SUMMER 2011
KENYA
Due: Friday, June 17 2011

PYTHON L02:
In this lab we practice all that we have learned on variables (lack of types), naming conventions, numeric types and coercion, strings, booleans, operator grouping, and relational operators.

Exercise 2.1) Variables

Recall that variables are containers for storing information. For example
Program text:

```
a = ‘‘Hello, world!’’
print a
```

Output:

```
Hello, world!
```

1. Now, remember the scene that you sketched in the Lab 1. Use variables to write a program that prints out the scene described. What is the minimum number of variables needed?

2. In python, variables are not classified into types. Remember that variable names must start with an letter and not a number. Before you write your code determine which of the following variable names are good. Feel free to check if you are right by using them in your python shell. What are your reasons for why some are not good?

a] list
b] 56thnumber
c] length
d] !Tayo!
e] NUMBER
f] hklgiup
g] Nokia_phone1
h] 1170hjhh
i] answer_to_ex 1.1
j] the answer

When making variable names in python be sure to make variable names that are relevant to your code. Avoid having variable names like “a” or “khlkh” which are ambiguous and do not give the reader any understanding or insight on what is being stored in your

variable. For example if you're making a variable that stores a phone number you can use variable names like "phone_number" or "phone_num" etc.

Exercise 2.2) Types

1. It is important that we know the type of the values stored in a variable so that we can use the correct operators. Python automatically infers the type from the value you assign to the variable. Write down the type of the values stored in each of the variables below. Pay special attention to punctuation: values are not always the type they seem!

1. a = False
2. b = 3.7
3. c = 'Alex'
4. d = 7
5. e = 'True'
6. f = 17
7. g = '17
8. h = True
9. i = '3.14159'
10. j = "---add---"

To verify your answers, you can use the interactive Python shell, but first try to do the exercise without help.

```
>>> a = False
False
>>> type(a)
<type 'bool'>
```

Exercise 2.3) Strings and String Operators

1. Given the following variables:

school = 'Strathmore'
now = ' NOW'

1. school[:4]
2. school[-1]
3. school*2
4. school[:-1] + now + school[-1]
5. now[1]
6. now[4]
7. school*2 + school*[:-1] + now + school*[-1]

You can use the python shell to clarify your answers.

2. In this part of the exercise you will explore string operators a discussed in class. Now write a series of print statements to print out a greeting box and a goodbye message that can be sent as a text.

```
name = 'John Adetunji'
greeting = 'Happy Birthday'
goodbye = 'Goodbye, all!'
space = ' '
star = '*'
```

Use only the first name in the greeting box and only the last name in the goodbye message e.g. they should read "Happy Birthday John" and "Goodbye Adetunji' respectively. You can use other like "-", "#" to make the greeting card prettier but be sure to store them in variable to make your work easy.

Save the file as **greetings.py**.

Exercise 2.4) Boolean:

1. Boolean operators can seem tricky at first, and it takes practice to evaluate them correctly. Write the value (True or False) produced by each expression below, using the assigned values of the variables a, b, and c.

```
a = False
b = True
c = False
```

1. b and c
2. b or c
3. not a and b
4. (a and b) or not c
5. not b and not (a or c)
6. not ((not b or not a) and c) or a

Remember, to work from the inside out, starting with the inner-most expressions, like in arithmetic.

Evercise 2.5) Operations: Order of Operation

Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators +, -, *, / as well as the power operator (**) and the modulus operator (%).

Program Text:

```
x = 5 + 7
```

```
print x
y = x + 10
print y
```

Output:

```
12
22
```

Note that we can use variables in the definition of other variables! Mathematical operators only work on numbers-ints or floats. Statements such as 'Hi' + 5 or '5' + 7 will not work.

1. Input the following sets of equations, and note the difference between int arithmetic and float arithmetic.
You can do this just in your interpreter (you don't need to turn anything in for this part), but pay attention to the output!

a] 5/2, 5/2.0 and 5.0/2- Note that as long as one argument is a float, all of your math will be floating point!

b] 7*(1/2) and 7* (1/2.0)

c] 5 □□2, 5.0 □ □2, and 5 □ □2.0

d] 1/3.0- Note the final digit is rounded. Python does this for non-terminating decimal numbers, as computers cannot store infinite numbers! Take 6.004 to find out more about this...

2. Transcribe the following equations into Python (without simplifying!), preserving order of operation with parenthesis as needed. Save each as the value of a variable, and then print the variable.

a] 3×5/2+3

b] 7 + 9 × 2

c] $\sqrt[4]{-19 + 100}$

d] 6 mod 4 - If you aren't familiar with modular arithmetic, it is pretty straightforward- the modulus operator in the expression x mod y, gives the remainder when x is divided by y. Try a couple modular expressions until you get the hang of it. It is written as 6%4

3. The order of evaluation for an arithmetic or logical expression is determined by operator precedence rules. For each of the following expressions, add parentheses so that the precedence is explicit, and then write down the value of the expression. Then add parentheses that change the value of the expression, and write the new value (you only need to find one example of this). For example:

Example: 3 * 4 + 60 / 2

Answer: (3 * 4) + (60 / 2) = 42

3 * (4 + 60) / 2 = 96

a] 4 + 3 * 4 + 4

b] 4.0 / 3.0 + 1

c] not False and (3 > 4)

4. Some new operators are +=, -=, *=, /=. They change the value of the stored variable in quickest way. we add 6 to a variable in two different ways; note that we get the same result! Try using all of these operators in your interpreter window before moving on.

```
>>> x = 5
>>> x = x + 6
>>> print x
11
>>> y = 5
>>> y += 6
>>> print y
11
```

Exercise 2.6) User input:

1. In this exercise, we will ask the user for her first and last name and date of birth and print them out formatted. Create a new file and call it **userinput.py**. Here is an example of what the program would do:

Enter your first name: *Anne*
Enter your last name: *Willims*
Enter your date of birth:
Month? *July*
Day? *31*
Year? *1987*
Anne Williams was born on July 31, 1987

The text in italics is the what the user inputs.

To print a string and a number in one line, you just need to separate the arguments with a comma. The print statement adds a space between the two arguments.

```
print 'October', 20
```

Or you can convert the number to a string and then use the string operator:

```
print 'October ' + str(20)
```

OPTIONAL: Now, for something completely different... a discussion on how to print strings, most prettily...

```
mo = ’October’
day = ’20’
```

```
year = '1987'
print mo, day, year
```

will have the output

```
October 20 1987
```

Note that none of the commas are in this output! To do that you want something like this:

```
print mo, day+',', year.
```

The + sign concatenates/add two strings together, but can only be used on two strings. Using it on a number and a string will cause an error (because it is ambiguous as to what you want the program to do!). That's why it's a great idea to use raw input for this problem; if you use input you'd have to convert the int to a string. We'll cover this more in-depth on Thursday, when we get to strings, but you may want to play with string concatenation operations now to get everything to look its prettiest.

Exercise 2.7) Zeller's Algorithm- Optional but helpful

Some problem sets will have optional exercises at the end. Feel free to work on these problems if you have time at the end of the assignment, but you certainly don't have to do them. However, you will get excellent practice in Python, and we will give you feedback on any optional work you turn in.

Zeller's algorithm computes the day of the week on which a given date will fall (or fell). In this exercise, you will write a program to run Zeller's algorithm on a specific date. You will need to create a new file for this program, zellers.py. The program should use the algorithm outlined below to compute the day of the week on which the user's birthday fell in the year you were born and print the result to the screen.

Start with the program in Exercise 2.6, but ask for the month as a number between 1-12 where March is 1 and February is 12. If born in Jan or Feb, enter previous year (see the notes below). In the end, print out the name of the user and the day of the week they were born.

Zeller's algorithm is defined as follows:

Let A, B, C, D denote integer variables that have the following values:
A = the month of the year, with March having the value 1, April the value 2, … December the value 10, and January and February being counted as months 11 and 12 of the preceding year (in which case, subtract 1 from C)
B = the day of the month (1, 2, 3, … , 30, 31)

C = the year of the century (e.g. C = 89 for the year 1989)
D = the century (e.g. D = 19 for the year 1989)
Note: if the month is January or February, then the preceding year is used for computation. This is because there was a period in history when March 1st, not January 1st, was the beginning of the year.

Let W, X, Y, Z, R also denote integer variables. Compute their values in the following order using integer arithmetic:
W = (13*A - 1) / 5
X = C / 4
Y = D / 4
Z = W + X + Y + B + C - 2*D
R = the remainder when Z is divided by 7

The value of R is the day of the week, where 0 represents Sunday, 1 is Monday, ... , 6 is Saturday. If the computed value of R is a negative number, add 7 to get a non negative number between 0 and 6.
Print out R. You can check to be sure your code is working by looking at http://www.timeanddate.com/calendar/.

Run some test cases- try today's date, your birth date, any other dates you like.

Feel free to submit your zellers.py code if you wish, we'll take a look at it if you do.