CMPT 120—LAB WEEK 5 (Feb. 3)

In Lecture 9, you were given more details on how to manipulate strings in Python. A goal of this lab is that you become familiar with the string manipulation features.

The material covered in this lab is complementary with the lecture.

GOALS OF THIS LAB:

- Get comfortable using a Python module.
- Understand how to manipulate strings.
- Understand that Python strings are immutable.

General instructions:

- It is recommended you do these exercises in teams of 2 people each. Make sure to take a partner that has roughly the same experience and understand of the topic as you. (If you team up with someone who comprehends the material much better than you, then chances are you will not learn anything.)

- Save your work for future reference and to show the TAs and instructor, so as to show you have actively participated in the lab.

- **Save your files in your home folder U:** (files saved on the Desktop may get lost). Python files should have the **.py** extension written explicitly—the *extension* is the end of filename.

Submission of this lab:

- This week, submission will be done through Coursys, which is available https://courses.cs.sfu.ca/ and it is only due on **Monday Feb. 10th**, so you may focus on studying for the quiz.

- You are asked to submit one Python source code for your entire lab, which means exactly two things:

  1. You should work almost entirely inside an IDLE code window (instead of the IDLE shell).

  2. You should add comments to your code to separate questions, and mark observations or textual answers (answers which are not code).

—PART 1: EXPLORATION—

Exercises in this first part encourage you to explore the behavior of Python.

EXERCISE 1: discovering the functionalities of the str class
The purpose of this exercise is get comfortable with the string manipulation functions.

As you should recall from the previous lab, Python has a built-in documentation system. To get a list of all functions that are possible on strings, you may either type the following in a Python shell:

```
>>> help(str)
```

Listing 1: getting documentation on the string type

Or you could also browse this documentation on the web:

http://docs.python.org/2/library/stdtypes.html#string-methods

In the following, type the listings in your Python shell to see the results for yourself.

The functions that are described in that documentation are to be applied to strings in the following way. If `x` is a string variable, then the function `upper()`, which transforms the string to uppercase, is called doing `x.upper()` the result of this function is the string in uppercase.

IMPORTANT: the string `x` itself will not be modified, it will still contain the original string; experiment as mentioned in Listing 2.

```
>>> x = "hello world!"
>>> print x
hello world!
>>> moreenthusiastic_x = x.upper()
>>> print x
hello world!
>>> print moreenthusiastic_x
HELLO WORLD!
```

Listing 2: applying the upper() function

You may also use the functions that begin with `is...()` which are predicates: they test if a property of the string is verified and return `True` if it is, and `False` if it is not (remember that True and False are logical values, not strings).

Predicates can be used in `if` statements.

```
>>> x = "123"
>>> y = "one two three"
>>> z = ""
>>> x.isdigit()
True
>>> y.isdigit()
False
>>> z.isdigit()
False
>>> if x.isdigit():
       z = int(x) + 10
    else:
       z = 0

>>> z
133
```

Listing 3: using the string predicates

Recall that you can use the type conversion functions to convert from various types of Python: for instance, if you have a string that contains numbers, and you want to convert it an integer (which can then be used with addition, multiplication, etc.), you can use the `int(…)` type conversion function as illustrated in Listing 3.

All these functions can also be applied to a string literal (that means a string that is given with quotes, not a string contained in a variable).

```
>>> "hello world!".upper()
'HELLO WORLD!'
>>> print "hello world!".upper()
HELLO WORLD!
>>> "one".isdigit()
False
>>> "1".isdigit()
True
>>> int(1)
1
>>> int(1) + 3
4
```

Listing 4: applying string functions to literal strings

Try the following:

1. Write a short example of a program that uses `islower()`.
2. Write a short example of a program that uses `find(…)`.

3. Try the function `strip()` on a few strings and try to explain what it does. (This function will introduce a new data type that we have not yet covered in class, and you are invited to say what it is.)

4. Try to explain what the program in Listing 5 does (type in IDLE window, not the shell). You are not supposed to understand what all the commands are doing, but if you experiment with it you can understand how it works.

```
mystring = "hello there what are you doing"
for word in mystring.split():
  print "<", mystring, ">"
```

Listing 5: example of using the `split()` function

EXERCISE 2: getting input from the user
In this exercise you will learn how to ask for input from the user.

At the end of Lecture 9, you were shown the function `raw_input()`. This function asks for some input from the user in the shell. The function can be called with or without a prompt, for instance `raw_input("Type something: ")` will ask the user for input, but also print the message `Type something:` beforehand.

Because the user can input any sort of string of characters, the type that `raw_input()` returns is always a string. As such, when you want to get a numeric information, like the age of the user, to be able to manipulate (addition, subtraction, etc.), you must:
- check that the user entered a string that only uses digits;
- then convert that string of digits into an integer.

Try the following question to get a feel for this function:

1. Write an example where you ask the user to input his age, and then you print out a message indicating whether or not the user has the legal age to drink alcohol in North America. You may for this question assume that the user will only enter a correct numeric string.

2. What happens to your program above when the user enters his name instead of his age?

3. Write a *function* `ask_number()` which takes no parameter, and which will prompt the user for a number: then you will test the string to see if it is a numerical string. If it is, then you return the string converted to an integer using a type conversion function; if it is not, you print a message

saying that there was an error with the input, and to try again, and you call the function itself `ask_number()` so that the user is prompted to retry.

4. Write a program that first asks the user to type a sentence which you will store in the variable `userstring`, then asks the user for a position of a character in that sentence (you may use the function `ask_number()` which you just programmed for this), then tests if this character is between `0` and `len(userstring) - 1` (included): if it is, you return the character at that position in `userstring`, and if not you print an error message and return an empty string `""`.

```
Please type a sentence: hello my dear lass
Please enter a number: 6
The character you want is: m
```

Listing 6: possible execution of the program described in question 4

```
Please type a sentence: hello my dear lass
Please enter a number: 36
Sorry! That position is out of bounds!
```

Listing 7: other possible execution of the program described in question 4

—PART 2: DESIGN AND PROGRAMMING—

Exercises in this second part encourage you to write your own code from scratch.

Notes and recommendations for all exercises of this type (in this lab, and in your future work):

- **To test and debug,** include enough printing messages so that you may see intermediate and final results and also from where the printing is done.

- Alternatively if your program is running fine (that is, there is no syntax error), but rather your problem is that your problem is not computing the result you intend it to, you can also use the Python Tutor website to see step by step what is happening.

- Always **consider different cases and special limit cases**: when your parameters have values that require special case.

- Make sure to initialize all your variables before using them.

- Don't forget you can put comments in your code, by putting a sharp # at the beginning of the line.

- In a program, do not write sentences that are so long that you cannot see them in the IDLE editor original screen.

EXERCISE 3: parsing (= extracting information from) strings
We have seen type conversion that allows you to transform a string into another type, like an integer or a float, which you can manipulate more conveniently. Sometimes strings are not as convenient though, and information is a little bit harder to extract.

Tip: remember about splitting strings, about the function find…

In this exercise, we are interested in strings of the form "hh:mm" where "hh" are two digits comprised between "00" and "23", and "mm" are two digits comprised between "00" and "59". For the moment, you may assume that the hours "hh" is always comprised of two digits (with a prefix "0" if the hour is only on one digit).

1. In a Python shell, check that `"04".isdigit()` return `True` and then check that `int("04")` returns `4`. This means that you don't need to worry about the hours being expressed on one or two digits.

2. Write a function `get_hours(timestring)` that takes one parameter, `timestring`, which has the format "hh:mm" and returns the hours in integer format.

3. Write a function `get_minutes(timestring)` that takes one parameter, `timestring`, which has the format "hh:mm" and returns the minutes in integer format.

4. Using these functions, write `get_minutes_elapsed(timestring)` that takes one parameter, `timestring`, which has the format "hh:mm" and returns the number of minutes which has elapsed from midnight to that time.

5. Write a function `check_timestring_format(timestring)` that takes one parameter, `timestring`, and checks that this string has the format "hh:mm", that is two digits, then a colon, then two digits.

6. Finally, write a program that uses all these functions to do the following: ask the user to type a string in the format "hh:mm", check that this string is in the correct format, then print the number of minutes that have elapsed since that time.