**Defining Functions Recap**

We looked at a few examples of built-in functions in Python, but being able to define your own functions is incredibly powerful. We start a function definition with the def keyword, followed by the name we want to give our function. After the name, we have the parameters, also called arguments, for the function enclosed in parentheses. A function can have no parameters, or it can have multiple parameters. Parameters allow us to call a function and pass it data, with the data being available inside the function as variables with the same name as the parameters. Lastly, we put a colon at the end of the line.

After the colon, the function body starts. It’s important to note that in Python the function body is delimited by indentation. This means that all code indented to the right following a function definition is part of the function body. The first line that’s no longer indented is the boundary of the function body. It’s up to you how many spaces you use when indenting -- just make sure to be consistent. So if you choose to indent with four spaces, you need to use four spaces everywhere in your code.

# Returning Values Using Functions

Sometimes we don't want a function to simply run and finish. We may want a function to manipulate data we passed it and then return the result to us. This is where the concept of return values comes in handy. We use the return keyword in a function, which tells the function to pass data back. When we call the function, we can store the returned value in a variable. Return values allow our functions to be more flexible and powerful, so they can be reused and called multiple times.

Functions can even return multiple values. Just don't forget to store all returned values in variables! You could also have a function return nothing, in which case the function simply exits.

# Study Guide: Functions

This study guide provides a quick-reference summary of what you learned in this lesson and serves as a guide for the upcoming practice quiz.

In the Functions segment, you learned how to define and call functions, utilize a function’s parameters, and return data from a function. You also learned how to differentiate and convert between different data types utilizing variables. Plus, you learned a few best practices for writing reusable and readable code.

# Terms

* **return value** - the value or variable returned as the end result of a function
* **parameter (argument)** -  a value passed into a function for use within the function
* **refactoring code** - a process to restructure code without changing functionality

# Knowledge

* The purpose of the **def()** keywordis to define a new function.
* Best practices for writing code that is readable and reusable:
  + **Create a reusable function** - Replace duplicate code with one reusable function to make the code easier to read and repurpose.
  + **Refactor code** - Update code so that it is self-documenting and the intent of the code is clear.
  + **Add comments** - Adding comments is part of creating self-documenting code. Using comments allows you to leave notes to yourself and/or other programmers to make the purpose of the code clear.

# Coding skills

**Skill Group 1**

* Use a function that accepts multiple parameters
* Return a result value

# This function calculates the number of days in a variable number of

# years, months, and days. These variables are provided by the user and

# are passed to the function through the function’s parameters.

def find\_total\_days(years, months, days):

# Assign a variable to hold the calculations for the number of days in

# a year (years\*365) plus the number of days in a month (months\*30) plus

# the number of days provided through the "days" parameter variable.

    my\_days = (years\*365) + (months\*30) + days

# Use the "return" keyword to send the result of the "my\_days"

# calculation to the function call.

    return my\_days

# Function call with user provided parameter values.

print(find\_total\_days(2,5,23))

**Skill Group 2**

* Use a function to return the result of a measurement conversion
* Use arithmetic operators to perform a calculation
* Combine text with a function call within a print() statement
* Convert the return value from a float data type to a string for the print() function
* Call the function and perform a calculation on the return value within a print() statement

# This function converts fluid ounces to milliliters and returns the

# result of the conversion.

def convert\_volume(fluid\_ounce):

# Calculate value of the "ml" variable using the parameter variable

# "fluid\_ounce". There are approximately 29.5 milliliters in 1 fluid

# ounce.

    ml = fluid\_ounce \* 29.5

# Return the result of the calculation.

    return ml

# Call the conversion from within the print() function using 2 fluid

# ounces. Convert the return value from a float to a string.

print("The volume in millimeters is " + str(convert\_volume(2)))

# Call the function again and double the 2 fluid ounces from within

# the print function.

print("The volume in millimeters is " + str(convert\_volume(2)\*2))

# Alternative calculation:

# print("The volume in millimeters is " + str(convert\_volume(4))

# Python practice information

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)