DISCUSSION NOTES 01
ECS 10, WINTER 2008

SAMUEL JOHNSON

Contents

1. Installing Python 1
1.1. Beginner Windows and Macintosh 1
1.2. Advanced Macintosh and Linux 2
2. Using *IDLE* 2
2.1. The top-level 2
2.2. Editing source code 3
3. Variables 4
3.1. Variable basics 4
3.2. Converting between types using **int()**, **float()**, and **str()** 5
4. Simple operators 6
4.1. Mathematical operators for integers and floats 6
4.2. Operators on strings 7
5. Summary 7
6. About these notes 8

1. Installing Python

In this section we describe how to install the Python environment onto a Windows PC and a Macintosh. For advanced Macintosh and Linux users we show how to invoke python from a terminal session. Most students will only be interested in using Python via the *IDLE* application and can safely ignore the Advanced Macintosh and Linux section below. For this course we will assume that the student is using *IDLE*.

1.1. **Beginner Windows and Macintosh.** Python is available for download free of charge from the Python website at http://www.python.org/download/. From this page you can download the appropriate installer package for your system (probably either the Windows installer or the Macintosh OS X universal installer). Once the download is completed simply open the installer and follow the directions given.

To invoke a Python session in Windows simply click on the Start button, expand the Python folder and click on *IDLE (Python GUI)*. In Mac OS X look in the /Applications/ directory for the MacPython folder. In this folder double click on *IDLE*.

1.2. Advanced Macintosh and Linux.

Macintosh OS X and most Linux distributions have a Python run-time environment pre-installed. To invoke the run-time environment simply type **python** in a terminal session. To open a terminal in Macintosh OS X, double click on *Terminal* located in /Applications/Utilities/. In most Linux distributions you can launch a terminal session from somewhere in the tool bar.

If you choose to use Python in this fashion you must write your programs using a plain text editor, examples of which include *emacs* and *vi*. To execute your program from the command line type **python myprog.py** where myprog.py is the name of your program's source code file. Alternatively you can include the line

```
#! /usr/bin/env python
```

at the top of the source file and give the file "execution" permissions (from the command line type **chmod +x myprog.py**) and then you can execute your program by typing **/path/to/myprog.py** in the terminal (where /path/to/ is the path to myprog.py in the filesystem).

2. Using *IDLE*

In this section we will introduce the major components of *IDLE* and interact with the top-level. Then we will focus on how to write commands in a source file and then run that file.

2.1. The top-level.

The first screen that comes up once you launch *IDLE* is the top-level. In the top-level you can interact with Python directly. (Sometimes this is referred to as "interactive mode" rather than the "top-level".) You will notice a prompt that looks like

```
>>>
```

at which you can type commands for Python to evaluate immediately. If the command involves a **print** statement then Python will print the results on the following line and then another prompt.. If the command is an assignment or something that does not produce output then the following line will simply be another prompt.

Consider the following simple example:

```
>>> number_of_cats=22
>>> owners_name="Peggy"
>>> print owners_name, "has", number_of_cats, "cats."
Peggy has 22 cats.
>>>
```

In this example we first define two variables, **number_of_cats** and **owners_name**, and assign values to them with the assignment operator, =. The third line uses the **print** function then prints out the message ”Peggy has 22 cats.”. Notice that the print function printed the contents of the variables **number_of_cats** and **owners_name** rather than the names of the variables. Instead, the variables were first evaluated and then their value was printed. Variables will be discussed in more detail below, but first we will turn our attention to creating and editing source files in *IDLE*.

2.2. **Editing source code.** You will rarely be writing programs directly in the top-level because it does not provide a mechanism with which to save the program. Instead you will be using a plain-text editor (or programmer's text editor). Such an editor is available in *IDLE*. To get to it select File → New Window from the menu bar. This will open a blank window in which you can write your Python source code. To save the source code you have written simply go to File → Save As.. just as you would save a document in a word processor or photo in a image editing program. By convention, Python source code files end with **.py** extensions and *IDLE* does not add this extension automatically. So you will want to save your programs with names like **myProgram.py** (with the **.py** extension)[1]. To run the program that you are writing you use the Run → Run Module option from the menu bar. Alternatively, you can press F5 on the keyboard.

Lets write a simple example program.

```
name = raw_input("Enter your name: ")
age = raw_input("Enter your age: ")
print "Hi", name, ". You claim to be", age, "years old."
```

When running this program it will pause at each line that calls the function **raw_input()** where it waits for the user to type something. It will proceed once the user presses the enter key on the keyboard. A typical run of this program will look something like this:

[1]The notation whereYouWriteLikeThis is called ”camel hump” notation and is frequently used by programmers since white space (i.e. spaces) are not allowed in function names or variable names. Another common way around using white space is to separate_words_with_underscore_characters.

```
>>> =========================== RESTART ===========================
>>>
Enter your name: Rob
Enter your age: 45
Hi Rob . You claim to be 45 years old.
>>>
```

2.2.1. *Using* **raw_input()**. The **raw_input()** function will be used often throughout this course and it is important to understand how to use it properly. **raw_input()** takes a single parameter, a string, which it prints. Then it waits for the user to type in something. (The user must press enter when they are finished or else it will continue to wait indefinitely.) Then **raw_input()** returns what ever the user typed **as a string**. Below is an example of its use:

```
>>> m = raw_input("Enter your name: ")
Enter your name: Sam
>>> p = raw_input("What is your last name, " + m + "? ")
What is your last name, Sam? Johnson
>>> print "Hello", m, p
Hello Sam Johnson
>>>
```

In this example the user (me) entered **Sam** and then **Johnson** at the first and second **raw_input()** prompts, respectively.

3. Variables

The objective of this section is to explain what variables are and some of their properties.

3.1. **Variable basics.** Variables are containers that hold data. There are three components to a variables:

- The *name* of the variables; how it is referred to.
- The *type* of data that the variable contains. Some basic types are *strings*[2], *floats*[3], and *integers*[4].

[2]A string is a sequence of characters. This includes alphabetical, numerical, and punctuation (and others, refer to an ASCII table).

[3]A float, or "floating point number" is a representation of a real number and can be recognized as containing a decimal point.

[4]An integer is an integer in the mathematical sense; a whole number

- The *data* or *value* that the variable contains.

Consider one of the variables above, **number_of_cats**. In this case the variables is named **number_of_cats**, it is of type integer (since 22 is an integer), and the value that it contains is **22**. Likewise, the variable **owers_name** is of type string and contains the value **"Peggy"**.

In Python, variable types are changeable. For example, consider the following code segment:

```
>>> x=2
>>> print x
2
>>> x=2.2
>>> print x
2.2
>>>
```

In this example, the variable **x** first is assigned the number 2. Since 2 is an integer **x** is of type integer. We then reassign **x** the value 2.2, which is a float, so then **x** becomes of type float.

3.2. **Converting between types using int(), float(), and str().** Python includes some functions that can be used to convert a variable (and its value) between different types. These functions are summarized below.

- **int()** converts a numerical string or floating point number to an integer.
- **float()** converts a numerical string or integer to a floating point number.
- **str()** converts an integer or floating point number to a string.

Here are some examples of using these functions.

```
>>> a = "100"
>>> int(a)
100
>>> float(a)
100.0
>>> b=20.4
>>> str(b)
'20.4'
```

```
>>> int(b)
20
>>> c="3,003"
>>> int(c)

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    int(c)
ValueError: invalid literal for int() with base 10: '3,003'
>>>
```

In the first line the string **"100"** is assigned to variable **a**. It is important to notice that the assignment **a** = **"100"** is *not* the same as **a** = **100**. Notice the quotation marks in the former; quotation marks denote strings. In the example above, **a** is passed to the **int()** and **float()** functions and the results are shown. Another important thing to notice here is that the values of the variables that are passed as parameters to a function are not updated by the function. That is, if **a** = **"100"**, after executing the line **int(a)**, the value of **a** is still **"100"**; **a** is not changed to **100**. That is because the functions **int()**, **float()**, and **str()** do not have *side-effects*; the value of the parameters are not changed by the function.

The final thing to take note of in the above example is what happens when Python tried to execute the line **int(c)**. There was an error. This is because **c** contained the value **"3,003"** and the function **int()** requires its parameter to consist of only numerical characters. The comma is not a number. This is an example of a subtle error that will drive a programmer crazy.

4. Simple operators

In the preceding examples we have seen the use of the assignment operator, =, which is used to assign a value to a variable. Below we give some other basic operators for numerical and string types.

4.1. Mathematical operators for integers and floats.

Here are some of the simple mathematical operators that can be applied to integers and floats:

- + (Addition); ex: **22 + 23** returns **45**
- **-** (Subtraction); ex: **22 - 23** returns **-1**
- * (Multiplication); ex: **4 * 2** returns **8**
- / (Division); ex: **4 / 2** returns **2**

- ** (Exponentiation); ex: **4 ** 2** returns **16**
- % (Modulo); ex: **23 % 5** returns **3**

If one of the operators above is used with a float on one side and an integer on the other, the result will be a float.

4.2. **Operators on strings.** One of the operators above can also be used on strings.

- + (Concatenation); ex: **"ani" + "mal"** returns **"animal"**
- * (Repetition); ex: **"ha" * 3** returns **"hahaha"**

The concatenation operator will only work when both arguments are strings. If one side is a string and the other is, for example, a float then an error message will be produced. Consider the following examples:

```
>>> age = 5
>>> message = "The child is " + age + " years old."

Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    message = "The child is " + age + " years old."
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Here an error has occurred because the variable **age** is an integer and the literals[5] **"The child is "** and **" years old"** are strings. To fix this we would have to convert the integer **age** into a string before concatenating it with the rest:

```
>>> age = 5
>>> message = "The child is " + str(age) + " years old."
>>> print message
The child is 5 years old.
>>>
```

5. Summary

These notes have covered some basic properties of variables and some operations that can be performed on them. Although the examples given were trivial it is

[5]When a value is "hard coded" into a program it is termed a *literal*.

important to understand exactly what is happening in each of them before moving on to writing more complex (and useful) programs.

6. About these notes

These notes were written by Samuel Johnson for ECS 10, Basic Concepts of Computing, Winter Quarter 2008 for use in discussion sections. Sam can be reached by e-mail at *samjohnson@ucdavis.edu*.