Week 1: Review.
GBenoit

Usually the first class meeting is a chance for fork to introduce themselves, learn about the levels and breadth of experiences in computing, and learn about the admin/pedagogy of the course.

Unfortunately, we were unable to achieve all of these goals given the incompetence of the new systems' integration.

I'm still unable to log into bCourses and unable to do a lot of functions. I expect this to be resolved by Sept 01.

What were some of the main pedagogical goals of Week 1?

A lot of experienced programmers lack some fundamentals; new coders may be fearful/excited at the prospect of learning and that's great! The first 8 weeks of the course are mostly ramping up with code syntax in general and python-specific syntax and techniques. The version of python we use must be at least version 3.8.x although some of us use 3.10.1. To know what version you have leads us to the terminal window.

Terminal Window:

Our course uses Unix or a version/flavor or spin-off thereof. Unix as an OS (operating system) lets us communicate directly to our computer; and it lets python communicate directly with the computer. That's why we can invoke the python interpreter by starting a Unix terminal window and entering the command python. Some computers have python already installed (usually version 2, which won't help us at all!); and the updated version to three must be invoked by using "python3" instead of "python." [Note: If you change the symbolic link you can set "python" to refer to the latest version of python and just use "python" to start the python interpreter.]

Another reason we're concerned about the terminal window is (a) file permissions. All files and directories, you'll recall, have 3 levels of access rights and 3 types of users. If you create a file, you "own" the file and have automatically read and write permissions. Often we add "execute" permissions for scripts. Then there's the "group" level permissions. Finally, there's "all" or "world" or "other." Webpages are a good example. You create the webpage so you can read/write it. The webserver however might require "execute" permissions; but we don't want visitors to our webpage to write over our pages, so they have only "read" permissions. Hence in the terminal when we type "ls -Fla" we'll see an alphabetical list of file names, their permissions, dates created, and size. E.g.:

```
-rw-r--r--   1 gb   staff    5449 Feb 18  2021 class_get-set_1.ipynb
drwxr-xr-x@  4 gb   staff     128 Aug 30  2020 dwhelper/
-rw-r--r--   1 gb   staff  1884989 Aug 21  2020 get-pip.py
-rw-r--r--   1 gb   staff   19988 Feb 18  2021 get_set_techniques_week_07.ipynb
-rw-r--r--   1 gb   staff       0 Jan  6  2021 junkfile.txt
-rw-r--r--@  1 gb   staff      31 Jul 15 12:56 my.cnf
drwxr-xr-x  14 gb   staff     448 Sep 23  2021 node_modules/
-rw-r--r--   1 gb   staff    6513 Sep 23  2021 package-lock.json
-rw-r--r--   1 gb   staff      58 Sep 23  2021 package.json
```

filename: Week1-Review.rtf

Week 1: Review.
GBenoit

```
drwxrwxrwx   2 gb   staff     64 Jul 15 13:13 tmp/
```

When a program (like python) creates a file, it may be the owner and not you! So you'll try to read the file but can't! So you may need to change the ownership (chown) command before you can read the file, integrate its contents with other files, etc.

Work Processes and Encoding:

Lots of folk use SQL, or R, or other programs to manipulate or work with data. Ultimately we integrate those data into a new file. So it's not uncommon, say, for a work group to create spreadsheets and then export them as tab-delimited files. Others in a group might use R and output data in .json or .txt or whatever .... python, then, needs to be able to read all these different file types when we begin to collect and "clean" the data.

One of the first big issues is encoding. What is "UTF-8"? Well, older Windows OS would save files according to an encoding scheme called win-1265 (using 8 bits per char); some Chinese computers would use Big-5 or TwinBridge (and save as 16 bits per char) and so on. UTF-8 is the standard way of saving and using data today - and often we have to open and save files as UTF-8 despite their original encoding. [Otherwise there's a mismatch and the resulting data look like junk; you'll see square boxes or ? or whatever on the screen.]

Another very important point is the "new line" character. Today there's the \n. Older machines (and so files) might use \r\n [which is Windows' way of saying return, new line - like an old typewriter]. Consequently when reading files we have to look for that \r - just a heads-up for later this term.

File Names and Paths:

Sometimes we are required to save files using the full path - starting with the "root". E.g., this review file is in my ucb folder, which is in the Directory folder, which is in my user folder gb, which in turn ... and so on... here you see the pwd (print working directory) command to show me the "real" path:

```
gb@babykitty Fall2022 % pwd
/Users/gb/Documents/ucb/Fall2022
```

When python saves a file, you might be required to use this full path and not the "relative" path - relative to where we are in the hierarchical file structure of the hard drive, e.g.,

f.write("/Users/gb/Documents/ucb/Fall2022/project1.txt", encoding='utf-8')

The main points, then, of week 1 was to get us to think of the operating system, its relationship with python, and to begin to see a lot of the hiccups or "gotchas" that can happen.

Notebook:

The notebooks and resources are available on bCourses under Resources. I've created all these notebooks and posted all the optional resources. The optional resources address all kinds of topics: some in .pdf, some are books made available by their authors, etc.

filename: Week1-Review.rtf

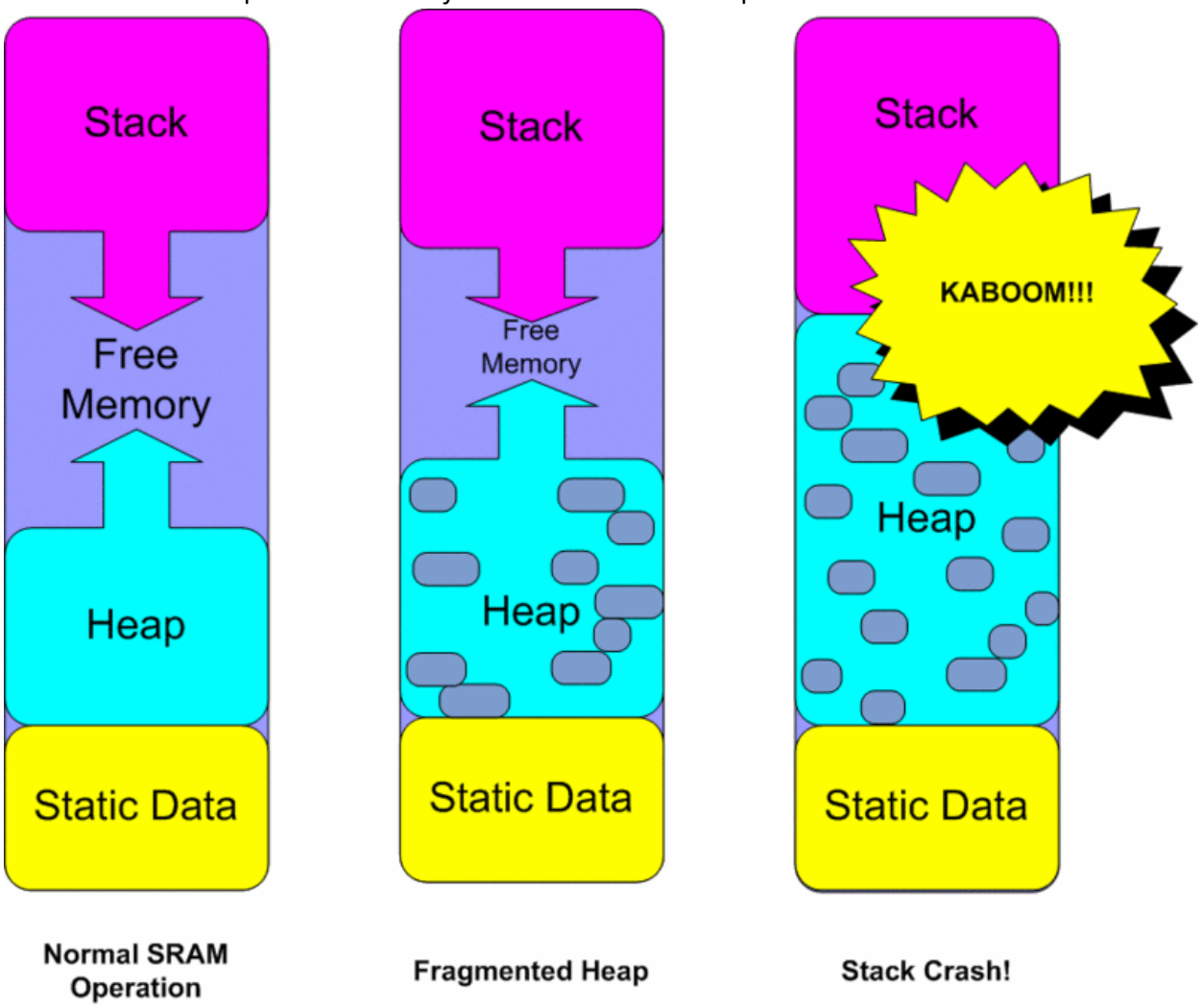
Week 1: Review.
GBenoit

Jupyter Notebook: I used version 3.8.3 for some of them; this term I'm using 3.10.1. This is because Python has a few new features. One of the most important is the "match" command - the equivalent of the "case" command in other languages.

For week 2:

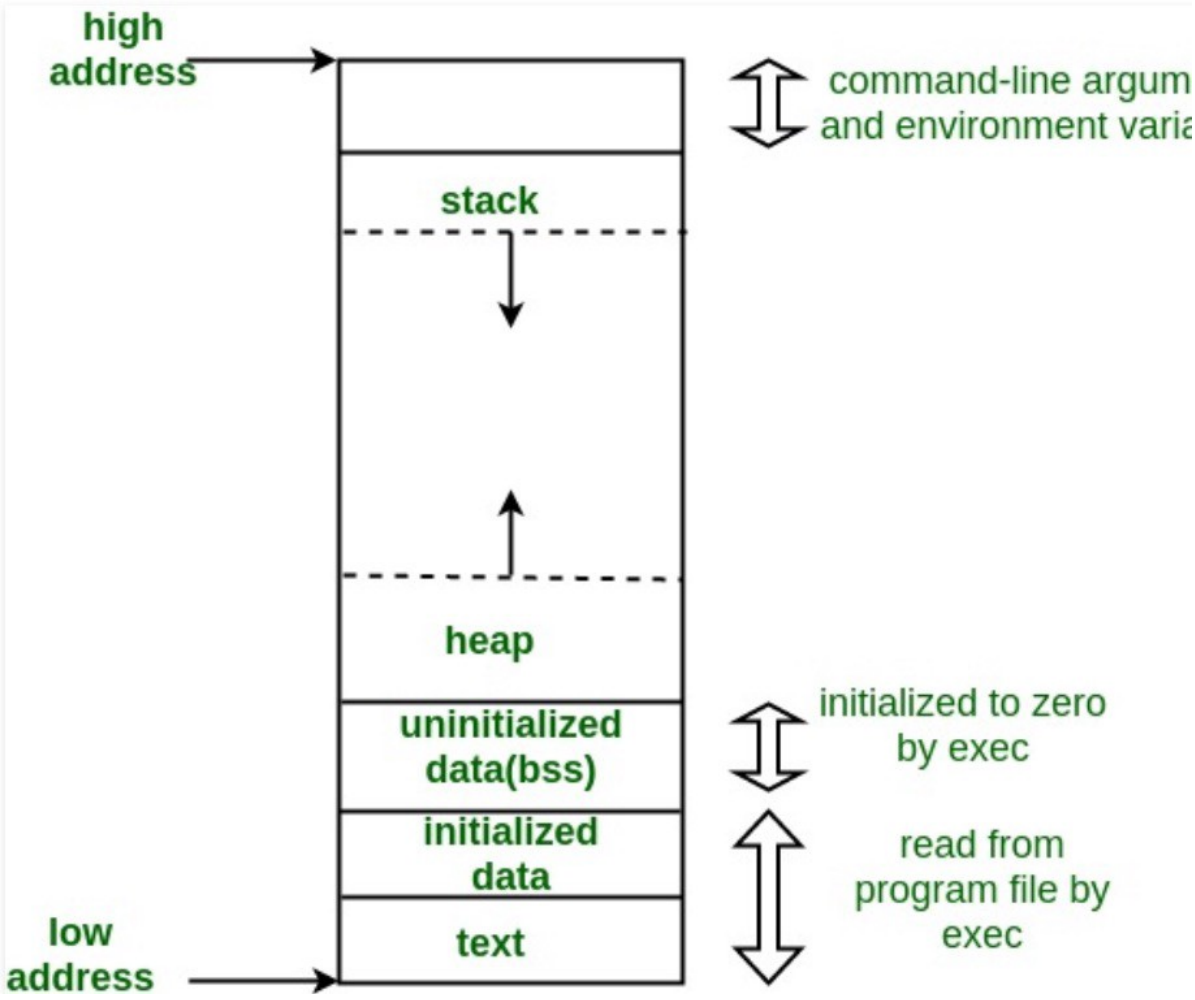
We start to build up sequentially from small data structures (like char, byte, int, ultimately working to sets, tuples, dictionaries, and lists). Along the way I try to demonstrate techniques learning them and especially trying to given them a useful context so you can see how these structures fit into the larger problem-solving and modularization that's required for all coding projects.

I think we still have images of RAM - this is important 'cause how data and functions are stored in RAM fall into a couple of areas. They're the stack and the heap. Functions and static variables



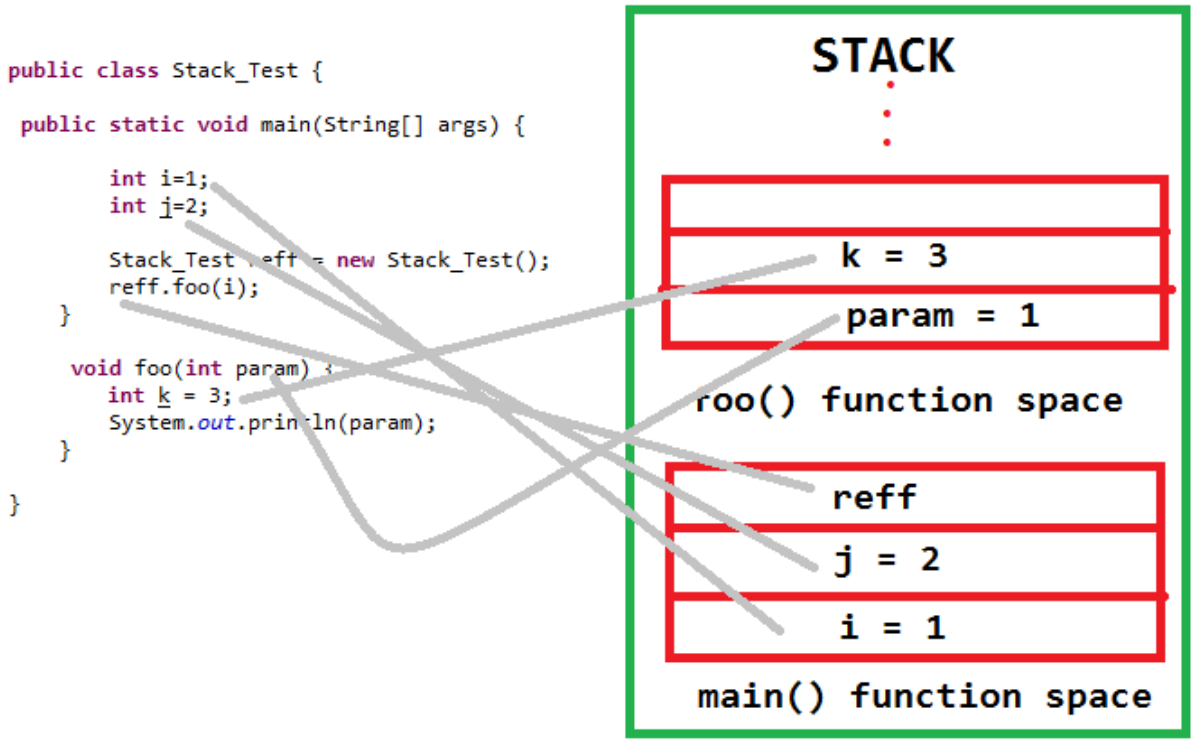
filename: Week1-Review.rtf

Week 1: Review.
GBenoit



filename: Week1-Review.rtf

Week 1: Review.
GBenoit



What this all means is that (a) we code to get the job done and usually have enough free RAM not to worry … and (b) we should remember that certain code choices we make end up in the heap (and can't change) or in the stack (which can) … and so when we start coding in python we'll find techniques (like lambda functions) that otherwise would take up room in the heap, but actually end-up in the stack and the memory space can be regained when the lambda function ends. Fun for week 6!

--- end of the notes. -gb

filename: Week1-Review.rtf