Lab 2
CSCI 127: Introduction to Computer Science
Hunter College, City University of New York
Fall 2024

Learning Objectives:

- Students will learn about Python **strings**
- Students will write programs that use **string methods**
- Students will write programs to perform **string indexing**
- Students will write programs that **get input from users**
- Students will write programs that **loop through strings**
- Students will further explore `for` loops and the **`range()` function**
- Students will write programs that **loop through lists**
- Students will write programs that use the **`split()` function**
- Students will access the **command line**
- Students will use **Unix commands** to list files, navigate directories and create directories

Software tools needed: web browser, Python IDLE programming environment and Unix shell.

Strings

A sequence of characters (i.e. letters, numbers, symbols) is called a `string`. To indicate a string, we use quotes (either single or double, just as long as they match up) to indicate the start and end (a fancier way to say that is: *quotes deliminate the string*). For example, for our first program, we wrote:

```
print("Hello, World!")
```

The string, or sequence of characters, `Hello, World!`, was printed to the screen. We can also store strings to be used again in the program. For example,

```
greeting = "Hello, World!"
print(greeting)
```

creates a location in memory that can be accessed by typing the name (or identifier) that we chose: `greeting`. `greeting` stores the string `Hello, World!`. While the quotes are not stored, we will often write `"Hello, World!"` to make it more clear where the string begins and ends. When we execute the code above, the first line will create the **variable**, `greeting` and store the string `"Hello, World!"` in it. The second line will print out the message:

```
Hello, World!
```

We can use the variable any number of times. For example, if we wanted to print the message twice, we could use:

```
greeting = "Hello, World!"
print(greeting)
print(greeting)
```

We defined a string as a sequence of characters. Each character in the string has a position or index. The first charcater is at index 0, the second character is at index 1, the third character is at index 2, etc.

Take `"Hello, World!"`, for example. This string contains 13 characters. Because we start counting at 0, the last character is at index 12.

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o	,		W	o	r	l	d	!

The first character of a string is always at index 0. If a string has `x` number of characters, the last character in the string is always at index `x-1`.

More Useful String Methods

Strings are a common data type and there are many built-in functions for them.

- Guess which each does from its name, then
- Click the forward button in the code window below to see if you guess was correct.

```

# Guess what the following string
# Then click the forward button t

print(greeting.upper())
print(greeting.lower())
print(len(greeting))
print(greeting.count('o'))
print(greeting.count(' '))
print(greeting.find('ll'))
```

Frames Objects

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 1 of 7

Visualized with pythontutor.com

NOTE: The `find()` command gives the location of `"ll"` which is 2 if you start by counting the first character as 0.

Getting Input

Last week, we used the `print()` function to write messages to the user of our program. This week, we introduce, `input()`, to get information from the user. Here's the basic format:

```
aString = input("Put a message here to show user: ")
```

where the string `"Put a message..."` is replaced by the prompt you would like the user to see and `aString` with the name of the string you are using in your program.

Python 3.6

```
mess = input('Please enter a messa
print("You entered", mess)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 1 of 2

Visualized with pythontutor.com

Print output (drag lower right corner to resize)

Frames Objects

Let's write a program that combines the asking the user for input with the string commands a the beginning of the lab. The program will:

1. Prompt the user for a message and store it in the variable, `mess`.
2. Print the message to the screen.
3. Print the message in all capital letters.
4. Print the message in all lower case letters.

To start, open `IDLE` and start a new file window. Put a comment (lines that begin with '#') that includes your name and a short description of what the program does.

1. Next, fill in the code that prompt the user for a message and store it in the variable, `mess` (see the example above).
2. Print the message to the screen (also done in the example above).
3. To print in all capital letters, you can use the `upper()` command (see first example in the lab).
4. Print the message in all lower case in a similar way.

Save your file as you go, and then run it. Try different messages to make sure it works with different inputs. When it works add your name in a comment with a short description of what it does and see the Programming Problem List.

Looping Through Strings

You can also loop through strings. Try running the code below by clicking the `Next` button to execute each instruction:

Python 3.6

```
myString = "I love Python!!"

print(myString)

for c in myString:
    print(c)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Print output (drag lower right corner to resize)

Frames Objects

Each character has a number assigned to it. When you write a character, it is converted to its number, and that is stored instead to save space. Python uses the standard Unicode encoding (which extends the popular ASCII encoding to new symbols and alphabets). For example, `ord('a')` give the Unicode number for the character, `a`, which is 97.

Let's look at the Unicode of the characters in our string:

Python 3.6

```
myString = "I love Python!!"

print(myString)

for c in myString:
    print(c, ord(c))
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Print output (drag lower right corner to resize)

Frames Objects

To go the other direction, there's a function `chr()` which takes numbers and returns the corresponding character. For example, `chr(97)` returns `'a'`. Let's look at the characters with unicode from 65 to 69:

Python 3.6

```

for i in range(65,70):
    print(i, chr(i))

print("That's it for now!")
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Step 1 of 12

Print output (drag lower right corner to resize)

Frames Objects

Combining `ord()` and `chr()` demonstrated in the two programs above, modify the first of the two programs to:

- To prompt the user to enter a string and store it in `myString`, and
- print each character, its Unicode number + 1 and the character corresponding to that number.

For example, for character 'a' with Unicode number 97, it prints:

```
a 98 b
```

The `range()` statement has several different options:

- `range(stop)`: if you have only a single number in the parenthesis, it will generate all the numbers from 0 to `stop-1`. For example, `range(5)` generates the numbers `0, 1, 2, 3, 4`.
- `range(start, stop)`: if you have only two numbers in the parenthesis, it will generate all the numbers from start to `stop-1`. For example, `range(65,70)` generates the numbers `65, 66, 67, 68, 69`.
- `range(start, stop, step)`: if you have three numbers in the parenthesis, it will generate all the numbers from start to `stop-1`, increasing by `step` each time. For example, `range(100,200,10)` generates the numbers `100, 110, 120, 130, 140, 150, 160, 170, 180, 190`.

Looping Through Lists

You can loop through a list the same way as you loop through a string.

The `split()` function breaks up a string into substrings, throwing away the character or delimiter on which the function splits.

Then you can loop through the list just like you loop through a string to access the individual substrings.

For example, given the string `"FridaysSaturdaysSundays"`, `split('s')` will break up the string into a list of days: `["Friday","Saturday","Sunday",""]` throwing away the 's' characters on which it splits.

Notice that it will produce an empty string at the end because of the 's' character at the end of "Sundays".

Try running the code below by clicking the `Next` button to execute each instruction:

Python 3.6

```
myString = "FridaysSaturdaysSunday

days = myString.split('s')

for day in days:
    print(day)
```

Edit Code & Get AI Help

line that just executed
next line to execute

< Prev Next >

Print output (drag lower right corner to resize)

Frames Objects

Modify the code to remove the 's' at the end of 'Sundays', now run and inspect the list. What changed?

Indexing: looping Through Strings, Revisited

There's another way we can loop through strings, using the index of each character. Let's assume that we have:

```
greeting = "Hello, World!"
```

Before, we printed out the whole string with:

```
print(greeting)
```

If we wanted to print out only the first letter, we could write:

```
print(greeting[0])
```

where the number between the **square brackets** is the **index** of the character, in this case, 0, or the very first character of the string.

Try guessing what the following code does and then running it by pressing the `Next` button to execute each instruction:

Python 3.6

```
greeting = "Hello, World!"

print(greeting[0])
print(greeting[1])
print(greeting[2])
print(greeting[-1])

for j in range(5):
    print(greeting[j])

for j in range(len(greeting)):
    print(greeting[j])

for j in range(0,10,2):
    print(greeting[j])

for j in range(10,0,-1):
    print(greeting[j])
```

Edit Code & Get AI Help

line that just executed
next line to execute

Print output (drag lower right corner to resize)

Frames Objects

Acronyms

It is often quite useful to break a string into pieces, and work with those pieces, instead of the whole string.

Based on the programs demoed above, combine the use of `split()` and indexing (the use of square brackets to access individual characters in a string using the character's index) to write a program that *takes a phrase and creates an acronym of the phrase: that is, the first letter of each word*.

To approach a problem, it is useful to break it into steps:

1. Prompt for a phrase & read it in.
2. Make the phrase upper case.
3. Split up the phrase into words.
4. Take the first letter of each word (keep in mind that `split()` returns a list of the words), concatenate and make an acronym of it

To **concatenate** strings you can use the + operator. For example:

```
print('hi'+'bye')
```

will output:

```
hibye
```

Now translate the above **pseudocode** (informal but detailed description of the steps in a program) into python and test that your program works as follows:

```
Enter a phrase:  City University New York
Your phrase in capital letters:  CITY UNIVERSITY NEW YORK
Acronym:  CUNY
```

Files & the Command Line Interface

The **terminal** is a program that allows users to type commands (separate from Python) that we can use to communicate with the operating system. It's often called the **command line interface** or **shell**. It predates the graphical interface that is now available and is incredibly useful for automating tasks and working on remote servers. We will slowly introduce **shell commands** over the course of the semester and eventually incorporate them into some programming assignments.

To access a Unix shell environment on your computer to run shell commands:

- Linux: You can launch a terminal by clicking on the icon for a terminal (or using the search feature to find it).
- Macintosh (OSX): The OSX operating system, run by modern mac computers, is based on Unix (albeit a slightly different one than Linux). You can bring up a terminal window by using Spotlight or search for "terminal".
- Windows: The underlying terminal uses commands that are different from Unix (they're based on the Microsoft Disk Operating System, or MS-DOS). You can get a Linux terminal on your Windows machine with the Windows Subsystem for Linux (WSL). You can find directions on how to install it in the next section. WSL is useful to have throughout this course and future computer science courses. Alternatively, there are several on-line emulators of Unix that you can use via the browser, such as cocalc.com which provides a terminal in which you can type shell commands and the ability to create and upload files which will be useful later on. You can also use replit and choose `bash` as the language.
- Phones/Tablets: Since most do not run Unix (or allow access to a terminal), it is best to use a web-based environment with terminal emulator like cocalc.com.

We have already used the shell in the previous lab. When you typed:

```
$  idle
```

you were asking the operating system to launch the program `idle3`.

Let's create a new **directory**, or folder, to hold your images (using the shell or command line). Where it says `thomasH` below, replace with your name. It is easier if you avoid spaces in the names:

- Open a new terminal window.
- In the terminal window, type:

  ```
  $ pwd
  ```

  (The `$` represents the prompt at the command line-- no need to type it.)

  This tells you the **path**, or location, of your current working directory.

- To see what is in the folder, you request a 'listing':

  ```
  $ ls
  ```

- To add a new directory, we type:

  ```
  $ mkdir thomasH
  ```

  (replace `thomasH` with your name to create a directory for you!).

  Now, when we list the current directory, we should see the new item (namely the directory with your name):

  ```
  $ ls
  ```

- Let's change directories:

  ```
  $ cd thomasH
  ```

  If we ask for a listing (`ls`), we will now see the contents of your directory, which is initially empty.

The graphical interface and shell are operating on the same underlying system, so, changes you make with one can be seen by the other. If you use your OS' graphical user interface to open the folder (directory) you started in when you typed `pwd`, you will see the folder you just created.

In the next lab, we will explain how to copy and move files between directories.

Installing the Windows Subsystem for Linux

If you don't have a Windows machine, you can move on to the Lab Quiz

If you are using Windows, here's directions on how to get the Linux terminal on your computer:

1. Search "Turn Windows features on or off" in the Windows search bar and open it.
2. Look for "Windows Subsystem for Linux", make sure the box is checked, and hit OK (You will have to restart your computer).
3. Open Microsoft Store application and search Ubuntu.
4. Click on Ubuntu, hit get, and wait for it to install (You can use any version, the one without a version number will give you the latest version of Ubuntu).
5. Launch the Ubuntu application and allow first time setup.
6. Create a username and password (When entering a password, no characters will appear, but you are typing your password).
7. Update Ubuntu with the command:

   ```
   sudo apt update && sudo apt upgrade
   ```

   (You will have to enter the password you set) You should run this command periodically as Ubuntu will not update on its own.

You have successfully installed and setup WSL and can now use the terminal as you would on Ubuntu.

These installation instructions were written by Owen Kunhardt. You can find the full guide here.

What's Next?

You can start working on this week's programming assignments. The Programming Problem List has problem descriptions, suggested reading, and due dates next to each problem.