The Practice of Computing Using

PYTHON

William Punch

Richard Enbody

Chapter 1

Beginnings

Addison-Wesley
is an imprint of
PEARSON

Our Goals

- Our goals are not to just write copious amounts of code, our goals are to:
- increase our problem solving skills
- design good solutions to problems
- test somehow how well they are indeed solutions to the problem
- provide the solution as a readable document

An Analogy

Let us say that you have signed up to study French poetry (how about Marot) in the original language.

You have two problems:

- you don’t speak French
- you don’t know much about poetry

Clement Marot 1496-1544

- *Ma mignonne*
- *Je vous donne*
- *Le bon jour;*
- *Le séjuour*
- *C'est prison.*
- *Guérison*
- *Recouvrez,*
- *Puis ouvrez*
- *Votre porte*
- *Et qu'on sorte*
- *Vitement,*
- *Car Clément*
- *Le vous mande.*
- *Va, friane*
- *De ta bouche,*
- *Qui se couche*
- *En danger*
- *Pour mange*
- *Confitures;*
- *Si tu dures*
- *Trop malade,*
- *Couleur fade*
- *Tu Prendras,*
- *Et perdras*
- *L'embonpoint.*
- *Dieu te doint*
- *Santé bonne,*
- *Ma mignonne.*

Crappy-Literal Translation

- *My sweet/cute [one] (feminine)*
- *I [to] you (respectful) give/bid/convey*
- *The good day (i.e., a hello, i.e., greetings).*
- *The stay/sojourn/visit (i.e., quarantine)*
- *[It] is prison.*
- *Cure/recovery/healing (i.e., [good] health)*
- *Recover (respectful imperative),*
- *[And] then open (respectful imperative)*
- *Your (respectful) door,*
- *And [that one (i.e., you (respectful)) should} go out*
- *Fast[ly]/quick[ly]/rapid[ly],*
- *For/because Clement*
- *It (i.e., thusly) [to] you(respectful) commands/orders.*
- *Go (familiar imperative), fond-one/enjoyer/partaker*
- *Of your (familiar) mouth,*
- *Who/which herself/himself/itself beds (i.e., lies down)*
- *In danger;*
- *For/in-order-to eat*
- *Jams/jellies/confectionery.*
- *If you (familiar) last (i.e., stay/remain)*
- *Too sick/ill,*
- *[A] color pale/faded/dull*
- *You)familiar) will take [on],*
- *And [you (familiar)] will waste/lose*
- *The plumpness/stoutness/portliness (i.e., well-fed look).*
- *[may] God [to] you (familiar) give/grant*
- *Health good,*
- *My sweet/cute [one] (feminine).*

Decent Trans, S.Jamar

- *My sweet dish,*
- *You I wish*
- *A good day.*
- *Where you stay,*
- *Is a jail.*
- *Though so pale,*
- *Leave your bed,*
- *Regain red.*
- *Ope' your door*
- *Stay not, poor*
- *Child; gain strength*
- *And at length,*
- *Steve does urge,*
- *Please emerge.*
- *Then go eat*
- *Jam so sweet.*
- *Lying ill*
- *Means you will*
- *become too thin -*
- *Merely skin*
- *Cov'ring bone;*
- *Regretted tone.*
- *Eat again,*
- *Avoid the fen.*
- *God grant thee*
- *Be healthy.*
- *This I wish,*
- *My sweet dish.*

Why is this so hard?

You have two related problems:

- the “syntax” of French is something you have to learn
- the “semantics” of poetry is something you have to learn

You have two problems you have to solve at the same time.

Programming, Syntax and Semantics

- You have to learn the “syntax” of a particular programming language
  - many details about the language, how to debug and use it
- You have to learn about “problem solving” and how to put it down on “computer.”
- There probably is no better way. It’s hard!

Computers & Problem Solving?

This is both the promise and difficulty of computers.

- The promise (perhaps the hope) of computers is that, somehow, we can embed our own thoughts in them. To some extent we can!
- The problem is the difficulty of doing so, and the stringent requirements, the real rigor, required to put simple “thoughts” into a working program.

Good Programs

What makes a good program?

- A program is a reflection of the writer and their thoughts
- First, you must have some thoughts! The difficulty for most people is to figure out what has to be done, the problem solving, before writing a program

Rule 1

- Think before you program!

Rule 2

- Think before you program!

Rule 3

- A program is a human-readable essay on solving a problem that also happens to execute on a computer.

Rule 4

- The best way to improve your programming and problem solving skills is to practice.

Why Python?

The book utilizes the programming language known as Python.

Why?

Why Python (1): Simpler

- Python is a “simpler” language than C++
- Simpler means:
  - Fewer alternatives (one way to do it)
  - Better alternatives (easier to accomplish common tasks)
- This allows us to focus less on the language and more on problem solving

Why Python(2): Interactive

- C++ requires an intermediate step before you can run a program, compiling.
- Python allows you to type program statements into the python window and *see results immediately*
- Better for experimenting (which you *need* to do)

Why Python(3): User Base

- While we want to (and will) teach the fundamentals of computer science, we want what you learn to be “useful”
- Python is used in many areas to solve problems related to that field. Many packages are available to help for a particular area

Why Python (4): Useful

- C++ is a good language, especially for majors. It teaches a level of detail that is needed
- Python is more generally "useful", you can do things with it quickly. If you only take this course in CS, you will learn something fundamental *and* practical.

Computational Thinking

Having finished this course, we want you to have the following thought in your subsequent college career.

“Hey, I’ll just write a program for that”.

Python allows this to happen more readily.

Is Python the Best Language?

- The answer is no. This is because there is no “best” languages.
- Computer languages, like tools, are suited for different tasks (What’s the best shovel? Depends on what you are doing).
- For introductory students, we think Python is a very good language.

What is a Computer Program?

Programs

- A program is a sequence of instructions.
- To *run* a program is to:
  - create the sequence of instructions according to your design and the language rules
  - turn that program into the binary commands the processor understands
  - give the binary code to the OS, so it can give it to the processor
  - OS tells the processor to run the program
  - when finished (or it dies :-), OS cleans up.

Interpreted

- Python is an *interpreted* language
- interpreted means that Python looks at each instruction, one at a time, and turns that instruction into something that can be run.
- That means that you can simply open the Python interpreter and enter instructions one-at-a-time.
- You can also *import* a program which causes the instructions in the program to be executed, as if you had typed them in.
- To rerun an imported program you *reload* it.

Your First Python Program

```
print "Hello World!"
```

Program to Calculate Circumference and Area

- First, need to prompt the user for a radius
- Then apply the circumference and area formulas
- Finally, print the results

Code Listing 1.1

```
# 1. prompt user for the radius
# 2. apply the circumference and area formulas
# 3. print the results
import math
radiusString = raw_input("Enter the radius of your circle: ")
radiusFloat = float(radiusString)
circumference = 2*math.pi*radiusFloat
area = math.pi*radiusFloat*radiusFloat

print
print "The cirumference of your circle is:", circumference,\
      "and the area is:", area
```

Getting Input

The function:

```
raw_input("Give me a value")
```

- prints “Give me a value” on the python screen and waits till the user types something (anything), ending with Enter
- Warning, it returns a string (sequence of characters), no matter what is given, even a number (‘1’ is not the same as 1, different types)

Assignment

The = sign is the assignment statement

```
circumference = 2 * math.pi *
  radiusFloat
```

- The value on the right is associated with the variable name on the left
- It does ***not*** stand for equality!
- More on this later

Conversion

Convert from string to integer

```
radiusString = raw_input("Enter
  the radius of your circle:")
radiusFloat =
  float(radiusString)
```

- Python requires that you must convert a sequence of characters to a number
- Once converted, we can do math on the numbers

Import of Math

- One thing we did was to import the math module with `import math`
- This brought in python statements to support math (try it in the python window)
- We precede all operations of math with `math.xxx`
- `math.pi`, for example, is pi. `math.pow(x,y)` raises `x` to the yth power.

Printing Output

```
myVar = 12
print "My var has a value of:",myVar
```

- `print` takes a list of elements to print, separated by commas
  - if the element is a literal, prints it as is
  - if the element is a variable, prints the value associated with the variable
  - after printing, moves on to a new line of output

Save as a “Module”

- When you save a file, such as our first program, and place a .py suffix on it, it becomes a python module
- You “run” the module from the IDLE menu to see the results of the operation
- A module is just a file of python commands

Common Issue

- Using IDLE, if you save the file without a .py suffix, it will stop colorizing and formatting the file.
- Resave with the .py, everything is fine

Errors

- If there are interpreter errors, Python cannot run your code because the code is somehow malformed
- You can then modify and rerun the program again until there are no errors

Whitespace

- white space are characters that don’t print (blanks, tabs, carriage returns etc.)
- For the most part, you can place “white space” (spaces) anywhere in your program
- use it to make a program more readable

Continuation

- However, python is sensitive to end of line stuff. To make a line continue, use the \

```
print "this is a test", \
"of continuation"
```

prints

```
this is a test of continuation
```

Also, Indenting is a Special

- The use of indentation is also something that Python is sensitive to.
- We'll see more of that when we get to control, but be aware that indentation has meaning to Python

Python Comments

- A comment begins with a “#”
- This means that from the “#” to the end of that line, nothing will be interpreted by Python.
- You can write information that will help the reader with the code

Python Syntax

- Let’s look at the syntax stuff
- We’ll pick more up as we go along

Python Keywords

You cannot use (are prevented from using) them in a variable name

and
as
assert
break
class
continue
def
del
elif
else
except
exec
finally
for
from
global
if
import
in
is
lambda
not
or
pass
print
raise
return
try
while
with
yield

Python Operators

Reserved operators in Python (expressions)

+	-	*	**	/	//	%
<<	>>	&	|	^	~	
<	>	<=	>=	==	!=	<>

Python Punctuators

- Python punctuation/delimiters ($ and ? not allowed).

‘ “ # \

( ) [ ] { } @

, : . ` = ;

+= -= *= /= //= %=

&= |= ^= >>= <<= **=

Literals

- Literal is a programming notation for a fixed value.
- For example, 123 is a fixed value, an integer
  - it would be “weird” if the symbol 123’s value could change to be 3.14!

Math Operators

- Integer
  - addition and subtraction: `+, -`
  - multiplication: `*`
  - division
    - quotient: `/`
    - remainder: `%`
- Floating point
  - add, subtract, multiply, divide: `+, -, *, /`

Variables

- A variable is a name we designate to represent “something” in our program
- A variable references a location in memory
- We use names to make our program more readable, so that the “something” is easily understood

Variable Pairs

- Python maintains a list of pairs for every variable:
  - variable’s name
  - variable’s value
- A variable is created when a value is assigned the first time. It associates a name and a value
- subsequent assignments update the associated value.
- A variable’s type depends on what is assigned.

X = 7 →

Name	Value
X	7

Namespace

- A namespace is the table that contains all variable pairs.
- We will see more about namespaces as we get further into Python, but it is an essential part of the language.

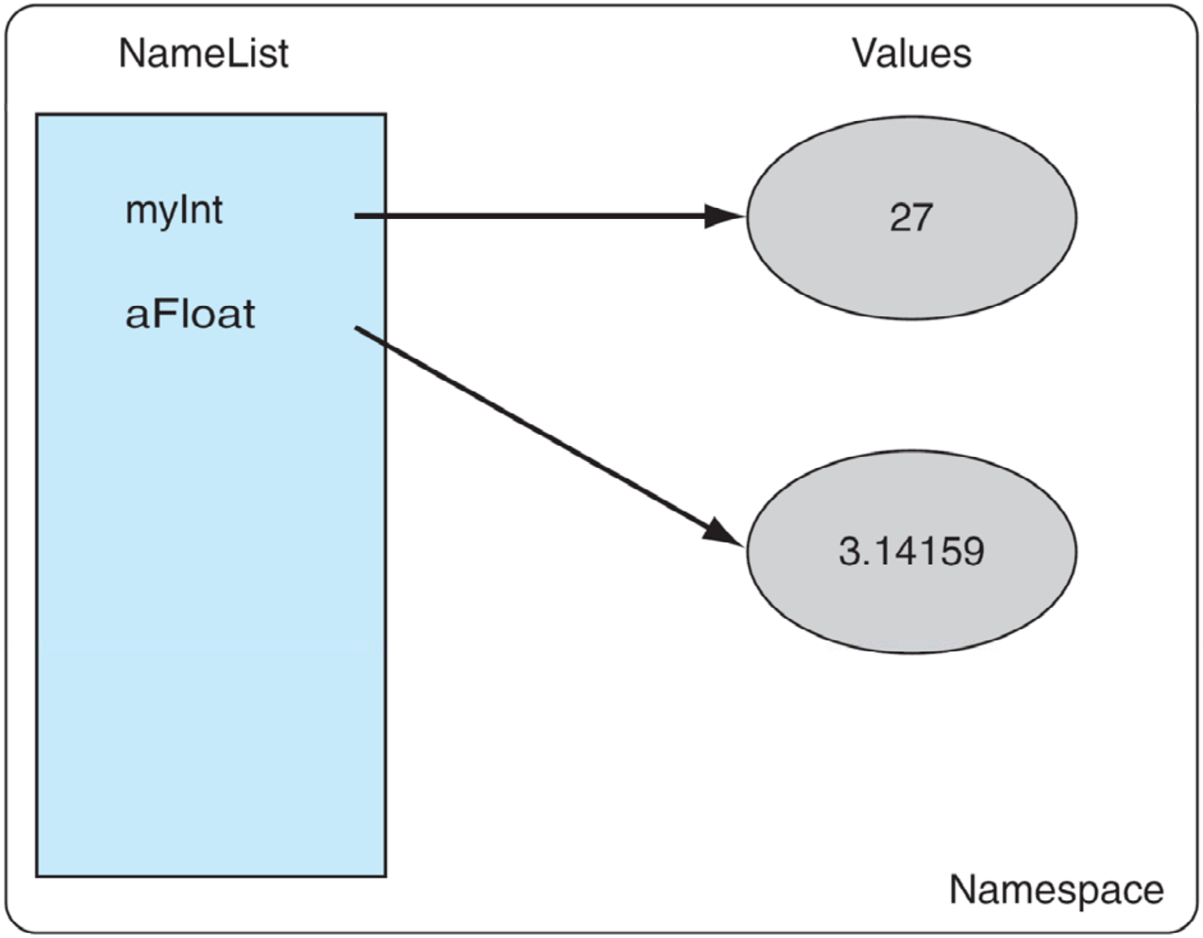

FIGURE 1.1 Namespace containing variable names and associated values.

Python Name Conventions

- must begin with a letter or _
  - **Ab123** is OK, but **123ABC** is not.
- may contain letters, digits, and underscores
  - **this_is_an_identifier_123**
- may be of any length
- upper and lower case letters are different
  - **LengthOfRope** is not **lengthofrope**
- names starting with _ have special meaning. Be careful

Variables and Types

- Python does not require you to pre-define the type of a variable
- What type a variable holds can change
- Nonetheless, knowing the type can be important for using the correct operation on a variable. Thus proper naming is important!

Python “Types”

- integers: **5**
- floats: **1.2**
- booleans: **True**
- strings: "anything" or "something"
- lists: [,]: ["a",1,1.3]
- others we will see

What is a Type?

- a type in Python essentially defines two things:
  - – the internal structure of the type (what is contains)
  - – the kinds of operations you can perform
- "abc".capitalize() is a method you can call on strings, but not integers
- some types have multiple elements (collections), we’ll see those later

Fundamental Types

- Integers
  - **1, -27 (to +/- 2^{31} – 1)**
  - **123L L suffix means any length, but potentially very slow. Python will convert if an integer gets too long automatically**
- Floating Point (Real)
  - **3.14, 10., .001, 3.14e-10, 0e0**
- Booleans (True or False values)
  - **True, False note the capital**

Converting Types

- A character `"1"` is not an integer 1.
- You need to convert the value returned by the `raw_input` command (characters) into an integer
- `int("123")` yields the integer `123`

Type Conversion

- int(someVar) converts to an integer
- float(someVar) converts to a float
- str(someVar) converts to a string
- should check out what works:
  - int(2.1) → 2, int("2") → 2, but int("2.1") fails
  - float(2) → 2.0, float("2.0") → 2.0, float("2") → 2.0, float(2.0) → 2.0
  - str(2) → "2", str(2.0) → "2.0", str("a") → "a"

Types and Division

Python does binary operations on two values of the same type, yielding a value of that type:

- 2/3, integer types, yield integer (0).
  - 2%3 is the remainder, an integer (2)
- 2.0/3.0, float types, yield float (0.66666)

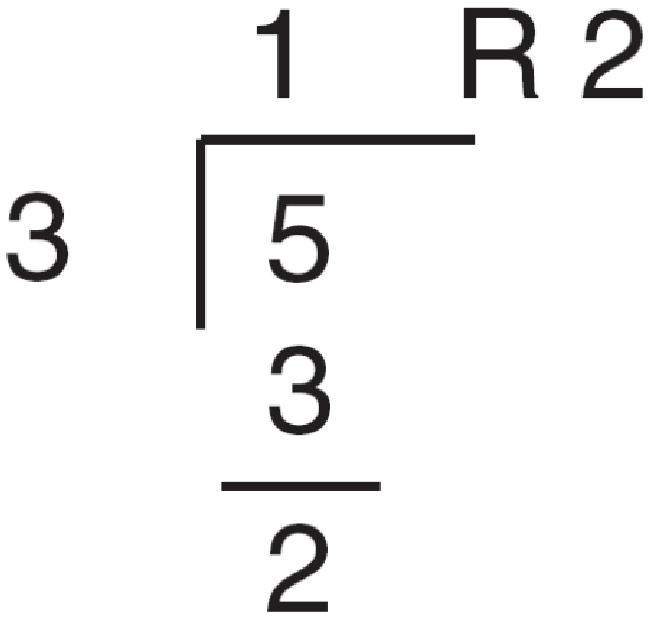

FIGURE 1.3 Long division example.

Mixed Types

- You know that 4/3 is 1 (integer division)
- You know that 4.0/3.0 is 1.3333333 (float)
- What is 4/3.0?
  - no mixed type operations. Must convert
  - Python will automatically convert to the most detailed result. Thus 4 → 4.0, the result is 1.3333333

Collections (Data Structures)

- lists
  - – sequence of any data elements
- dictionary
  - – a list of name:value pairs. Very powerful!
- Class (defines an object when instantiated)
  - – a user-defined data type

Develop an Algorithm

How do we solve the following?

- If one inch of rain falls on an acre of land, how many gallons of water have accumulated on that acre?

Develop an Algorithm

Need to know:

- How many square inches per acre?
- How many cubic inches per gallon?

Then we can compute:

gallons = squareInchesPerAcre*inches/
cubicInchesPerGallon