Computer Science 1000: Part #7

Programming in Python

PROGRAMMING LANGUAGES: AN OVERVIEW
THE PYTHON PROGRAMMING LANGUAGE
IMPLEMENTING PROGRAMMING
Programming Languages: An Overview

- Disadvantages of assembly language:
  1. Low-level / concrete conception of data, e.g., numbers, registers $\leftrightarrow$ memory.
  2. Low-level / concrete conception of task, e.g., ADD, COMPARE, JUMP
  4. Not like natural language.

- Advantages of high-level programming language:
  1. High-level / abstract conception of data, e.g., lists, data item $\leftrightarrow$ data item.
  2. High-level / abstract conception of task, e.g., IF-THEN-ELSE, WHILE loop.
  4. Like natural language.
A programming language is defined by the valid statements in that language (syntax) and what those statements do (semantics).

A programming language can be compiled (whole program translated into machine language) or interpreted (individual program-statements translated as needed).

Machine-independence achieved formally by standards, e.g., ANSI, IEEE, and implemented in practice by intermediate languages, e.g., bytecode.

Machine-independence is often violated, e.g., may exploit particular machines and/or modify available language features; additional incompatible variants may arise as language evolves over time.
Figure 8.1
Transitions of a High-level Language Program
Programming Languages: An Overview (Cont’d)

Two reasons why there are many programming languages:

1. Languages are designed for different tasks, e.g.,
   - Scientific computation (FORTRAN)
   - Business applications (COBOL)
   - Web-page creation (HTML)
   - Database creation (SQL)

2. Languages are designed for different ways of thinking about programming, e.g.,
   - Procedural programming (FORTRAN, COBOL, C)
   - Object-oriented programming (OOP) (C++, Java)
   - Logic Programming (Prolog)
   - Script-based programming (Javascript, Ruby)
The Python Programming Language: Overview

- Created by Guido van Rossum in 1991 as an easy-to-learn general-purpose programming language.
- Procedural scripting language that allows but does not require OOP (“as OOP as you wanna be”).
- Key design principles:
  - Control structure indicated by indentation.
  - Powerful built-in data types.
  - Any variable can refer to any type of data, and this type can change as a program executes.
- Primarily interpreted but can be compiled for speed.
- General machine-independence achieved by bytecode; however, Python 3.x not directly backward-compatible with Python 2.x.
The Python Programming Language:
A First Example Program

1. # Example program; adapted from
2. # Online Python Supplement, Figure 1.2

4. speed = input("Enter speed (mph): ")
5. speed = int(speed)
6. distance = input("Enter distance (miles): ")
7. distance = float(distance)

9. time = distance / speed
10.
11. print("At", speed, "mph, it will take")
12. print(time, "hours to travel", \
13.     distance, "miles.")
The Python Programming Language:
A First Example Program (Cont’d)

- Python programs are stored in files with extension .py, e.g., example1.py.
- When this program is executed using a Python interpreter and the user enters the boldfaced values, this is printed:

```
Enter speed (mph):  58
Enter distance (miles):  657.5
At 58 mph it will take
11.3362068966 hours to travel 657.5 miles.
```
The Python Programming Language:
A First Example Program (Cont’d)

- Line numbers not necessary; are given here to allow easy reference to program lines.
- Lines beginning with hash (#) are **comments** (Lines 1-2); a **prologue comment** at the top of the program gives a program’s purpose and creation / modification history.
- Comment and blank lines (Lines 3, 8, and 10) are ignored.
- Each line is a program statement; multiline statements are linked by end-of-line backslashes (\) (Lines 12-13).
- No variable-type declaration statements; this is handled by **assignment statements** (Lines 4-7 and 9).
- This program also has basic **I/O statements** (Lines 4, 6, and 11-13); **control statements** will be shown later.
The Python Programming Language: Assignment Statements

- General form: `variable = expression`, e.g.,
  
  - `index = 1`
  - `myDistanceRate = curDistanceRate * 1.75`
  - `name = "Todd Wareham"`
  - `curDataFilename = main + ".txt"`
  - `callList = ["Bob", "Sue", "Anne"]`

- Sets the value of `variable` to the value of `expression`.
  
  - If `variable` did not already exist, it is created.
  - If `variable` did already exist, its previous value is replaced. Note that the data-type of this previous value need not be that of the value created by `expression`. 
The Python Programming Language: Assignment Statements (Cont’d)

- Variable names (also called **identifiers**) can be arbitrary sequences of letters, numbers and underscore symbols (_) such that (1) the first symbol is a letter and (2) the sequence is not already used in the Python language, e.g., `if`, `while`.

- Python is case-sensitive wrt letter capitalization, e.g., `myList` is a different variable than `mylist`.

- By convention, variables are a mix of lower- and upper-case letters and numbers; words may be combined to form a variable name in so-called “camel-style”, e.g., `myList`, `dataFilename1`. 
The Python Programming Language: Assignment Statements (Cont’d)

• By convention, constants use only upper-case letters and numbers, e.g., \( \pi \), TYPE1COLOR.
  
• Though constants should not change value, they are still technically variables, e.g.,

  \[
  \ldots
  \]

  \[
  \pi = 3.1415927
  \]

  \[
  \ldots
  \]

  \[
  \pi = -1
  \]

  \[
  \ldots
  \]

  It is up to programmers to make sure that such changes do not happen.

• Underscores reserved for Python system constants.
• The **int** and **float** data-types

• Encode “arbitrary” integers, e.g., \(-1001, 0, 57\), and floating-point numbers, e.g. \(-100.2, 3.1415927\).

• Supports basic arithmetic operations (**+, **-, ***, /**); also has floor-division (**//**) and remainder (**%**) operations, e.g.,

\[
\begin{align*}
7 & \div 2 \implies 3.5 \\
7 & \div \div 2 \implies 3 \\
7 & \% 2 \implies 1
\end{align*}
\]

Behaviour of **/** incompatible with Python 2.x.

• Many additional math functions and constants available in the **math** module, e.g., **pow**(base, exponent), **abs**(x), **sqrt**(x), pi.
The Python Programming Language:
Assignment Statements (Cont’d)

radius = input("Enter radius: ")
radius = float(radius)
area = 3.1415927 * radius * radius
print("Circle Area = ", area)

import math

radius = input("Enter radius: ")
radius = float(radius)
area = math.pi * math.pow(radius, 2)
print("Circle Area = ", area)
The Python Programming Language:
Assignment Statements (Cont’d)

- The `str` data-type
  - Encodes “arbitrary” character strings, e.g., "657.5", "Todd Wareham".
  - Supports many operations, e.g.,
    - Concatenation (+) ("Todd" + " " + "Wareham" \(\Rightarrow\) "Todd Wareham")
    - Lower-casing ("Todd".lower( \(\Rightarrow\) “todd”)
    - Upper-casing ("Todd".upper( \(\Rightarrow\) “TODD”)
  - Convert between data types using **type casting** functions, e.g.,
    - `float("657.5") \(\Rightarrow\) 657.5`, `int(657.5) \(\Rightarrow\) 657`, `str(58) \(\Rightarrow\) "58".`
The list data-type

- Encodes “arbitrary” lists, e.g., [22, 5, 13, 57, -1], ["Bob", "Sue", "Anne"].
- Items in list L indexed from 0 as L[IND], e.g., if L = [22, 5, 13, 57, -1], L[0] ➞ 22 and L[4] ➞ -1.
- Supports many operations, e.g.,
  - Append x to right end of list (L.append(x))
  - List sorting (L.sort())
  - Get list maximum value (max(L))
The Python Programming Language: I/O Statements

- Keyboard input done via `input(string)`.  
  - Prints `string` on screen, waits for user to enter input followed by a key return, and then returns this input-string.  
  - Input-string can be converted as necessary by type-casting functions, e.g., `float(radius)`.

- Screen output done via `print(plist)`.  
  - Comma-separated items in `plist` converted to strings as necessary and concatenated, and resulting string printed.  
  - By default, each `print`-statement prints one line; can override this by making `end = " "`) the last item.  
  - Can include escape characters to modify printout, e.g., `\t (tab), \n (newline)`,

- Above I/O incompatible with Python 2.x.
The statements

print("Here is \t a weird")
print("way \n of printing ", end = " ")
print("this message.")

print out

Here is a weird way of printing this message.
The Python Programming Language: A First Example Program Redux

1. # Example program; adapted from
2. # Online Python Supplement, Figure 1.2
3.
4. speed = input("Enter speed (mph): ")
5. speed = int(speed)
6. distance = input("Enter distance (miles): ")
7. distance = float(distance)
8.
9. time = distance / speed
10.
11. print("At", speed, "mph, it will take")
12. print(time, "hours to travel", \
13. distance, "miles.")
Implementing Programming: The Software Crisis

• Act of programming made easier by compilers, languages, and operating systems; problem of developing algorithms remained.

• Special notations like flowcharts help with small- and medium-size programs; hope was that appropriate management would help with large ones.
Implementing Programming:
The Software Crisis (Cont’d)

IBM System/360 (1967)
Fred Brooks Jr. (1931–)

- OS/360 initially planned for 1965 costing $125M; limped to market in 1967 costing $500M, and virtually destroyed IBM’s in-house programming division.
- Brooks discussed causes in *The Mythical Man Month*. 
As both larger programs and larger teams have more complex internal relationships, adding more programmers to larger projects makes things worse.
Software Engineering born at 1968 NATO-sponsored conference; goal of SE is to develop efficient processes for creating and maintaining correct software systems.

Many types of processes proposed, e.g., design and management methodologies (Agile), automatic software derivation methods; however, “No Silver Bullet” (Brooks).
... And If You Liked This...

- MUN Computer Science courses on this area:
  - COMP 1001: Introduction to Programming
  - COMP 2001: Object-oriented Programming and HCI
  - COMP 2005: Software Engineering

- MUN Computer Science professors teaching courses / doing research in this area:
  - Ed Brown
  - Rod Byrne
  - Adrian Fiech