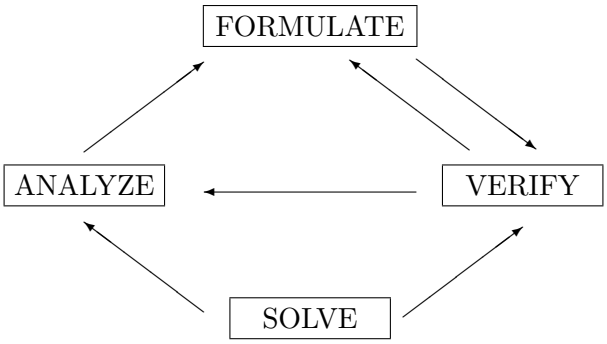
George Mason University

CDS 230

Fall 2019

Carlos Cruz



Modeling and Simulation I

Course Notes

Contents

Chapter 1

Setup

1.1 Your first program

This week, our plan is to lead you into the world of Python programming by taking you through the basic steps required to get a simple program running. The Python system (or simply Python) is a collection of applications, not unlike many of the other applications that you are accustomed to using (such as your word processor, email program, and web browser). As with any application, you need to be sure that Python is properly installed on your computer. You also need a text editor and a terminal application. By now, you should have installed the Python programming environment using the Anaconda distribution.

1.1.1 What is a Python program?

A Python program is nothing more than a sequence of characters stored in a file whose name has a *.py* extension. Python executes this sequence of statements in a specific, consistent, and predictable order. To create one, you need only define that sequence characters using a text editor.

A Python **statement** contains zero or more expressions. A statement typically has a side effect such as printing output, computing a useful value, or changing which statement is executed next.

A Python **expression** describes a computation, or operation, performed on data. For example, the arithmetic expression 2+1 describes the operation of adding 1 to 2. An expression may contain sub-expressions - the expression 2+1 contains the sub-expressions 2 and 1.

Evaluating an expression computes a Python value. This means that the Python expression 2 is different from the value 2.

The program *hello.py*, shown below, is an example of a complete Python program. The line numbers are shown to make it easy to reference specific lines, but they are not part of the program and should not be in your hello.py file.

```
1 print("Hello World!")
```

The program's sole action is to write a message back to the terminal window. A Python program consists of statements. Typically you place each statement on a distinct line.

1.1.2 Executing a Python Program

Once you compose the program, you can run (or execute) it. When you run your program the Python **compiler** translates your program into a language that is more suitable for execution on a computer[1]. Then the Python **interpreter** directs your computer to follow the instructions expressed in that language. Note that the **interpreter** is a loop[2] that:

- Reads an expression
- Evaluates the expression
- Prints the result

If the result is **None**, the interpreter does not print it. To run your program, type the python command followed by the name of the file containing the Python program in a terminal window.

```
$ python hello.py
```

For the time being, all of your programs will be just like hello.py, except with a different sequence of statements. The easiest way to compose such a program is to:

- Copy hello.py into a new file whose name is the program name followed by *.py*.
- Replace the code with a different statement or sequence of statements.

1.1.3 Python interpreter vs. Python program

Running a Python file as a program gives different results from pasting it line-by-line into the interpreter. In general the interpreter prints more output than the program would. That's because in the Python interpreter, evaluating a top-level expression prints its value while in a Python program, evaluating an expression generally does not print any output.

1.1.4 Errors

It is easy to blur the distinction among editing, compiling, and interpreting programs. You should keep them separate in your mind when you are learning to program, to better understand the effects of the errors that inevitably arise.

You can fix or avoid most errors by carefully examining the program as you create it. Some errors, known as *compile-time errors*, are raised when Python compiles the program, because they prevent the compiler from doing the translation. Python reports a compile-time error as a *SyntaxError*. Other errors, known as *run-time errors*, are not raised until Python interprets the program.

[1]Though Python is known as an interpreted language, when you run a Python **program** the source code is compiled into a much simpler form called **bytecode**. This also happens at the Python interactive prompt. However, you will never notice this compilation steps because it is implicit.

[2]An interpreter is also called a "read-eval-print loop", or a REPL

1.1.5 References

You are encouraged to visit the official Python website, `http://www.python.org`. More specifically:

- `http://docs.python.org/reference/index.html` provides information on the Python language.
- `http://docs.python.org/library/index.html` provides information on the Python standard libraries.
- `http://www.python.org/dev/peps/pep-0008/` provides information on Python programming style.

1.1.6 Programming Style

One final item that deserves some elaboration is programming style.

The overarching goal when composing code is to make it easy to understand. Understandable programs are more likely to be correct, and are more likely to stay correct as they are maintained over time.

Programmers use style guides to make programs easier to understand. The official Python style guide is given in `http://www.python.org/dev/peps/pep-0008/`. We recommend that you give the style guide a quick read now, and that your return to it occasionally as you gain more experience with composing Python programs.

Appendix A

Computational Problem Solving

Computational problem solving does not simply involve the act of computer programming. It is a process, with programming being only one of the steps. Before a program is written, a design for the program must be developed. And before a design can be developed, the problem to be solved must be well understood. Once written, the program must be thoroughly tested. These steps are outlined below.

```
ANALYSIS
                    Clearly understand the problem
                    Know what constitutes a solution
```

```
DESIGN
                    Determine what type of data is needed
                    Determine how the data is to be structured
                    Find another design appropriate algorithm
```

```
IMPLEMENTATION
                    Represent data within programming language
                    Implement algorithms in programming language
```

```
TESTING
                    Test the program on a selected set of problem instances
                    Correct and understand the causes of any errors found
```

1. ANALYSIS

   (a) Understanding the problem. Once a problem is clearly understood, the fundamental computational issues for solving it can be determined.

   (b) Knowing what constitutes a solution. For some problems, there is only one solution. For others, there may be a number (or infinite number) of solutions. Thus, a program may be stated as finding

- A solution
- An approximate solution
- A best solution
- All solutions

2. DESIGN

   (a) Describing the data needed. This, of course, depends on the problem at hand. We can use a list, a table, a matrix, etc.

   (b) Describing the Needed Algorithms. For some problems, there is only one solution. When solving a computational problem, either suitable existing algorithms may be found or new algorithms must be developed. Algorithms that work well in general but are not guaranteed to give the correct result for each specific problem are called *heuristic algorithms*.

3. IMPLEMENTATION Design decisions provide general details of the data representation and the algorithmic approaches for solving a problem. The details, however, do not specify which programming language to use, or how to implement the program. That is a decision for the implementation phase. Since we are programming in Python, the implementation needs to be expressed in a syntactically correct and appropriate way, using the instructions and features available in Python.

4. TESTING Software testing is a crucial part of software development. Testing is done incrementally as a program is being developed, when the program is complete, and when the program needs to be updated.

Appendix B

References

Tutorials

Tutorials for beginners:
`https://www.w3schools.com/PYTHON/python_lists.asp`
`https://www.tutorialspoint.com/python/`

A Python tutorial from the official Python website:
`https://docs.python.org/3/tutorial//`

For exact syntax and semantics of the Python language:
`http://docs.python.org/3/`

Reference manual of the standard library:
`http://devdocs.io/python`

Online Tools

The Python Online Tutor allows you to visualize execution of Python code.
`http://people.csail.mit.edu/pgbovine/python/tutor.html`

Online Python interpreter: Just in case Jupyter Notebook is not enough.
`https://www.onlinegdb.com/online_python_interpreter`

Modeling and Simulation

This book has material similar in spirit to CDS230 but slightly different approach:
`http://greenteapress.com/wp/modsimpy`