

CS 161 Lab Activities: Selection and String Operations

Part One: Conditional Operators and Expressions

Enter these statements in the interactive environment and study (and understand) the results.

```
>>> a = 10
>>> b = 20
>>> a == b
>>> a > b
>>> a < b
>>> a = = b # with a space
>>> "Alfa" < "Romeo"

>>> a > = b
>>> a != b
>>> a ! = b
>>> a => b
>>> a <= b
>>> "A" < "a"
>>> "Z" < "a"
```

Note that the ASCII codes for lower case letters are higher than those for uppercase letters.

Part Two: Selection Structures

Try each of these in the interactive environment and study the results. (You may prefer to create a small program file and run it to get satisfactory results.) If any results seem strange, study the code and figure out why it works the way it does. Try each of these with several values for the tested variables and see which branches the flow of control takes.

```
1)
>>> x = 18
>>> if x < 18:
...     print "too young"
... else:
...     if x > 18:
...         print "old enough"
...
>>>
```

Try re-writing number one using **elif**.

```
2)
>>> y = "TOM"
>>> if y > "GEORGE":
...     if x < 50:
...         print y, " ", x
...     else:
...         print "place one"
... else:
...     print "y is before GEORGE"
>>>
```

Try this one with various values for *x* and *y*, and try to get it to follow all the paths. (Making a little program and inputting the values will make this easier).

3) Create a little program that asks for an input score (0 to 100) and outputs a letter grade. Try different variations on the structure (for instance, the examples I

showed in class), and test each with values at the borderlines of each grade. Which structures seem to be the most elegant? Which don't work as one would expect?

Part Three: String Operations

1) Try this series of statements in the interactive environment:

```
>>> a = "Peter, Paul and Mary"
>>> len(a)
>>> len ("The hills are alive")
>>> b = "Puff the Magic Dragon"
>>> c = a + " sang "+b
>>> c
>>> c [ 10 ]
>>> len(c)
>>> for i in range (len(c)):
...     print c [ i ]
...
>>> for i in range (len (c) ):
...     print c[ i ],
...
...
```

Note the difference in the results for the two previous **for** structures as a result of the comma at the end of the print statement.

Part Three: String Formatting

Embedding “\t” and “\n” in a string provides tab and carriage-return characters, respectively:

```
>>> print "Hello\nWorld"
>>> print "Hello" * 3
>>> print "Hello \t" * 3
```

Outputting float numbers can be problematic. If we are dealing with dollars and cents, printing out over a dozen digits is a pain. Python provides a solution unlike those available in other languages.

Formatting output requires use of the **string formatting operator** (%) in this syntax:

```
%<width>.<precision><type>” % value
```

This cryptic-looking series before the quote is a **template string** and is used to tell Python how to format the value.

width is how many characters to allocate for the whole thing. If you use 0 the width will be whatever is required. You can use this to force alignment in a column

precision is the number of positions to show after the decimal point. Python will round if necessary

type is a letter that indicates what type of numeric data the value being formatted will be:
f for float, **d** for decimal (integer), and **s** for string.

An example of the use: this print statement forces the value to be shown with two decimal places in a slot wide enough for 8 characters (so it could display up to 99999.99):

```
print "Amount earned this quarter: %8.2f" %amountEarned
```

Here are a few to try in the interactive environment:

```
>>> x = 1234.56789
>>> print "x is %0.3f"%x
>>> print "x value: $%8.2f" %x
>>> "%12.1f"%x
```