list through list2

first_list = [10, 11]
second_list = [20, 21]
parameters_are_reference_copies(first_list, second_list)
print(f'first_list={first_list}, second_list={second_list}') # shows 2c
```

To learn more, see \url{Defining Functions} and \url{More on Defining Functions}.