Coding Basics

3.4.1

Methods, Functions, Modules, and Classes

It may seem easy to throw code in a file and make it work. But as project size and complexity grows, and as other developers (and stakeholders) get involved, disciplined methods and best practices are needed to help developers write better code and collaborate around it more easily.

One thing that most developers agree on is trying to write clean code. But, what is clean code?

3.4.2

Clean Code

Clean code is the result of developers trying to make their code easy to read and understand for other developers. What constitutes "clean code" is somewhat subjective, but here are some common principles:

- Is the formatting of the code neat, and does it follow generally-accepted formatting practices for the computer language(s) involved, and/or meet specific requirements of the institutional, project, and/or team "stylebook"?
  - Does it stick with ALL tabs or ALL spaces?
  - Does it use the same number of tabs or spaces per indentation level, throughout the file? (Some languages, such as Python, make this a requirement.)
  - Does it have the indentation in the right places?
  - Does it use consistent formatting for syntax, such as the location of curly braces ({})?

- Are variable, object, and other names used in the code intuitive?
- Is the code organized in a way that it makes sense? For example, are declarations grouped, with functions up top, mainline code at the bottom, or otherwise, depending on language and context?
- Is the code internally documented with appropriate comments?
- Does each line of code serve a purpose? Have you removed all unused code?
- Is the code written so that common code can be reused, and so that all code can be easily unit-tested?

Clean code emphasizes:

- standardized formatting and intuitive naming for ease of reading, understanding, and searching
- overall organization to communicate intention and make things easier to find
- modularity to facilitate testing and reuse
- inline and other comments
- other characteristics that help make code self-documenting

In theory, other developers should be able to understand, use, and modify your code without being able to ask you questions. This accelerates development enormously, enabling reuse, debugging, security analysis, code review and merging, along with other processes. It lets longstanding projects (for example, open source projects) incorporate your code with greater confidence. And it lets organizations keep using your code efficiently, after you have moved on to other projects or jobs.

By contrast, code that is not clean quickly becomes technical debt. It may be unmaintainable, or unusable with complete confidence. It is code that needs to be refactored, or removed and rewritten, which is expensive and time-consuming.

What are some other reasons developers want to write clean code?

1. Clean code is easier to understand, more compact, and better-organized, which tends to result in code that works correctly (fewer bugs) and performs as required.
2. Clean code, being modular, tends to be easier to test using automated methods such as unit testing frameworks.
3. Clean code, being standardized, is easier to scan and check using automated tools such as linters, or command-line tools like grep, awk, and sed.
4. It just looks nicer.

Now that you understand the goals of writing clean code, you can dive deeper into coding best practices. Specifically, looking at how to break code into methods and functions, modules, and classes.

3.4.3

Methods and Functions

Methods and functions share the same concept; they are blocks of code that perform tasks when executed. If the method or function is not executed, those tasks will not be performed. Although there are no absolute rules, here are some standard best-practices for determining whether a piece of code should be encapsulated (in a method or function):

- Code that performs a discrete task, even if it happens only once, may be a candidate for encapsulation. Classic examples include utility functions that evaluate inputs and return a logical result (for example, compare two strings for length), perform I/O operations (for example, read a file from disk), or translate data into other forms (for example, parse and process data). In these case, you encapsulate for clarity and testability, as well as for possible future re-use or extension.
- Task code that is used more than once should probably be encapsulated. If you find yourself copying several lines of code around, it probably needs to be a method or function. You can

accommodate slight variations in usage using logic to assess passed parameters (see below).

The most powerful thing about methods and functions is that they can be written once and executed as many times as you like. If used correctly, methods and functions will simplify your code, and therefore reduce the potential for bugs.

Syntax of a Function in Python:

```
# Define the function
def functionName:
  ...blocks of code...
# Call the function
functionName()
```

Arguments and Parameters

Another feature of methods and functions is the ability to execute the code based on the values of variables passed in on execution. These are called arguments. In order to use arguments when calling a method or function, the method or function needs to be written to accept these variables as parameters.

Parameters can be any data type and each parameter in a method or function can have a different data type. Arguments passed to the method or function must match the data type(s) expected by the method or function.

Depending on the coding language, some languages require the data type to be defined in the parameter (so-called 'strongly typed' languages), while some permit this optionally.

Even when parameter type specification is not required, it is usually a good idea. It makes code easier to reuse because you can more easily see what kind of parameters a method or function expects. It also makes error messages clearer. Type mismatch errors are easy to fix, whereas a wrong type passed to a function may cause errors that are difficult to understand, deeper in the code.

Parameters and arguments add flexibility to methods and functions. Sometimes the parameter is just a boolean flag that determines whether certain lines of code should be executed in the method or function. Think of parameters as being the input to the method or function.

Syntax of a function using arguments and parameters in Python:

```
# Define the function
def functionName(parameter1,...,parameterN):
  # You can use the parameters just like local variables
  ...blocks of code...
# Call the function
functionName("argument1", 4, {"argument3":"3"})
```

The example above is passing this function a string, an integer (or number), and an object containing a key and a value.

Return Statements

Methods and functions perform tasks, and can also return a value. In many languages, the return value is specified using the keyword return followed by a variable or expression. This is called the return statement. When a return statement is executed, the value of the return statement is returned and any code below it gets skipped. It is the job of the line of code calling the method or function to grab the value of the return, but it is not mandatory.

Syntax of a function with a return statement in Python:

```
# Define the function
def functionName(parameter1,...,parameterN):
  # You can use the parameters just like local variables
  ...blocks of code...
  someVariable = parameter1 * parameter2
  return someVariable
# Call the function
myVariable = functionName("argument1", 4, {"argument3":"3"})
```

In the above example, the returned value would be the string "argument1argument1argument1argument1", because Python lets you concatenate strings using the multiplication operator.

Function Example

Let's say your original code looks like this:

```
# Print the circumference for circles with a radius of 2, 5, and 7
radius1 = 2
radius2 = 5
radius3 = 7
# Formula for a circumference is c = pi * diameter
# Formula for a diameter is d = 2 * radius
pi = 3.14 # (Will hardcode pi in this example)
circumference1 = pi * radius1 * 2
print ("Circumference of a circle with a radius of " + str(radius1) + " is " +
str(circumference1))
circumference2 = pi * radius2 * 2
print ("Circumference of a circle with a radius of " + str(radius2) + " is " +
str(circumference2))
circumference3 = pi * radius3 * 2
print ("Circumference of a circle with a radius of " + str(radius3) + " is " +
str(circumference3))
```

As you can see, there is a lot of duplicate code here.

By using methods with parameters, your code can be cleaned up:

```
# Print the circumference for circles with a radius of 2, 5, and 7
# In this example, circumference and
# printCircumference are the names of
# the functions. 'radius' is a parameter to
# the function and can be used in the function.
# This function returns the value of the
# circumferenceValue to the code that called
# the function.
def circumference(radius):
  # Formula for a circumference is c = pi * diameter
    # Formula for a diameter is d = 2 * radius
  pi = 3.14 # (Will hardcode pi in this example)
  circumferenceValue = pi * radius * 2
  return circumferenceValue
def printCircumference(radius):
  myCircumference = circumference(radius)
  print ("Circumference of a circle with a radius of " + str(radius) + " is " +
str(myCircumference))
radius1 = 2
radius2 = 5
radius3 = 7
# In the below line of code, the value of radius1 (2)
# is the argument to the printCircumference function
printCircumference(radius1)
printCircumference(radius2)
printCircumference(radius3)
```

Notice that the version of code that uses functions, parameters, and arguments results in no duplicate code. Also, by using functions, you are able to label blocks of code, which make their purposes more understandable. If this example were more complicated and there were a lot of lines of code within each function, having the blocks of code duplicated three times in the file would make it much harder to understand.

Methods vs. Functions

If methods and functions share the same concept, why are they named differently? The difference between methods and functions is that functions are standalone code blocks while methods are code blocks associated with an object, typically for object-oriented programming.

Method example

The code from the function example can be modified to turn the function into a method, producing the same result:

```
# Print the circumference for circles with a radius of 2, 5, and 7
# In this example, there is a class named Circle.
# Classes will be discussed later. circumference
# and printCircumference are the names of the
# method in the class. These methods returns the
# value of the circumferenceValue to the code
# that called the method.
class Circle:
    def __init__(self, radius):
        self.radius = radius
    def circumference(self):
        # Formula for a circumference is c = pi * diameter
        # Formula for a diameter is d = 2 * radius
        pi = 3.14 # (Will hardcode pi in this example)
        circumferenceValue = pi * self.radius * 2
        return circumferenceValue
      def printCircumference(self):
        myCircumference = self.circumference()
        print ("Circumference of a circle with a radius of " + str(self.radius) + "
  is " + str(myCircumference))
  radius1 = 2
  radius2 = 5
  radius3 = 7
  # Since Circle is a class, it must be instatiated
  # with the value of the radius first.
  circle1 = Circle(radius1)
  # Since printCircumference is a method, it must be
  # called using the [class instance].[method]
  # syntax. Just calling printCircumference() will
  # not work
  circle1.printCircumference()
  circle2 = Circle(radius2)
  circle2.printCircumference()
  circle3 = Circle(radius3)
  circle3.printCircumference()
```

3.4.4

Modules

Modules are a way to build independent and self-contained chunks of code that can be reused. Developers typically use modules to divide a large project into smaller parts. This way the code is easier to read and understand, and each module can be developed in parallel without conflicts. A module is packaged as a single file. In addition to being available for integration with other modules, it should work independently.

A module consists of a set of functions and typically contains an interface for other modules to integrate with. It is essentially, a library, and cannot be instantiated.

Module example

Below is a module with a set of functions saved in a script called `circleModule.py`. You will see this script again later in the lab for this topic.

```
# Given a radius value, print the circumference of a circle.
# Formula for a circumference is c = pi * 2 * radius

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def circumference(self):
      pi = 3.14
      circumferenceValue = pi * self.radius * 2
      return circumferenceValue

    def printCircumference(self):
      myCircumference = self.circumference()
      print ("Circumference of a circle with a radius of " + str(self.radius) + "
is " + str(myCircumference))
```

An application that exists in the same directory as `circleModule.py` could use this module by importing it, instantiating the class, and then using dot notation to call its functions, as follows:

```
circle1 = Circle(2)
# Call the printCircumference for the instantiated circle1 class.
circle1.printCircumference()

# Two more instantiations and method calls for the Circle class.
circle2 = Circle(5)
circle2.printCircumference()

circle3 = Circle(7)
circle3.printCircumference()
```

3.4.5

Classes

Object-orient programming (OOP), as originally conceived, is based on some formally defined properties: encapsulation , data abstraction , polymorphism, and inheritance. In this course, using Python, you will focus on Python class structures as one manifestation of OOP.

In most OOP languages, and in Python, classes are a means of bundling data and functionality. Each class declaration defines a new object type.

Encapsulating functionality together with data storage in a single structure also accomplishes one aspect of data abstraction. Functions defined within a class are known as class methods. Classes may have class variables and object variables. As a new class object is created, new class data members and object data members (variables) are created. New classes may be defined, based on existing, previously defined classes, so that they inherit the properties, data members, and functionality (methods).

As with other Python data structures and variables, objects are instantiated (created) as they are first used, rather than being predeclared. A class may be instantiated (created) multiple times, and each with its own object-specific data attribute values. (Python classes also support class variables that are shared by all objects of a class.) Outside of the class name scope, class methods and data attributes are referenced using the dot notation: [class instance].[method name].

Note: Unlike other OOP languages, in Python, there is no means of creating 'private' class variables or internal methods. However, by convention, methods and variables with a single preceding underscore (_) are considered private and not to be used or referenced outside of the class.

3.4.6

Lab - Explore Python Classes

In this lab, you review Python methods, functions, and classes. You then create a class and instantiate it several times with different values. Finally, you review the Circle class example used in the course.

You will complete the following objectives:

- Part 1: Launch the DEVASC VM
- Part 2: Review Functions, Methods, and Classes
- Part 3: Define a Function
- Part 4: Define a Class with Methods
- Part 5: Review the circleClass.py Script

Explore Python Classes

3.3
Version Control Systems

3.5
Code Review and Testing