Homework 6

Abhigyan Chattopadhyay
ME19B001

November 1, 2019

1 Homework - Session 13

1.1 Question 1: List out all the built-in data types of C and C++ language, the space they consume in the memory and their ability to represent information.

1.1.1 Data Types in C

Type	Size (in bits)	Type of Data stored
signed char	8	Integers (-128 to 127)
unsigned char	8	Integers (0 to 255)
char	8	Characters (ASCII) (Actually Integers allocated to ASCII Code)
short int	16	Integers (-32768 to 32767)
unsigned short int	16	Integers (0 to 65535)
int	32	Integers (-2,147,483,648 to 2,147,483,647)
unsigned int	32	Integers (0 to 4,294,967,295)
long int	32 or 64	Integers (-2,147,483,648 to 2,147,483,647 or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
unsigned long int	32 or 64	Integers (0 to 4,294,967,295 or 0 to 18,446,744,073,709,551,615)
long long int	64	Integers (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
unsigned long long int	64	Integers (0 to 18,446,744,073,709,551,615)
float	32	Real Numbers (1.2E-38 to 3.4E+38)
double	64	Real Numbers (2.3E-308 to 1.7E+308)
long double	80	Real Numbers (3.4E-4932 to 1.1E+4932)

1.1.2 Data Types in C++

Type	Size (in bits)	Type of Data stored
bool	1	True or False value (0 or 1)
signed char	8	Integers (-128 to 127)
unsigned char	8	Integers (0 to 255)
char	8	Characters (ASCII) (Actually Integers allocated to ASCII Code)
wchar_t	16 or 32	Characters (Unicode or other ASCII Extensions)
short int	16	Integers (-32768 to 32767)
unsigned short int	16	Integers (0 to 65535)
int	32	Integers (-2,147,483,648 to 2,147,483,647)
unsigned int	32	Integers (0 to 4,294,967,295)
long int	32 or 64	Integers (-2,147,483,648 to 2,147,483,647 or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
unsigned long int	32 or 64	Integers (0 to 4,294,967,295 or 0 to 18,446,744,073,709,551,615)
long long int	64	Integers (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
unsigned long long int	64	Integers (0 to 18,446,744,073,709,551,615)
float	32	Real Numbers (1.2E-38 to 3.4E+38)
double	64	Real Numbers (2.3E-308 to 1.7E+308)
long double	80	Real Numbers (3.4E-4932 to 1.1E+4932)

References:

1. GNU C Manual

2. C - Geeks for Geeks

1.2 Question 2: Are there built in data types in Fortran that are not in C? Check out the complex number, for example and write about that.
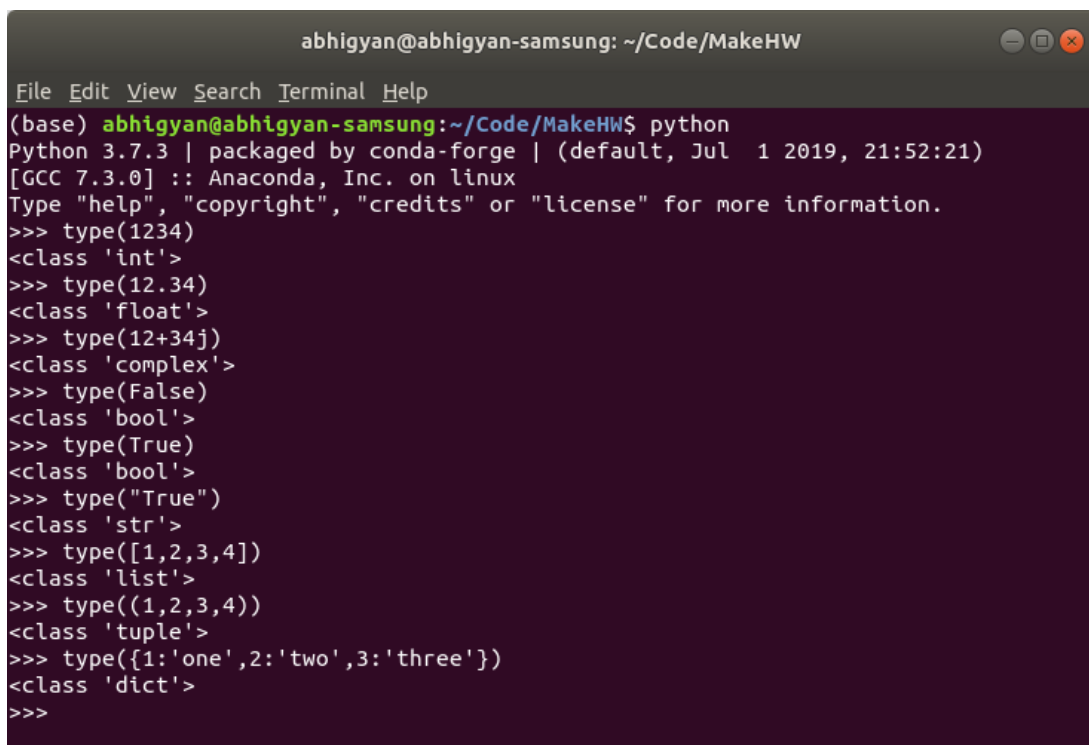
C does not have the logical or the complex data type which are built-in for Fortran. However, newer versions of C have these in them as well, in the standard headers. C++ has a built-in boolean data type, but doesn't have a Complex data type either. This basically shows that Fortran is much more oriented towards scientific coding than C/C++ which are more general purpose languages.

1.3 Question 3: From what you can pick from Wiki pages, list out the built in data types in the languages Python, Sage and Octave. Comment on what needs to be programmed if you were to build up such a functionality in, say, C language.

1.3.1 Data Types in Python/Sage:

Sage is basically built on top of multiple libraries, and at its base, it is purely Python. As a result, Python and Sage have the same built-in or primitive data types. On top of this, Sage adds multiple derived or user-defined data types (like VectorSpace, Matrix, MatrixSpace, etc.) Thus, I'm only listing the Python data types here, as both Python and Sage have too many derived data types to list out explicitly here. To know more about the, see the Sage Documentation and the Python Documentation.

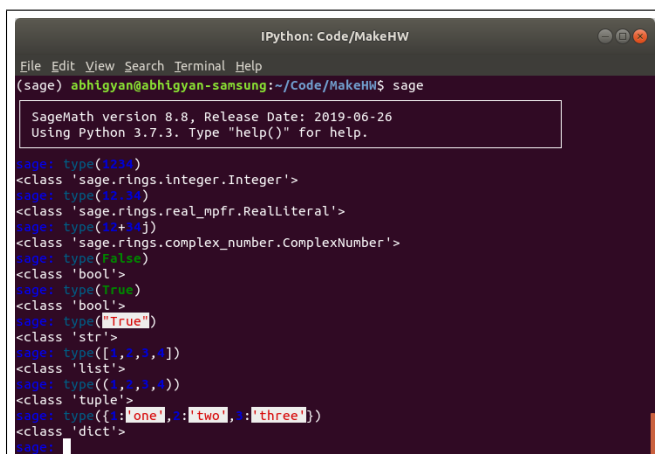| Type	Sub-Types	Purpose
Numeric	int, float, complex	Represent any numerical value
Boolean	bool	True/False Values
Sequences	str, list, tuple	Groups of ordered items
Dictionary	dict, set	Groups of unordered items, linked lists

Figure 1: One of each in-built data type in Pure Python
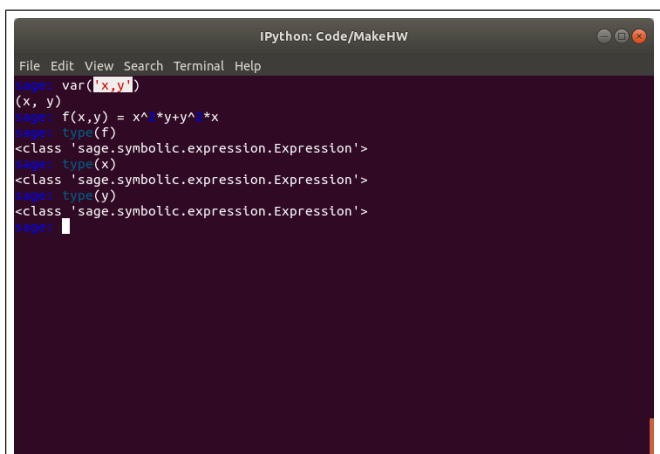
(a) One of each of the Python built-in data types in Sage. Note how Sage stores the numerical data types as types other than the Python primitives

(b) Some of the other (cooler) data types in Sage. Expression is the most commonly encountered one in symbolic computations

Figure 2: Various data types in Sage

1.3.2 Data Types in Octave:

In Octave, every single piece of information is a matrix. The contents of the matrix can be of different data types, which are tabulated below:

Data Types	Use Case
int8, int16, int32, int64	Integers
uint8, uint16, uint32, uint64	Unsigned Integers
char	Characters
double, single	Real Numbers
logical	Boolean

Also, see the Octave Documentation.

1.3.3 Difficulties in implementing this in C

One of the biggest disadvantages of C is that it doesn't support operator overloading. This leads to many major bottlenecks during the implementation of various functionality that are available in Python, Sage and Octave. As a result, we would need to create structs and implement each of them as individual constructs, and still not have the notational simplicity available in Python, Sage and Octave.

Thus, to build up such a functionality in C language, we would need to create various structs that could contain multiple numerical and non-numerical values in them, depending on what they were created for. However, the only way to make them work like their Sage or Octave counterparts would be to call functions. For example, while adding two matrices in Octave, we can use: x = A+B; or x=A∗B where both A and B are matrices.

In C, it might be implemented as: x = A.add(B) or x = A.matrix_product(B) which loses out the intuitive meaning that is conveyed in the former language. However, C is a blazing fast language, and anything that can be accomplished in Sage or MATLAB/Octave can be also achieved in C, albeit via a longer and more tedious code writing process. This is the reason why MATLAB/Octave and Sage might be used for prototyping and modelling, while C is mostly used for production.

1.4 Question 4: After you have installed jupyter-notebook and python, launch the notebook and watch out which port number the notebook server is running on. Can you also find out the process ID of this notebook server while it is running?
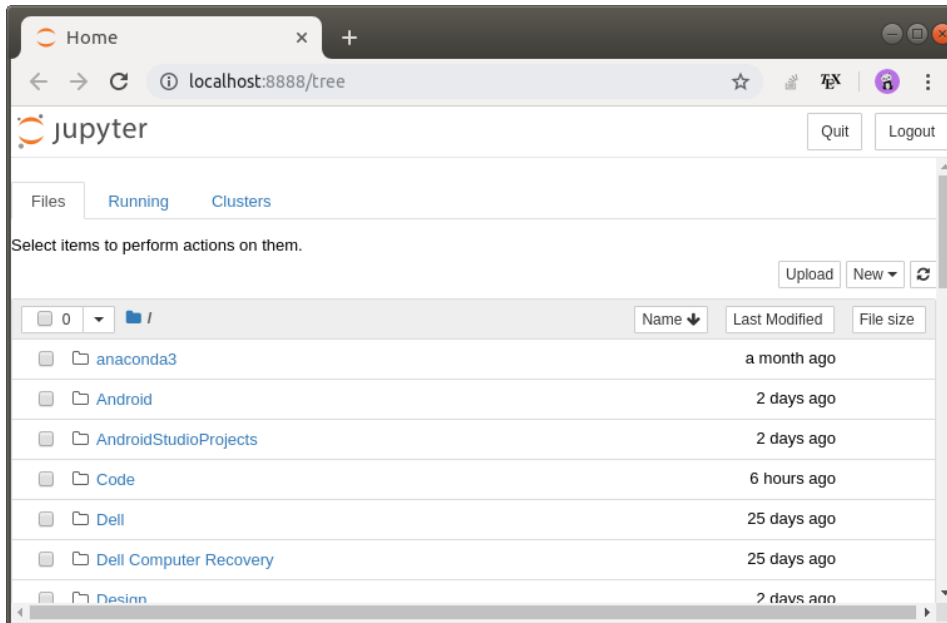

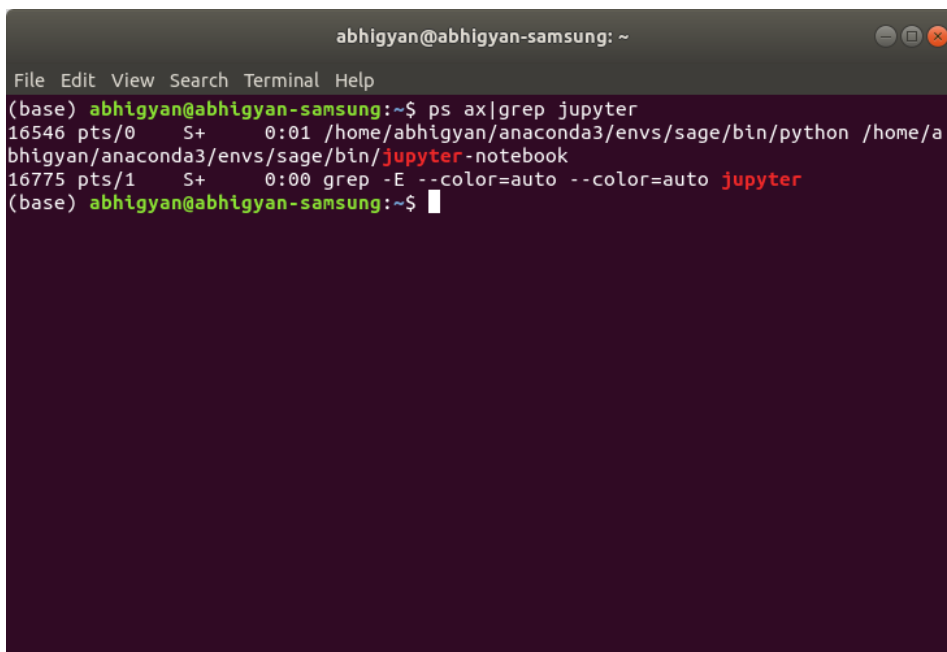Figure 3: Port Number is 8888, gleaned from address bar

Figure 4: Process ID is 16546, gleaned from bash command

1.5 Question 5: From Wiki pages, list out what are the different languages that are supported by the jupyter notebook interface. Figure out what is meant by "markup syntax used in html or xml" and the "mark down" syntax of jupyter notebook.

1.5.1 Jupyter Notebooks

Quoting verbatim from the Jupyter Edu Book:

"The Jupyter system supports over 100 programming languages (called kernels in the Jupyter ecosystem) including Python, Java, R, Julia, Matlab, Octave, Scheme, Processing, Scala, and many more. Out of the box, Jupyter will only run the IPython kernel, but additional kernels may be installed. Language support continues to be added by the open source community and the best source for an up-to-date list is the wiki page maintained by the project: Jupyter Kernels. These projects are developed and maintained by the open source community and exist in various levels of support. Some kernels may be supported by a vast number of active (and even paid) developers, while others may be a single persons pet project. When trying out a new kernel, we suggest exploring a kernels community of users and documentation to see if it has an appropriate level of support for your (and your students) use."

Thus, some languages supported in Jupyter (which we students of MM2090 have heard of in the last couple of weeks) are:

- Python

- Julia

- Sage

- MATLAB/Octave

1.5.2 Markup vs Markdown

Markup is the name given to any language that uses tags to define elements in a document. It is a generic term for a number of languages, like Hyper Text Markup Languages (HTML) and XML (eXtensible Markup Language).

Markdown, on the other hand, is a specific language, which is used to generate various types of markup depending on the scenario. In a browser, Markdown produces HTML, which is then rendered by the browser. LaTeX can also be used within Markdown, and Markdown can basically be read without needing to be rendered, as it focuses on readability. There are some flavours of Markdown, most notably the GitHub flavour (used in many GitHub pages in the Readme.md file).

2   Homework - Session 14

2.1   Question 1: Edit the jupyter notebook to explore a mapping of all data types of C language with the corresponding ones of python.

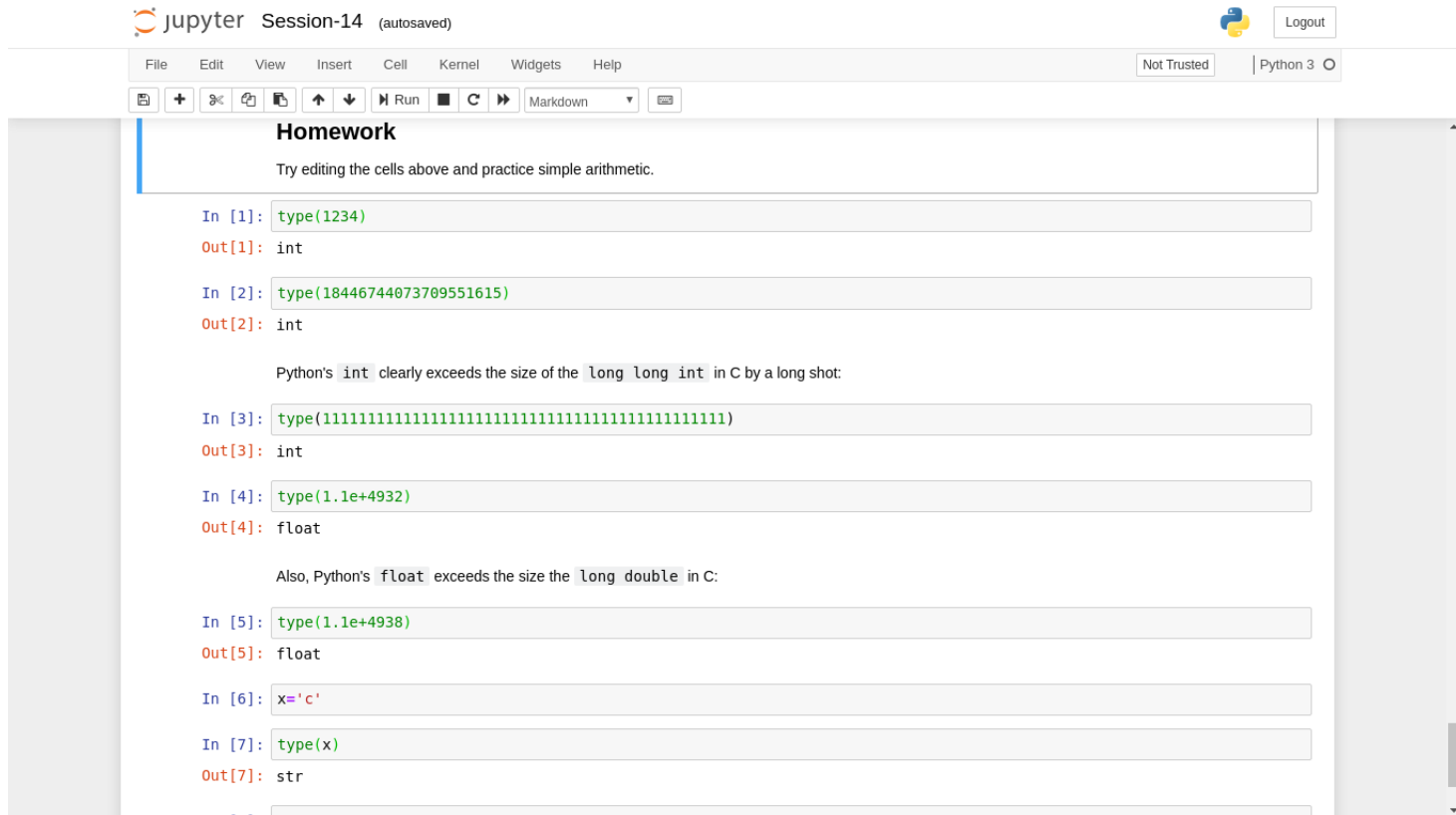| C	Python
(unsigned and signed) short int, int, long int, long long int	int
float, double, long double	float
(unsigned and signed) char	str

Figure 5: Trying out various Data Types in Python

2.2   Question 2: Figure out what data types are available in python that are not readily available from C.

As it has already been seen earlier, we know that C doesn't have any built in complex data type. Neither does it have dicts, which are available in Python. (Note: C *does* have an array data type that closely corresponds to Python's list , and by extension, char∗ is similar to Python's str . Thus, we can't say that those are not reily available in C, as they are available in <stdio.h> itself.)