CHAPTER 2

Getting Started with GNU Octave and Jupyter

In Chapter 1, you learned in detail how to get your Windows computer, Linux, and Raspberry Pi ready for working with GNU Octave and Jupyter Notebook. You also learned how to get started with GNU Octave programming and Jupyter Notebook. In this chapter, you will delve deeper into GNU Octave programming and you will mostly use Jupyter Notebook for programming demonstrations in interactive mode. The following is the list of topics you will learn and demonstrate in this chapter:

- Simple mathematical operations
- Built-in mathematical constants
- Getting help
- Variables in GNU Octave
- Global variables
- Conventions for naming variables
- Clearing the command prompt

A. Pajankar and S. Chandu, *GNU Octave by Example*,
https://doi.org/10.1007/978-1-4842-6086-9_2

Simple Mathematical Operations

Let's get started with some simple concepts. In this section, you will learn how to perform simple mathematical operations on numerical operands. It is recommended to create a new notebook for every chapter and save all of the notebooks in the same directory on your computer. So create a new notebook and create a markup cell with a heading that says **Simple Mathematical Operations**. Then run the following statement in the next cell:

```
2 + 5
```

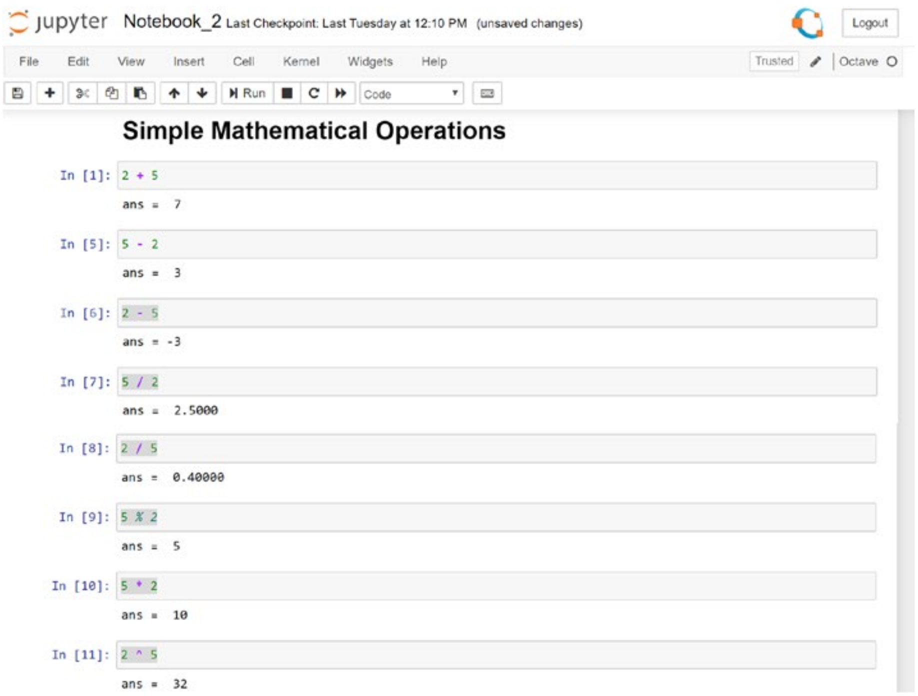
It will execute and show the following output:

```
ans =  7
```

Similarly, run the following statements and see the output:

```
2 - 5
5 / 2
2 / 5
5 % 2
5 * 2
2 ^ 5
```

After executing the statements above, you'll get the output shown in Figure 2-1.



Simple Mathematical Operations

```
In [1]: 2 + 5
ans =  7

In [5]: 5 - 2
ans =  3

In [6]: 2 - 5
ans = -3

In [7]: 5 / 2
ans =  2.5000

In [8]: 2 / 5
ans =  0.40000

In [9]: 5 % 2
ans =  5

In [10]: 5 * 2
ans =  10

In [11]: 2 ^ 5
ans =  32
```

Figure 2-1. *Screenshot of the simple mathematical operations in action*

When various operators are used in a single expression, the operator precedence is similar to the behavior you find in mathematics or other programming languages. In mathematics, it is commonly referred to as BODMAS or PEDMAS, which is shown in Table 2-1.

Table 2-1. *BODMAS/PEDMAS*

Operation	Notation	Operation
Brackets	{[( )]}	**P**arentheses
Orders	^, **	**E**xponents
Division	/	**D**ivision
Multiplication	*	**M**ultiplication
Addition	+	**A**ddition
Subtraction	-	**S**ubtraction

In Octave, the operator preference is parentheses over other operators, and division and multiplication over addition and subtraction. When operators with equal precedence occur, the operator precedence goes from left to right.

For example, the answer to the expression

```
10 * 5 - (5 + 2)^2 + 10 / 5
```

```
10 * 5 - 7^2 + 10 / 5
expression inside the parentheses is computed
10 * 5 - 49 + 10 / 5
exponents are computed
50 - 49 + 2
division and multiplication (operators of equal precedence) are
computed
3
addition and subtraction (operators of equal precedence) are
computed
```

You can run this complex expression in a new cell. It will execute the following output:

```
ans = 3
```

It is advisable to use parentheses when writing complex expressions because they overrule any operator, make your code readable, and you can avoid mistakes that are easy to overlook.

Built-in Mathematical Constants

There are many built-in mathematical constants in GNU Octave. In a new cell, create a markdown cell with a heading that says **Built-in Mathematical Constants**. You can retrieve them in multiple formats. Run the following code:

```
e
```

This returns the value of the constant e that is the base of natural logarithms:

```
ans =  2.7183
```

Run the following code:

```
e(3)
```

It returns a 3x3 matrix of es as follows:

```
ans =

   2.7183   2.7183   2.7183
   2.7183   2.7183   2.7183
   2.7183   2.7183   2.7183
```

You will learn about matrices in the next chapter. You can even have a custom sized matrix of es as follows:

```
e(3, 2)
ans =
   2.7183   2.7183
   2.7183   2.7183
   2.7183   2.7183
```

You can also create a matrix of more than two dimensions:

```
e(2, 2, 3)
ans =
ans(:,:,1) =
   2.7183   2.7183
   2.7183   2.7183
ans(:,:,2) =
   2.7183   2.7183
   2.7183   2.7183
ans(:,:,3) =
   2.7183   2.7183
   2.7183   2.7183
```

You can have these constants in single (32-bit representation) or double (64-bit representation) precision as follows:

```
e(3, 2, class="single")
e(3, 2, class="double")
```

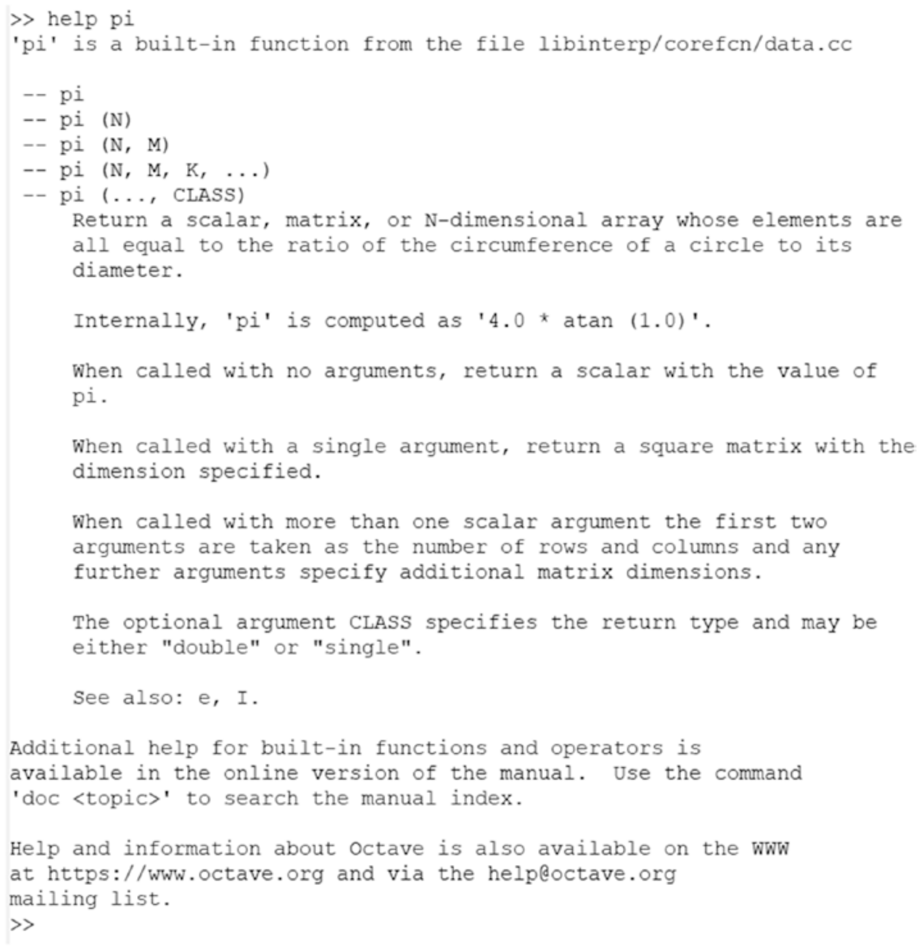
Similarly, there are other constants that can return a single value or matrices as demonstrated above. Let's look at each of them one by one. If you run `pi`, it returns the value of the constant pi. The constants `i`, `j`, `I`, and `J` return the imaginary unit that is $\sqrt{-1}$. `Inf` returns infinity and `NaN` returns **Not a Number**. The next three constants are system-dependent (processor-dependent, to be precise). The first one is `eps`. It returns the relative spacing between any two adjacent numbers in the machine's floating-point system representation. `realmax` returns the largest floating-point number, and `realmin` returns the smallest floating-point number represented by the system. Check these constants and their respective matrices yourself, like you did in the demonstration for the constant `e`.

Getting Help

You can get help for built-in constants and functions (you will learn about them later in the book). Suppose you want to get more information about the built-in constant `pi`. You can run the following in the command prompt of the GNU Octave GUI:

```
help pi
```

The output is shown in Figure 2-2.



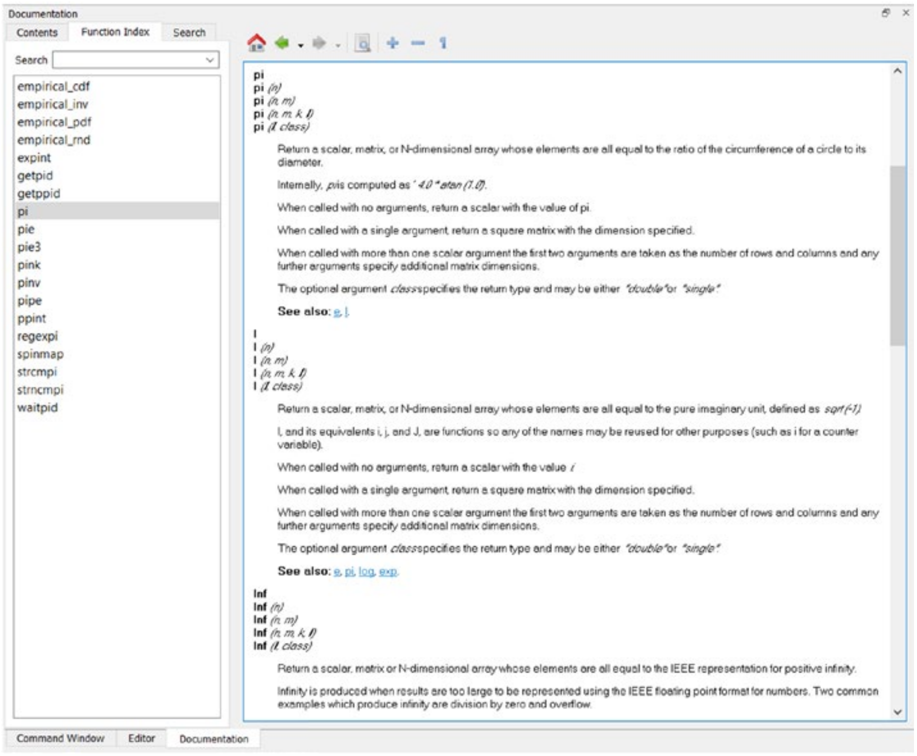
```
>> help pi
'pi' is a built-in function from the file libinterp/corefcn/data.cc

 -- pi
 -- pi (N)
 -- pi (N, M)
 -- pi (N, M, K, ...)
 -- pi (..., CLASS)
     Return a scalar, matrix, or N-dimensional array whose elements are
     all equal to the ratio of the circumference of a circle to its
     diameter.

     Internally, 'pi' is computed as '4.0 * atan (1.0)'.

     When called with no arguments, return a scalar with the value of
     pi.

     When called with a single argument, return a square matrix with the
     dimension specified.

     When called with more than one scalar argument the first two
     arguments are taken as the number of rows and columns and any
     further arguments specify additional matrix dimensions.

     The optional argument CLASS specifies the return type and may be
     either "double" or "single".

     See also: e, I.

Additional help for built-in functions and operators is
available in the online version of the manual.  Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
>>
```

***Figure 2-2.** The output of the command help*

Similarly, you can execute the following command for documentation:

```
doc pi
```

It opens the relevant documentation in the documentation tab of the GNU Octave GUI, as shown in Figure 2-3.



pi
pi *(n)*
pi *(n, m)*
pi *(n, m, k, ...)*
pi *(..., class)*

Return a scalar, matrix, or N-dimensional array whose elements are all equal to the ratio of the circumference of a circle to its diameter.

Internally, *pi* is computed as '*4.0 * atan (1.0)*'.

When called with no arguments, return a scalar with the value of pi.

When called with a single argument, return a square matrix with the dimension specified.

When called with more than one scalar argument the first two arguments are taken as the number of rows and columns and any further arguments specify additional matrix dimensions.

The optional argument *class* specifies the return type and may be either *"double"* or *"single"*.

See also: e, I.

I
I *(n)*
I *(n, m)*
I *(n, m, k, ...)*
I *(..., class)*

Return a scalar, matrix, or N-dimensional array whose elements are all equal to the pure imaginary unit, defined as *sqrt (-1)*.

I, and its equivalents i, j, and J, are functions so any of the names may be reused for other purposes (such as i for a counter variable).

When called with no arguments, return a scalar with the value *i*.

When called with a single argument, return a square matrix with the dimension specified.

When called with more than one scalar argument the first two arguments are taken as the number of rows and columns and any further arguments specify additional matrix dimensions.

The optional argument *class* specifies the return type and may be either *"double"* or *"single"*.

See also: e, pi, log, exp.

Inf
Inf *(n)*
Inf *(n, m)*
Inf *(n, m, k, ...)*
Inf *(..., class)*

Return a scalar, matrix or N-dimensional array whose elements are all equal to the IEEE representation for positive infinity.

Infinity is produced when results are too large to be represented using the IEEE floating point format for numbers. Two common examples which produce infinity are division by zero and overflow.

Figure 2-3. *GNU Octave documentation*

This way, you can find out more information about the built-in functions and constants offered by GNU Octave.

Variables in GNU Octave

A variable is an addressable memory (RAM) location where you can store data temporarily as long as your program (in script mode) or session (in interactive mode) is running. You can address the variable with a name. Each variable in the same program or session has a unique name. Let's look at how you can create a variable. Type and run the following statement:

```
a = 3.14
```

It will immediately show the value of variable `a` in the following line. `=` assigns the value on the left to the variable on the right (in the above case, `a`). You can suppress the display of output by adding `;` to the statement as follows:

```
a = 3.14;
```

If you type the variable again in a new cell and execute it, it shows the value of the variable. You can also assign values to multiple variables as follows:

```
a = 1, b = 2, c = 3
a =  1
b =  2
c =  3
```

As you can see, you use the comma (,) between assign statements to do this.

We will discuss different types of variables later in this book.

Global Variables

There is a special way of declaring some variables as global variables: you use the keyword `global` before the variable name. The global variables may only be initialized once. If you run the following two lines one after the other

```
global a = 1
global a = 2
```

the variable `a` still contains a value of 1.

Using global variables has other benefits which will be addressed in later chapters.

Conventions for Naming Variables

In order to avoid errors and confusion while programming, you should adhere to the following conventions when naming variables:

- Names should not start with a number but you can use numbers in the variable name anywhere after the first character.
- Variable names are case sensitive.
- Names can include the underscore character.
- Keywords cannot be used as names of variables.

It is a good practice to use meaningful variable names because the code will be easier to read and debug.

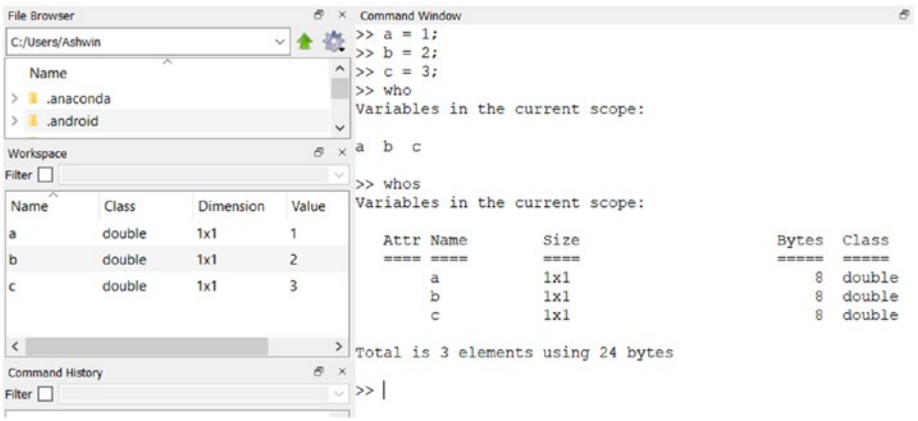
You can retrieve the list of the current keywords in the Octave version by running the statement `iskeyword()`. The following is the list of reserved keywords in the current version of Octave:

```
ans =
{
  [1,1] = __FILE__
  [2,1] = __LINE__
  [3,1] = break
  [4,1] = case
  [5,1] = catch
  [6,1] = classdef
  [7,1] = continue
  [8,1] = do
  [9,1] = else
  [10,1] = elseif
  [11,1] = end
  [12,1] = end_try_catch
  [13,1] = end_unwind_protect
  [14,1] = endclassdef
  [15,1] = endenumeration
  [16,1] = endevents
  [17,1] = endfor
  [18,1] = endfunction
  [19,1] = endif
  [20,1] = endmethods
  [21,1] = endparfor
  [22,1] = endproperties
  [23,1] = endswitch
  [24,1] = endwhile
  [25,1] = enumeration
  [26,1] = events
  [27,1] = for
  [28,1] = function
  [29,1] = global
  [30,1] = if
  [31,1] = methods
  [32,1] = otherwise
  [33,1] = parfor
  [34,1] = persistent
  [35,1] = properties
  [36,1] = return
  [37,1] = switch
  [38,1] = try
  [39,1] = until
  [40,1] = unwind_protect
  [41,1] = unwind_protect_cleanup
  [42,1] = while
}
```

The commands `who` and `whos` show the list of variables and details, respectively. Create a few variables in Octave's interactive prompt in the GUI and run the command. First, create a few variables as follows:

```
>> a = 1;
>> b = 2;
>> c = 3;
```

The output of the `who` and `whos` commands and the workspace panel in the Octave GUI is shown in Figure 2-4.



Name	Class	Dimension	Value
a	double	1x1	1
b	double	1x1	2
c	double	1x1	3

```
>> a = 1;
>> b = 2;
>> c = 3;
>> who
Variables in the current scope:

a  b  c

>> whos
Variables in the current scope:

   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        a           1x1                          8  double
        b           1x1                          8  double
        c           1x1                          8  double

Total is 3 elements using 24 bytes

>> |
```

***Figure 2-4.** The output of the commands who and whos*

Try running the commands `who` and `whos` in the Jupyter notebook too. The output will be the same. Many times, it is recommended to purge all the unused variables from memory. You are required to manage the memory manually when you handle large datasets. You can purge all the variables from memory with the command `clear`. If you run this command in the Octave GUI's command prompt, you can see all of the variables disappear from the workspace. Use the `doc` and `help` commands to obtain more information about the usage of `clear`.

Clearing the Command Prompt

You can clear the command prompt of Octave (running in terminal or the GUI, both) by running the `clc` command. You don't need to use this command in Jupyter Notebook. We discussed the methods to clear the output in the cells of Jupyter Notebook in the last chapter.

Summary

In this chapter, you learned about the basics of GNU Octave programming and explored the GUI interface in a bit more detail. The concepts you use in this book will be helpful to you in further chapters.

In the next chapter, you will explore different data types in GNU Octave in detail. The next chapter will be more coding-extensive than this one.