Differences between the CS 102 Course and Textbook

Due to the textbook using an older version of Python (Version 2) and a different application for writing your applications (JES), there are a few important differences to keep in mind when reading the text and working through examples.

Python 2 vs Python 3

In this course we will be using Python 3 instead of the older and deprecated (no longer supported) Python 2. Most code in your textbook is still valid Python 3 code, but it is important to note some key differences.

The `print` Function (Textbook Section 2.4)

In Python 2, the print function look like this:

```
print "hello"

print 10
```

In Python 3, print is used like a function, so you need to use parenthesis to surround the data you would like to print like this:

```
print("hello")

print(10)
```

If you forget and use the old Python 2 style for printing, you will get an error telling you that you need to use parenthesis.

Integer Division (Textbook Section 2.4)

In Python 2, when you divide two integers using the division operator like this:

```
5 / 3
```

The division operation would truncate (drop or cut off) the decimal part of the answer resulting in the value 2.

Using the division operator in Python 3 always results in a floating-point value (rational number with decimal point precision). In the case of 5 divided by 3, Python 3 would return 2.5 as the result. Sometimes it is useful to only perform integer division especially when a rational number would not make sense. Consider dividing an odd number of people in a room into two groups. It is not actually possible to have less than one whole person in a group. In Python 3 we can use the Integer division operator to perform division without returning the decimal precision like this:

```
5 // 3
```

The result of this operation will return the value 2.

The `range` Function (Textbook Section 3.2)

From a semantic (behavioral) standpoint the range function in Python 2 and 3 are essentially the same for most use cases. There is an important difference to be aware of between the two Python versions.

In Python 2, the range function returned a list (a collection) of values representing the requested range like this (observe the use of >>> indicating executed code and a following indented line representing the output):

```
>>> print(range(5))
      [0, 1, 2, 3, 4]
```

In Python 3, the range function returns an object which behaves similarly to the list when used in loops or decision structures. This difference becomes apparent in Python 3 if you were to try and print the range like this:

```
>>> print(range(5))
      range(0, 5)
```

Instead of a list showing the individual values in the range, we receive an object that represents the range.

User Input (Textbook Section 10.1)

In Python, users can respond to a request for input data from a running application by typing data into the Thonny Shell, Unix Terminal, or Windows Command/Powershell prompt and then hitting the enter key. In Python 2 user input can be collected using the following command:

```
myName = raw_input('Please enter your name: ')
```

In Python 3, user input is collected the same way, but the function looks like this instead:

```
myName = input('Please enter your name: ')
```

Other than using different names for the functions, both versions of Python are semantically similar from a general usage perspective.

JES versus Thonny

The textbook uses a program called JES (Jython Environment for Students). This application provides two different components.

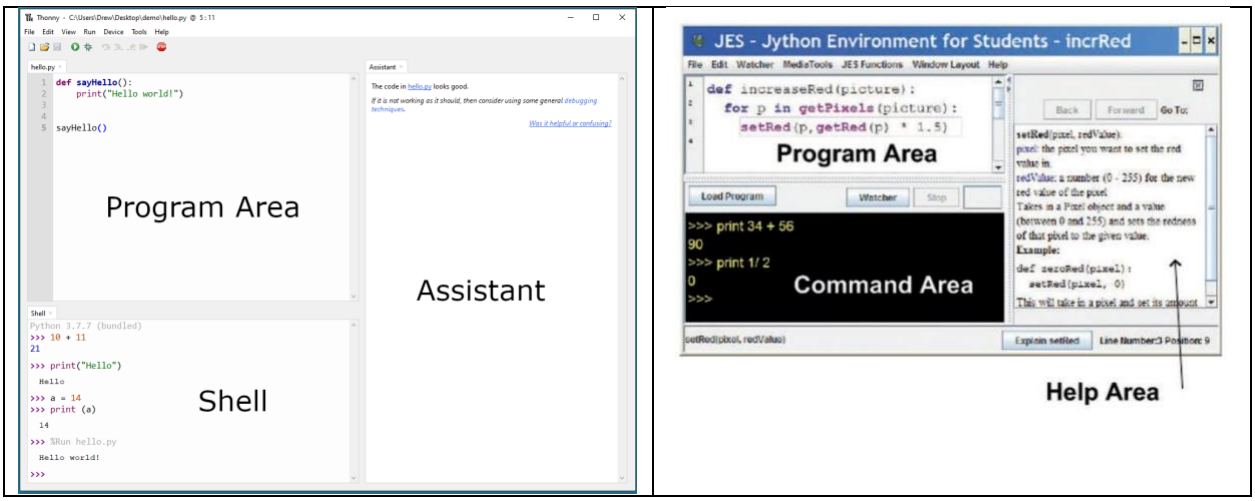
1. An IDE (integrated development environment) to write your applications
2. A library (collection of additional functionality or features) that is not built-in to the Python 3 programming language to allow for interactions with media (pictures, sound, video, etc.)

We will replace each of these components with two different alternatives to preserve as much of the functionality that JES provided as possible.

The Thonny IDE

To address the IDE component, we will be using Thonny. This application is very similar to the JES editor as seen in Figure 1 below:

Figure 1- Thonny (right) and JES (left) configured with similar layout and windows.



Many of the windows and their associated functionality in JES are also available in Thonny.

- The Program Area in both Thonny and JES is where you can open and edit a Python program.
- The Shell in Thonny is the same as the Command Area in JES. This window shows the results of running a program from the Program Area and allows for interactive execution of Python code to see the results in real-time.
- The Assistant in Thonny is very similar to the Help Area in JES. Here you can find helpful hints about any issues that might be present in your code when you attempt to run/save.

Thonny has several other windows available from the View menu. Many of these may not be of much use to you, but I will demonstrate some of them in lecture videos. If you would like to explore Thonny more on your own, there is a brief overview video of the application including some of the windows and their functions by Aivar Annamaa from the University of Tartu in Estonia (where Thonny was developed) https://www.youtube.com/watch?v=nwIgxrXP-X4&feature=emb_logo.

The Media Library

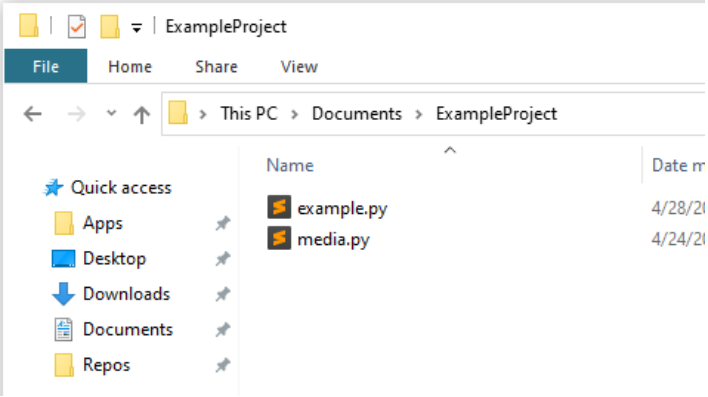
JES comes with a set of features not found in the standard Python 3 program language. These features from JES have been re-implemented in a library called “media.py”. This file can be added to your python projects to support much of the same functionality JES provides.

Adding media.py to a Project

To use media.py you will need the following statement at the top of your python code file after any leading heading comments but before any other lines of code:

```
# Example Heading Comments…
…
import media
```

You will also have to make sure that the media.py file is in the same folder as your python project like this:



Accessing media.py Functions

With the file imported, you can now call (or access) any of the functions in the media.py library. For example if you wanted to use the `pickAFile()` function from Chapter 1 of your book, you will use that function like this:

```
media.pickAFile()
```

Notice the `media.` at the beginning of the function name. This is required so Python knows to look inside the media.py file to find the function. Since the example in the book do not require this, it is very easy to forget to add `media.` to the function call or to know when you need it. **DON'T PANIC!** One way is to try a function without it. If the call function needs the `media.` added, you will get an error message that looks like this:

```
NameError: name 'pickAFile' is not defined
```

Simply append the function call with `media.` and rerun the code. This will likely work unless you have misspelled the function or used incorrect capitalization (Python is case sensitive). Another option is to look up the function name in the media.py documentation file [ADD LINK HERE]. Any function in the documentation will require the leading `media.` text to work properly.

At the end of any project where you have used `import media`, you will need to use a special function to make sure all the media resources quit properly. This function is called `media.quit()` and it will need to be the last statement executed by your program.

Extra Features

A few convenience features have been added to media.py.

- `media.savePicture()` – Save an image to an output file so it can be viewed or used again later.
- `media.saveSound()` – Save an audio file to an output file so it can be listened to or used again later.

Known Issues / Limitations

The media.py library provides most of the functionality you will need for you projects, however you may encounter some bugs along the way. Most of these issues require using a slightly different function or an external application as a work around. For any other issues not listed below that arise during the semester, I will post a public solution or alternative to resolve the problem.

Limitation: Only WAV files are supported

Solution: Media.py cannot play audio files of any other format.

Limitation: Media functions cannot be run from the Thonny Shell Window

Solution: Using media.py from the Thonny Shell does not work reliably. You will have to run media.py functions using the program area.

Issue: Occasion problems with JPEG image files

Solution: On Windows JPEGs image files can sometimes have issues. Simply convert the file to a PNG image format and it should work fine.

Issue: WAV file sound does not play

Solution: Use the function `media.blockingPlay()` instead of `media.play()`.