MIVA

OPEN UNIVERSITY

Practical Demos

Introduction to Computing Sciences COS 101

Credit Unit: 3

MIVA OPEN UNIVERSITY

HEADQUARTERS

PLOT 1059, O.P. FINGESI ROAD,

UTAKO, ABUJA

Course Development Team

Content Editor:

Language Editor: Udochi Obiukwu, MA English Language

Instructional Designer:

Table of Contents

List of Figures

Course Introduction

Programming is a way of instructing the computer to perform various tasks. These instructions are written in a language that the computer understands. Just like the Westerner in Nigeria can understand Yoruba and the Northern Hausa, so is the case with computers. Computers understand instructions that are written in a specific syntax called a programming language.

Learning Outcomes

When you have completed this session, you should be able to:

1. Describe various computer programming languages.
2. Navigate through Anaconda distribution for Python, RStudio for R and VS Code for C++, and Java.
3. Perform basic arithmetic operations in Python, R, C++, and Java.
4. Assign values to variables and work with different data types
5. Work with comparison and logical operators

Session 1: Introduction to Programming

Programming is a way of instructing the computer to perform various tasks. These instructions are written in a language that the computer understands.

The instruction could be as simple as:

- Adding the population of Lagos and Abuja.
- Or the square root of 16.

Just like the Westerner in Nigeria can understand Yoruba and the Northern Hausa, so is the case with computers. Computers understand instructions that are written in a specific syntax called a programming language.

Terminologies, Acronyms and their Meaning

.ipynb	Jupyter notebook or JupyterLabextension
.py	Python extension
IDE	Integrated Development environment (IDE)
R	R programming language
EDA	Exploratory Data Analysis

NaN	Not a Number
np	NumPy
pd	Pandas
os	Operating system
AI	Artificial Intelligence
int	Integer data type
str	String data type
bool	Boolean data type

What is a Programming Language?

A programming language is the set of instructions through which we can interact with computers. This instruction(s) is given in the form of syntax or codes by a computer programmer to execute certain tasks. Programmers use specialised languages to communicate with computers for a certain task.

Figure 1.1 Programming language

Various Programming Languages

Some of the popular Programming languages you will be learning in CS 101 include Python, R, C++, and Java. The figure below shows some of the programming languages that are available in the world today:

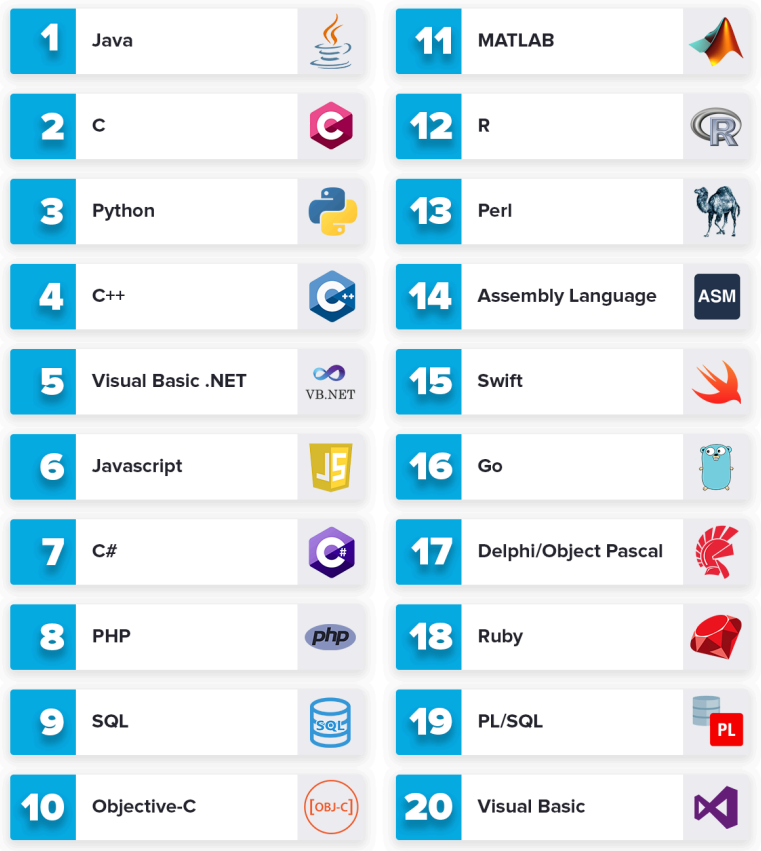


Figure 1.2 Various programming languages

Why Should You Learn Computer Programming?

- Programming is fun
- Programming increases your critical thinking
- You can automate a task easily
- You can earn more money by coding
- The backbone of a Technology Company

Programming Languages for Data Science

There are several programming languages, such as Python, R, Scala, and Julia, for data science and you, who would soon become a data scientist (a data steward), should learn and master at least one language for data science projects. This course will teach you the Python programming language.

Figure 1.3 Programing languages for data science

Python Programming Language

Python is a powerful general-purpose programming language. It is an open-source and easy-to-use language that was created by Guido van Rossum in 1991. Python is used in data science, web development, and so on. Python is one of the most widely used data science programming languages in the world today. Its simple, easy-to-use syntax makes Python an excellent language to learn to program for beginners. Python was designed for readability and has some similarities to the English language with influence from mathematics.

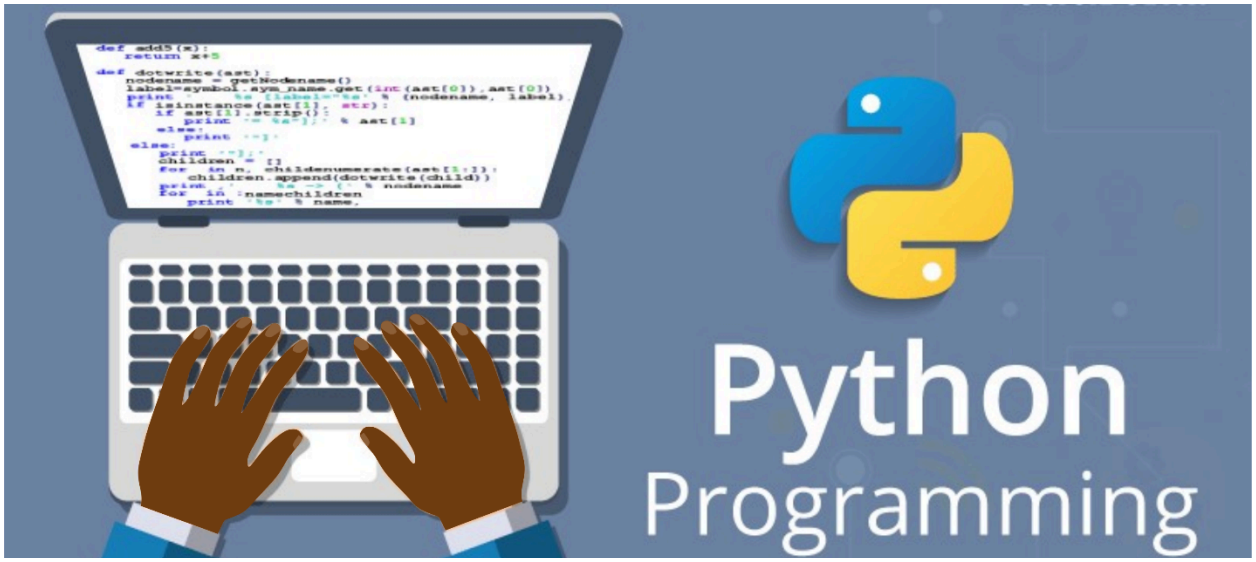
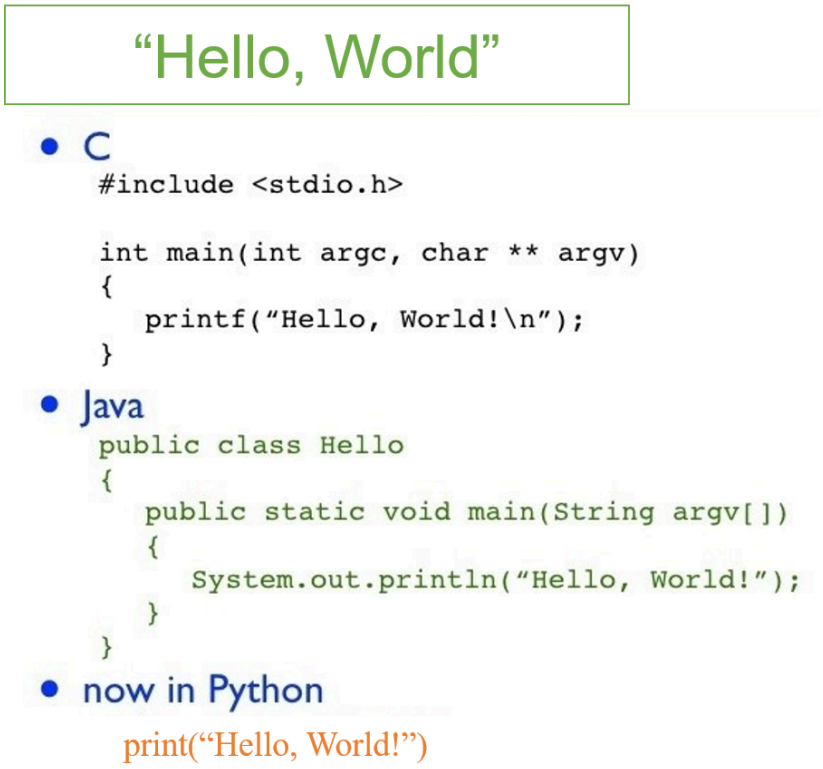


Figure 1.4 Python programming language

Why Python?

- It is easy to learn, and the code is readable.
- It is one of the most popular programming languages
- There is a lot of Open Source contribution on Python, i.e., people have made a lot of free contribution on python to make programming easier for others.
- Instructions are given in a shorter code compared to that of other programming languages. For example, the figure below shows how to print 'Hello World'. Merely looking at the code, we would see that python's syntax is shorter and easy to read.



Session 2: Introduction to Anaconda Distribution

Anaconda is a distribution of the Python for scientific computing, that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS.

To install Anaconda, please follow the following steps:

- Download Anaconda Navigator App using via this link https://www.anaconda.com/products/individual#windows, then click on Download

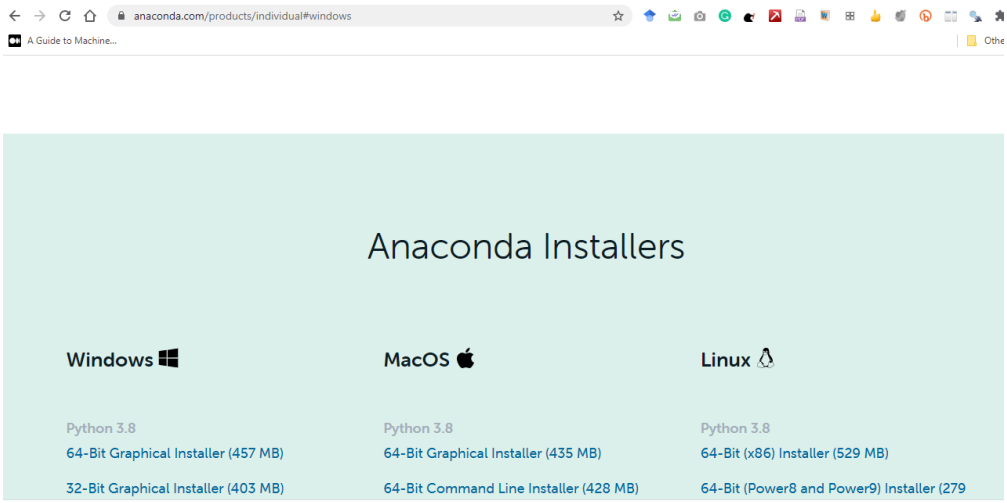
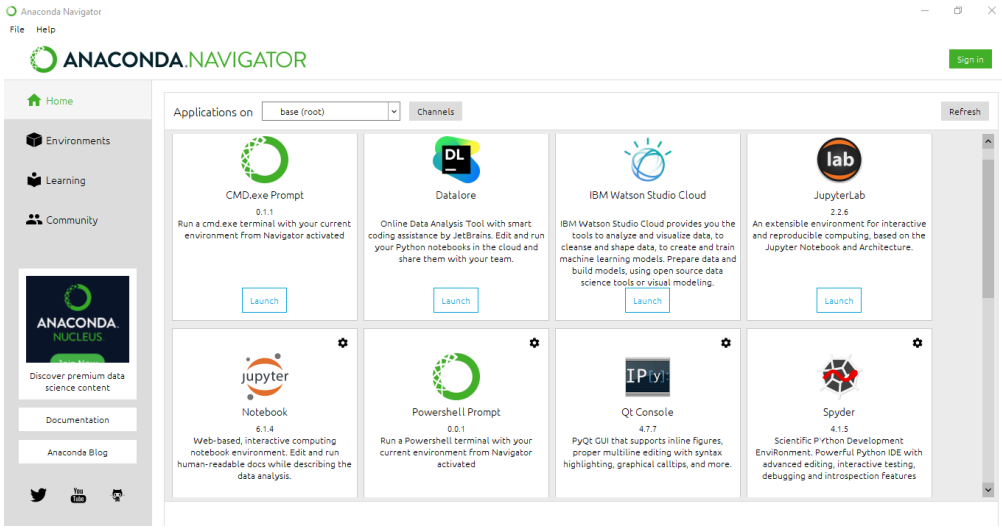


Figure 2.1 Anaconda installer

- The output above is what you will see after clicking the link, you can select the operating system you are using. In this case, I will click on Windows (64-Bit Graphical Installer (457 MB)) science I am using a windows operating system with 64-bit.
- Then your Anaconda will start downloading. After download, then you can install it.

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications without using command-line commands.



You can launch Anaconda Navigator by typing 'Anaconda Navigator' in the search bar, then double-clicking on Anaconda Navigator (Anconda3)

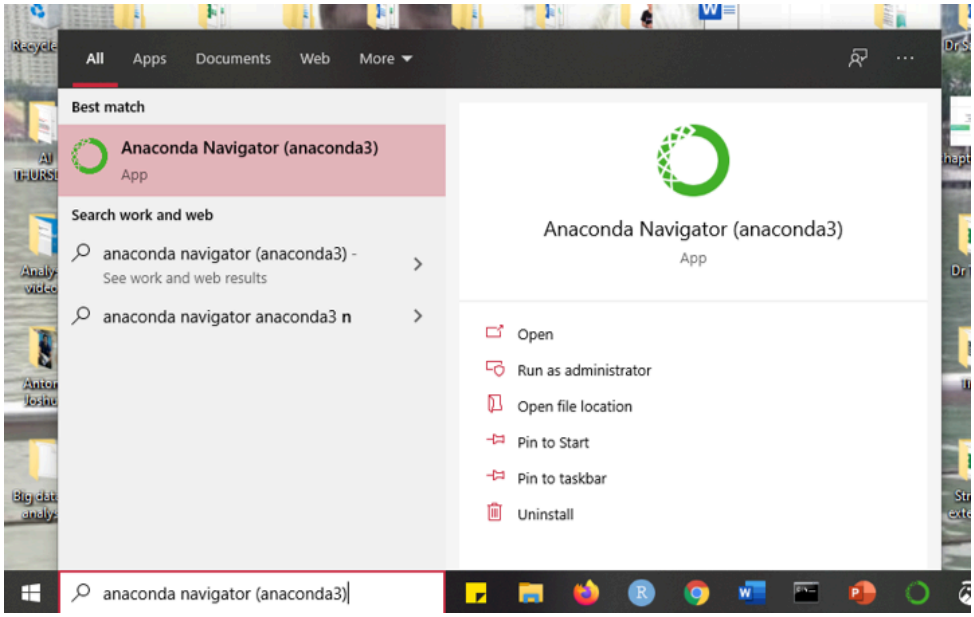


Figure 2.2 Anaconda Navigator

Session 3: Introduction to Jupyter Notebook and/or Jupyter Lab

Among the applications present in the Anaconda are JupyterLab and Jupyter Notebook

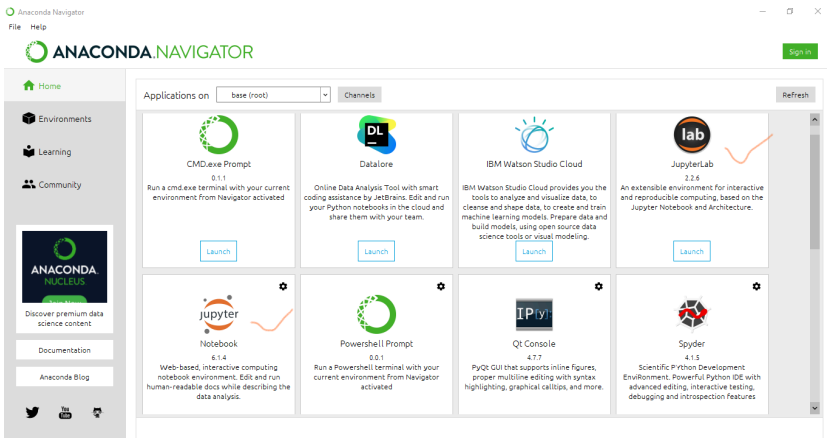


Figure 3.1 Jupyter notebook and/or jupyter lab

JupyterLab and/or Jupyter notebook is a web application that allows you to create and share documents that contain:

- live code (e.g. Python code)
- explanatory text (written in markdown syntax)
- visualizations

This notebook supports Python and other programming languages. It provides an environment for data science enthusiasts who just started out their career in data science field. This IDE supports markdown and enables you to add HTML components, images, and videos.

Difference Between JupyterLab and Jupyter Notebook

Both JupyterLab and Jupyter Notebook are browser compatible interactive python with the extension .ipynb where you can divide the various portions of

the code into various individually executable cells for the sake of better readability.

In Jupyterlab, you can create a Python script directly (.py), create a notebook (.ipynb), open a terminal, etc. Jupyter Notebook allows for only notebook (.ipynb) files while providing you the choice to choose between the two versions of Python i.e. python 2 or python 3.

JupyterLab runs in a single tab, with sub-tabs displayed within that one tab, while Jupyter Notebook opens new notebooks in new tabs.

JupyterLab can open multiple notebooks (.ipynb) files inside a single browser tab. Whereas Jupyter Notebook will create a new tab to open new notebook (.ipynb) files every time. Hovering between various tabs of a browser is tedious.

JupyterLab Interface

Many tabs in JupyterLab notebook:

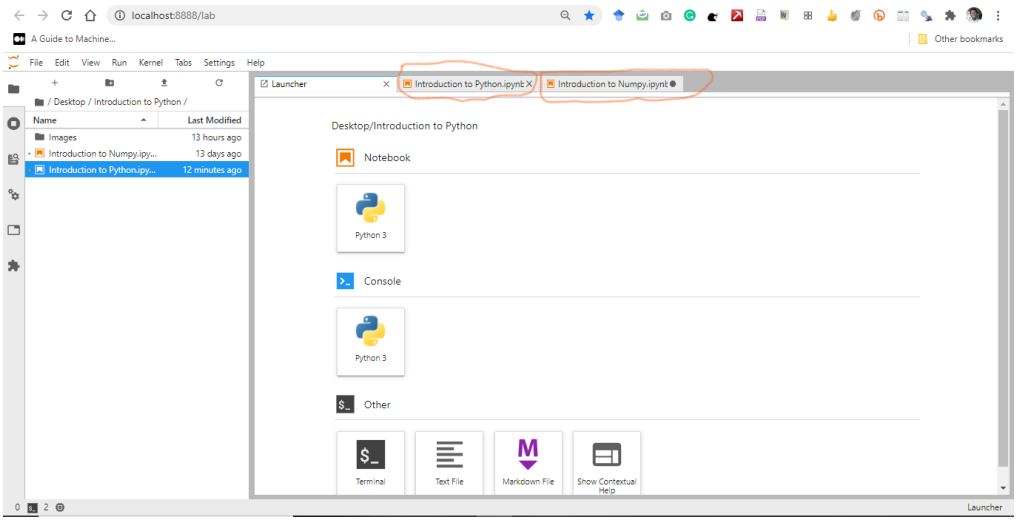


Figure 3.2 JupyterLab interface

How to launch JupyterLab

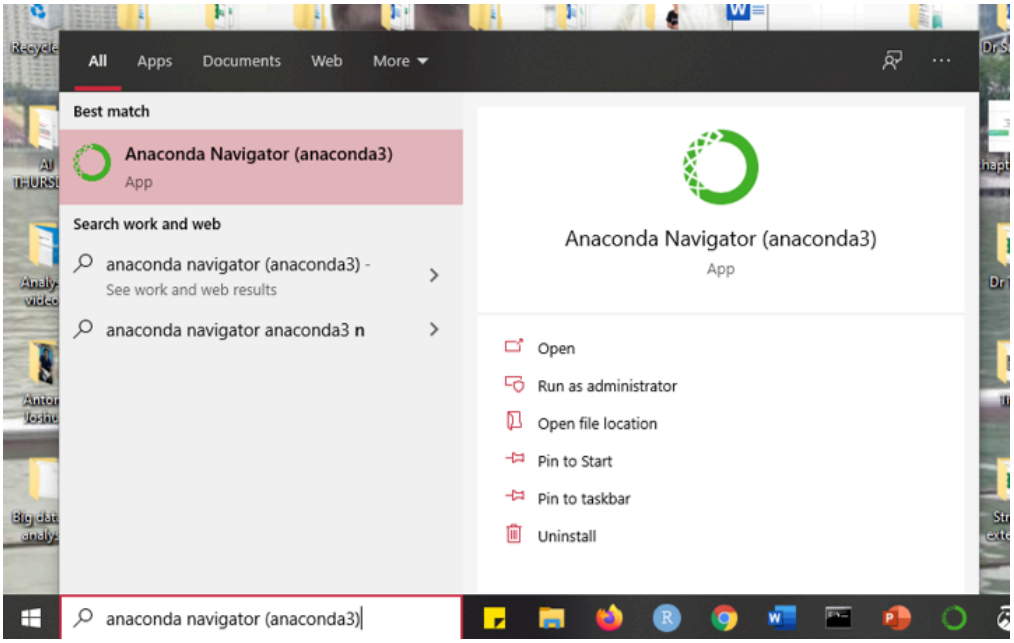
Now that you have understood JupyterLab Notebook, let's see how to launch it.

There are two ways of launching JupyterLab Notebook.

1. Anaconda Navigator
2. Through the terminal or Anaconda Prompt

Method 1: Anaconda Navigator

You can launch Anaconda Navigator by typing 'Anaconda Navigator' in the search bar



Locate JupyterLab Notebook and then click on Launch

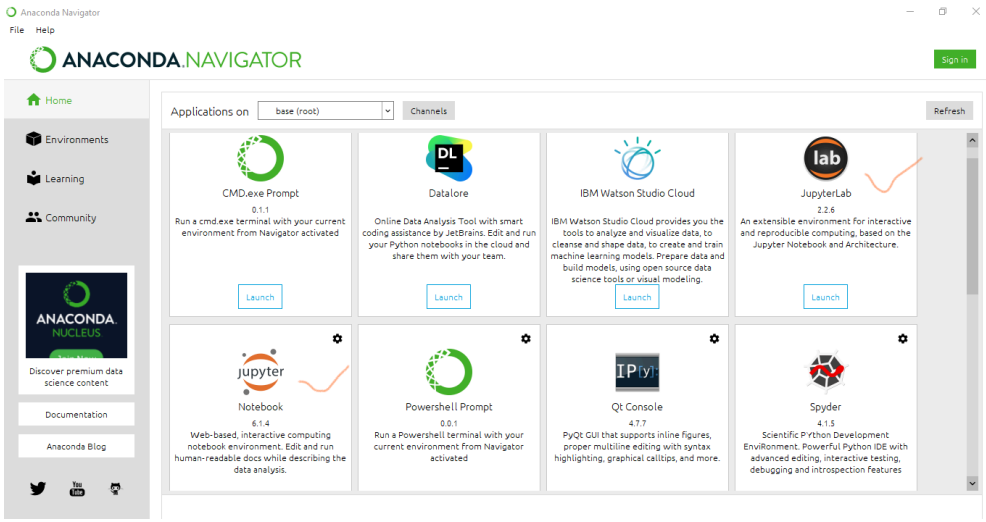
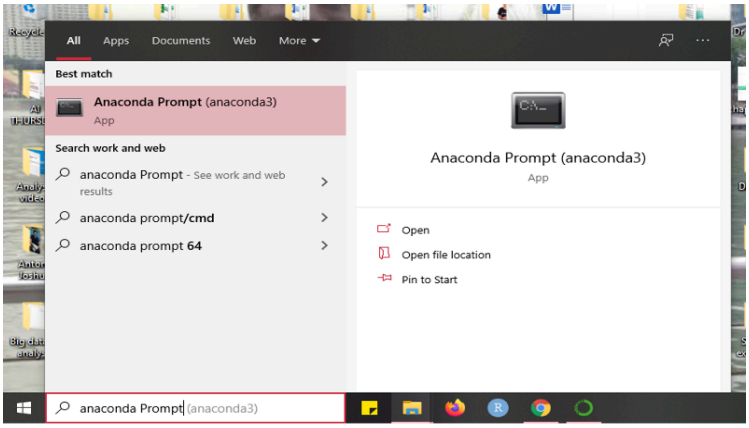


Figure 3.3 Launch anaconda navigator

Method 2: Through Terminal or Anaconda Prompt

We can also launch JupyterLab and/or Jupyter Notebook through the terminal or Anaconda Prompt. To do that, type 'Anaconda Prompt' in the search bar and double-click on Anaconda Prompt (Anaconda3).



A command prompt window will come up; then type 'jupyter lab' and press enter (or return) on your keyboard.

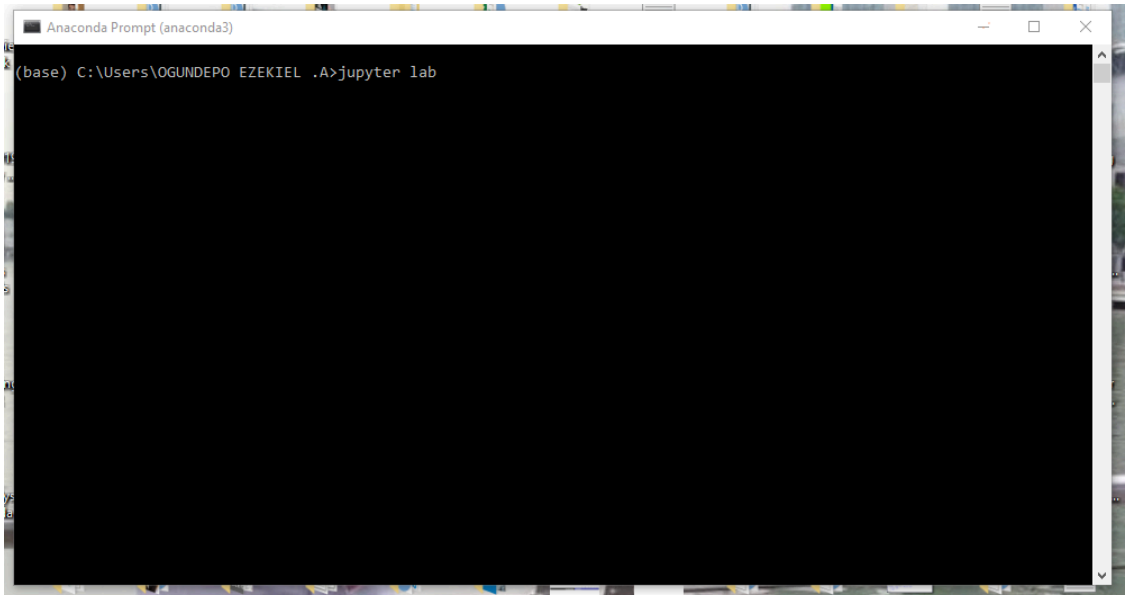
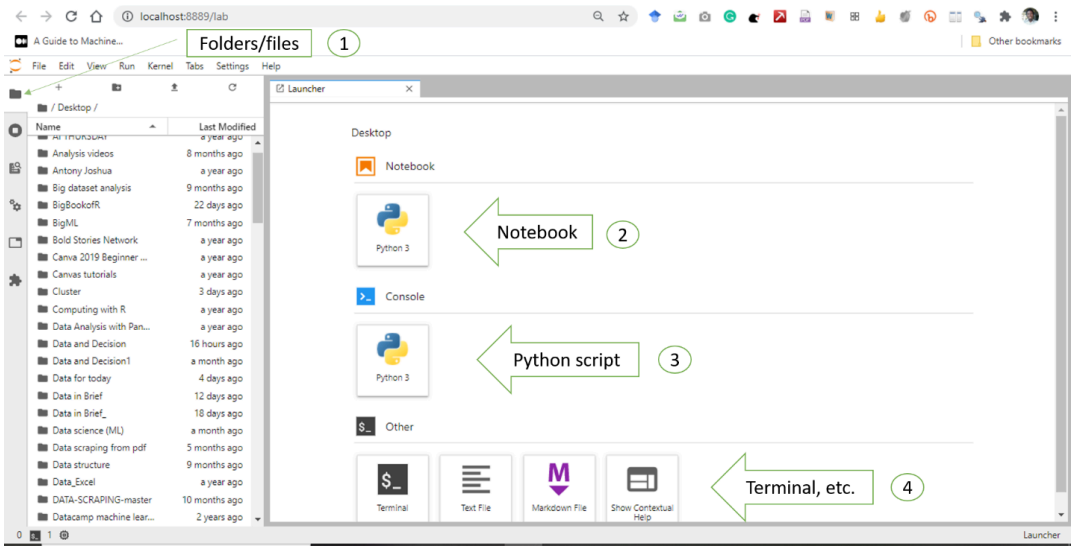


Figure 3.4 Terminal or anaconda prompt

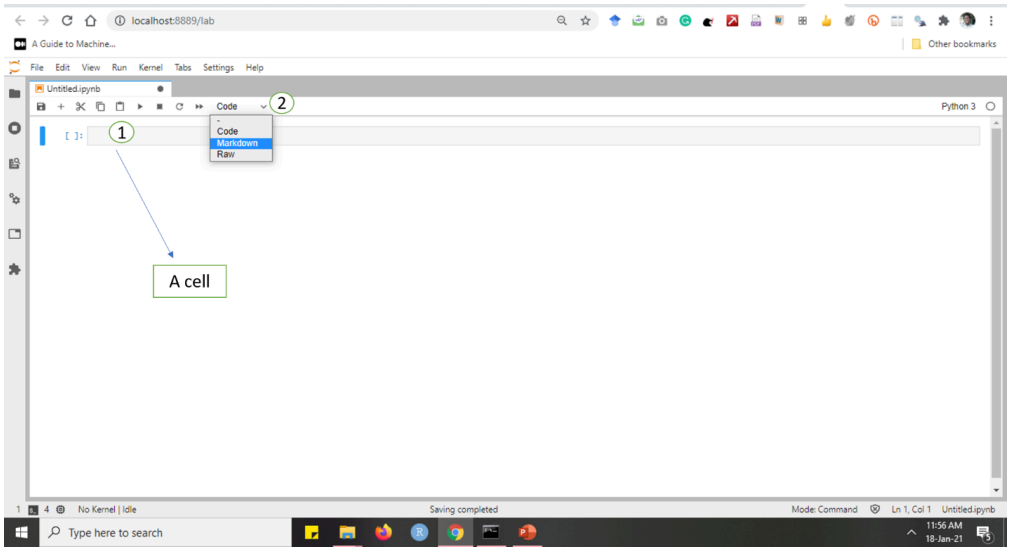
How To Navigate Through the JupyterLab Notebook Interface

After launching JupyterLab, you will see an interface that looks like this:



- 1 shows where you can select a folder or file that may contain your already worked on Jupyter Notebook (.ipynb).
- 2 shows where you can create a new notebook
- 3 shows where to create a new script
- 4 we can install some new package(s) with the terminal

Let's start by clicking on Python 3 to create a new notebook



- 1 a new cell
- 2 shows where to change the type of cell. We have Code (a pure python code), Markdown (markdown texts) or Raw (run it as it is)pure

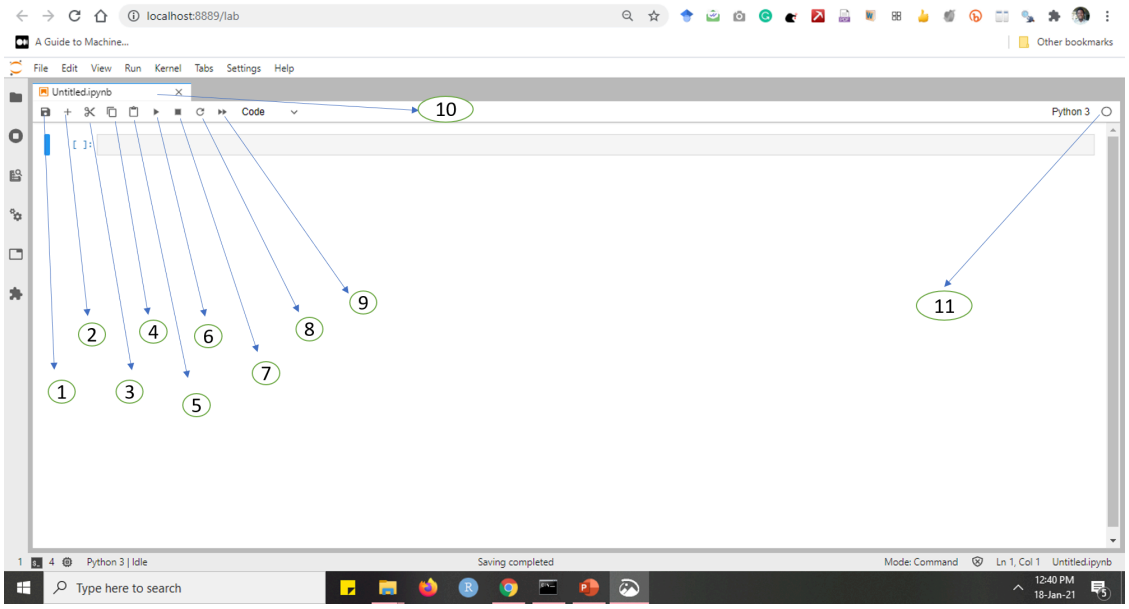


Figure 3.5 Navigate through the jupyterlab notebook interface

- 1 to save the notebook you are currently working on
- 2 to create a new cell
- 3 to cut or delete a cell
- 4 to copy a cell (which you may want to paste in another cell)
- 5 to paste a cell that was initially copied
- 6 to run the selected cell
- 7 to stop the kernel (to stop Python from working)
- 8 to restart the kernel
- 9 to restart the kernel and rerun the whole notebook
- 10 name of the notebook you are working on
- 11 status of the notebook you are working on

Working With JupyterLab Notebook

In this course, I would prefer you use the JupyterLab notebook because of the aforementioned advantages that were discussed above. If you have already closed your JupyterLab notebook, you can launch it again! In this case, I will use Anaconda prompt instead of Anaconda Navigator. Do you remember how to launch Jupyterlab notebook? If no, click here.

Notebook Cell

A cell in a Jupyter notebook can accept python code or markdown. For example, the cell that contains what is your name?' is a markdown. In markdown, you can write the full explanation of what the code does with texts mixed with simple text formatting such as boldface, italics, bulleted lists, etc.

Example of cells in Markdown:

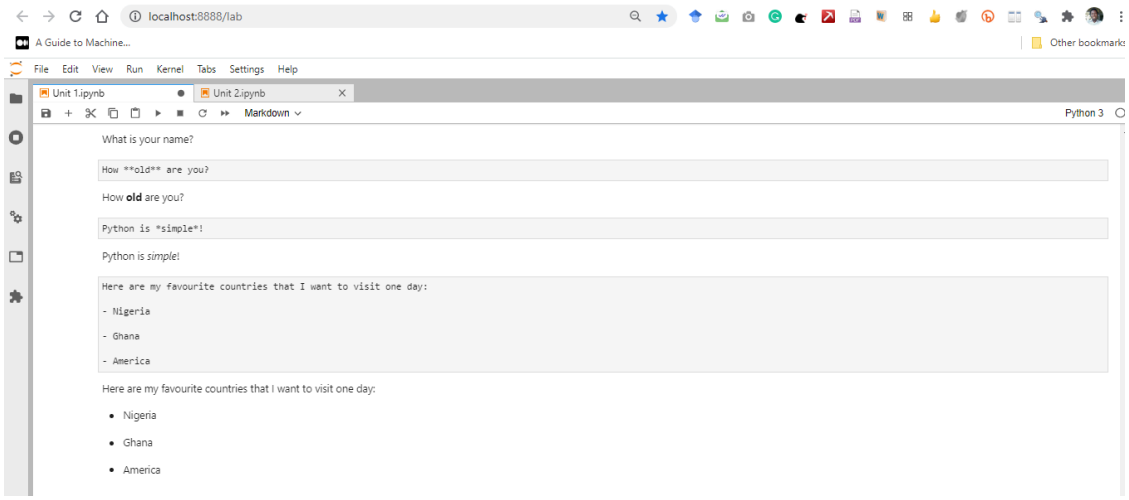


Figure 3.6 Working with jupyterlab notebook

The cell that contains 2 + 5 is a Python code and you can easily see the result (7) immediately below the cell.

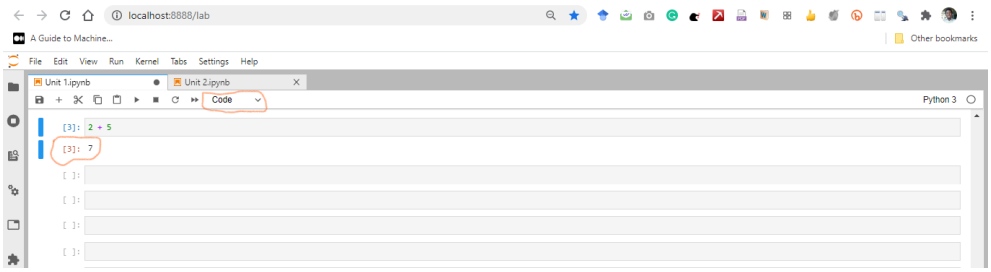


Figure 3.7 Result

Additional resources:

For more resources in this section, please consider the following:

- http://bit.ly/why-jupyter-lab
- https://www.youtube.com/watch?v=7wfPqAyYADY
- https://www.datacamp.com/community/tutorials/markdown-in-jupyter-notebook
- https://realpython.com/jupyter-notebook-introduction/

Session 4: Expression, Data Type, and Variable Assignment

Python as a Calculator

In its most basic form, Python can be used as a calculator to perform basic arithmetic operations.

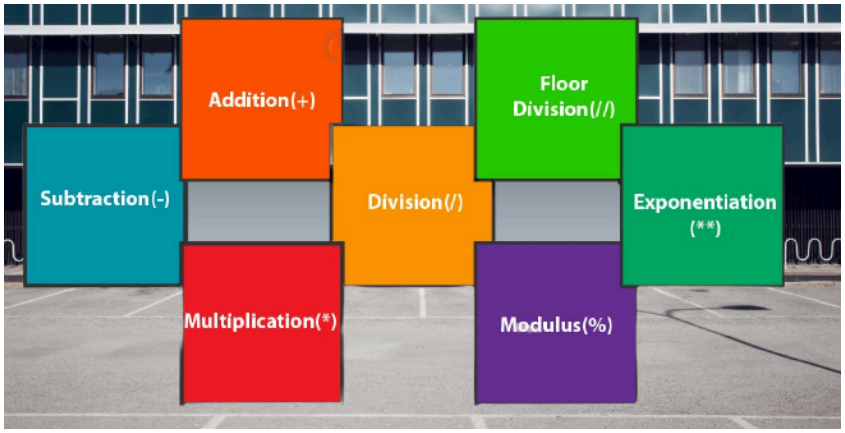


Figure 4.1 Python as a calculator

Consider the following arithmetic operations:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation (or Powers)
- Modulo

Arithmetic operation	Mathematical symbol	Python operator	Example in Python	Result
Addition	+	+	3 + 2	5
Subtraction	-	-	3 - 1	2
Multiplication	x	*	3 * 2	6
Division	÷	/	4 / 2	2
Exponentiation	2^2	**	2 ** 2	4
Parentheses/brackets	( )	( )	2 * (2 + 1)	6
Modulus	3 mod 2	%	3 % 2	1

+, -, /, **, and % are called arithmetic operators.

Expressions:

Python is well able to perform the well-known mathematical operations. An expression is an operation or a set of operations in python that can be evaluated to determine its value, i.e., it$1 + 2$ is an expression and the result 3 is a value.

Addition

Example 1: Add 8 and 2 together, i.e., 8 + 2

8 + 2

10

Example 2: Calculate 1 + 3 + 2

1 + 3 + 2 = 6

Subtraction

Example 1: Calculate 8 - 5

8 - 5 = 3

Example 2: Calculate 20 - 15

20-15 = 5

Example 3: There are 30 sweets in a box. Ezekiel picked 6 sweets from the box, Awokoya picked 8 sweets, and Emeka picked 5, while Kunle picked 4. How many sweets are remaining in the box?

30 - 6 - 8 - 5 - 4 = 7

There are 7 sweets that are still remaining in the box.

Multiplication

Example 1: What is it?

2 * 3 = 6

Example 2: What is it?

5 * 4 = 20

Example 3: What is it?

-2 * 4 = 8

Example 4: What is it?

3 x 3 = 9

```
  File "<ipython-input-11-ffb2b8152026>", line 1
    3 x 3
      ^
SyntaxError: invalid syntax
```

It is quite common in programming to make mistakes, even for programmers with years of experience. For the most part, python will tell you where you have made a mistake by providing an error message. It is important to read the message and understand where you have made a mistake and resolve it accordingly.

In this case, our error is where we have used × instead of *. Therefore, we corrected the error as shown in the cell below:

3 * 3 = 9

Division

Example 1: Calculate 10 / 5

10 / 5 = 2.0

Example 2: Calculate 200 / 20

200 / 20 = 10.0

Example 3: Calculate 11 / 2

11 / 2 = 5.5

Exponentiation (or Powers)

Example 1: Calculate 4^2

4 ** 2 = 16

Example 2: Calculate 3^2

3 ** 2 = 9

Example 3: Calculate 9^2

9 ** 2 = 81

Modulus

The modulo (or "modulus" or "mod") is the remainder after dividing one number by another. For example, $9 \ mod \ 2 = 1$. Because $9/2 = 4$ with a remainder of 1. In mathematics, we write that as 'and', $9 \ mod \ 2 = 1$ and in Python, we write it as $9 \ \% \ 2 = 1$

Example 1: Calculate 9 mod 2

9 % 2 = 1

Example 2: Calculate $11 \ \% \ 3$

11 % 3 = 2

That is, 2 is the remainder when you divide 11 by 3

Example 3: Calculate $21 \ \% \ 4$

21 % 4 = 1

Example 4: Calculate 15 % 6

15 % 6 = 3

The function divmod() in Python can be used to get the quotient and the remainder when dividing a number by another number. A quotient in mathematics is the actual result of the division of two numbers.

Example 1: When you divide 5 by 2, the actual result is 2 and the remainder is 1

divmod(5, 2)

(2, 1)

The quotient is 2 and the remainder is 1

Class Activity 1:

What is the result of $divmod(18, 4)$?

The Floor Division (//)

The floor division (//) rounds the result down to the nearest whole number

Example 1: This is a normal division

15 / 2 = 7.5

Example 2: This is a floor division

15 // 2 = 7

Class Activity 2:

Calculate the following in Python:

- 2 + 6 - 12
- 4 * 3 - 8
- 81/6
- 9 mod 2
- 16 % 3
- 2^3
- divmod(17, 5)
- 11 // 5

Parentheses or Brackets

Parentheses are used to denote the grouping of operations in mathematics. It denotes modifications to the normal order of operations. Remember BODMAS in mathematics? We shall use BEDMAS in programming:

B - Bracket

E - Exponentiation

D - Division

M - Multiplication

A - Addition

S - Subtraction

In an expression like this, the part of the expression within the parentheses is evaluated first, and then this result is used in the rest of the expression, i.e. $3\times5 = 15$.

Example 1: 3 * (2 + 3) = 15

Example 2

$2^2 + 1$

2**2 + 1 = 5

Example 3: $(3 + 2) \times (6 - 4)$

(3 + 2) * (6 - 4) = 10

Example 4: Use python to evaluate $2(3 + 1)$

2(3+1)

```
<>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
<ipython-input-30-f57bf854ff11>:1: SyntaxWarning: 'int' object is not callable; perhaps you missed a comma?
  2(3+1)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-f57bf854ff11> in <module>
----> 1 2(3+1)

TypeError: 'int' object is not callable
```

This throws an error because we need to add an operator (*) before the parentheses or brackets.

2 * (3 + 1) = 8

There must be an operator (+, –, *, /) before and/or after the parentheses.

Example 5: What is the value of $3 \times (3 + 2)$?

3 * (3 + 2) = 15

Example 6: Calculate $(8 - 7) \times 2$

(8-7) * 2 = 2

Example 7: What is the value of (6 - 4) + 2?

(6 - 4) + 2 = 4

Example 8" What is the value of $(3 + 2)\times(6 - 4) + 2$?

(3 + 2) * (6 - 4) + 2 = 12

Comment in Python

Adding comments to your code is a common practice in all programming languages.

This is important because:

1. It makes you remember a line of code you have written after leaving it for a while
2. It makes others who want to check your code understand it line by line
3. It is good for documentation purpose

In order to add comments to your code, you will use the # sign before writing your comments.

Comments are not run as Python code, so they will not influence your result. Python ignores anything that has a # sign.

Example 1

```
# This adds 2 and 8 together. The result will be 10
2 + 8 = 10
```

As you can see, This adds 2 and 8 together. The result will be 10 as our comment and we do that by using the # sign.

Example 2

```
# You will know how to import packages in Python later
# to get the square root of a number, we will import numpy package and then use the numpy.sqrt() function
import numpy
numpy.sqrt(4)
2.0
```

As you can see, we use # to tell you what you are about to learn and how to get the square root of a number by using a comment.

Comparison Operators

We can compare two or more values by using comparison operators. These types of operators compare one value to another based on a condition. The result can either be True or False, known as Boolean (bool). The following are the most common comparison operators in python:

- double equal to ==
- not equal to !=
- greater than >
- less than <
- greater than or equal to >=
- less than or equal to <=
- Double equal operator ==

This operator returns True if the two values are equal; otherwise, it returns False

Is 5 equal to 3?

5 == 3

False

As you know, the answer is False.

Is 5 equal to 5?

5 == 5

True

Yes! 5 is equal to 5.

Below we compare whether two numbers i.e. 5 are equal

5 = = 7

False

Since the two values are not equal, then the result is False

Class Activity 3

Write Python code to compare 5 and 6

Pilot answer 2

5 == 6

Not equal to operator !=

This operator returns True if the two values are not equal to each other; otherwise, it returns False.

Below we compare whether 5 is not equal to 10

5 != 10

True

Below we compare whether 200 is not equal to 100

200 != 100

True

Greater than operator >

This operator returns True if the value on the left side of the operator ⊙ is bigger than the value on the right side of the operator and False if otherwise.

is 100 greater than 30?

100 > 30

True

Since 100 is greater than 5, the result is True

is -60 greater than 50?

-60 > 50

False

The answer is False. 50 is greater than -60

is -6 greater than -5?

-6 > -5

False

As you can see, it -6 is less than -5.

Less than operator <

This operator returns True if the value on the left side of the operator (<) is smaller than the value on the right side of the operator and False if otherwise.

is 100 less than 1000

100 < 1000

True

The result of the above expression is True since 100 is less than 10000

is -6 less than -5?

-6 < -5

True

As you can see, −6 is less than −5 .

Greater than or equal to operator >=

This operator returns True if the value on the left side of the operator (>=) is bigger than or equal to the value on the right side of the operator and False if otherwise.

Below we compare whether 100 is greater than or equal to 1000

100 >= 1000

False

Since 100 is not greater than or equal to 1000, the result is False

Below we compare whether 60 is greater than or equal to 45

60 >= 45

True

We compare whether 100 is greater than or equal to 100

100 >= 100

True

Less than or equal to operator <=

This operator returns True if the value on the left side of the operator (<=) is smaller than or equal to the value on the right side of the operator and False if otherwise.

is 100 less than or equal to 1000?

100 < = 1000

True

Yes, 100 is less than or equal to 1000

Rules for naming variables

- All variables must begin with a letter of the alphabet
- After the initial letter, variable names can also contain an underscore (_), a period (.) and/or a number. No spaces or special characters, however, are allowed
- Uppercase strings are different from lowercase strings in Python

Example

	Sample of acceptable variable names	Unacceptable variable names
1	grade.test	Grade(Test)
2	test_grade	test grade
3	term2	2 term
4	Sale_price_2021	2021sales_price
5	sudan_students	sudan&students

Data Types in Python

Since Python is an object-orientated programming language. That is, we have many different objects in Python. The most common ones are strings, integers, floats, and Boolean.

- Words or texts in Python are known as strings (str for short), e.g., “Miva”, “Male”, “Presidential Election”
- Numbers without decimals are known as integers (int for short), e.g. – 1, 0, 9

- Numbers with decimals are known as floats, e.g., $3.142, -1.6, 4.2$
- Values with either True or False are known as Boolean (bool for short)

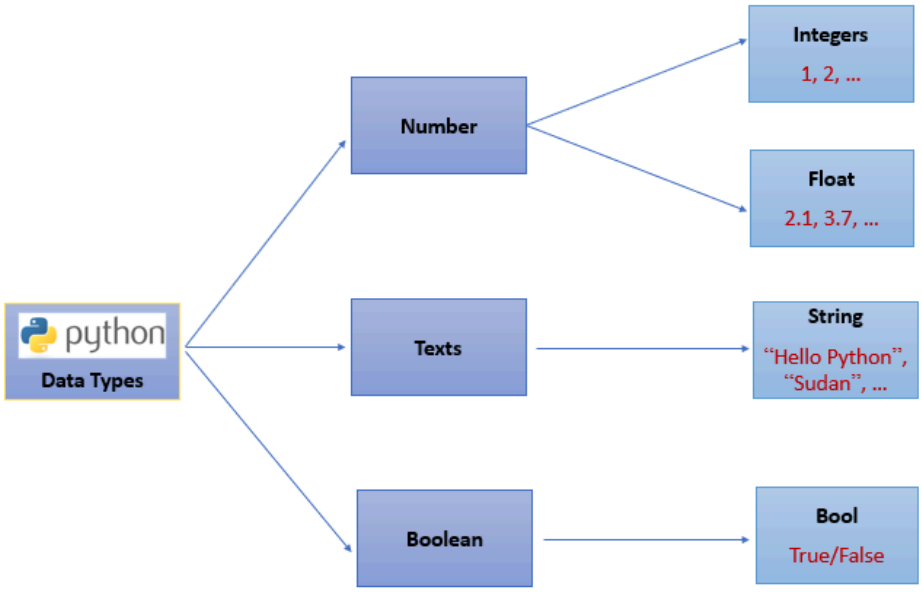


Figure 4.2 Data types in python

Python can tell you the data type by using the type() function. Python refers to integer numbers as 'int', floats as characters or text as 'str' and Boolean as 'bool'.

Integer Data Type

Integers are numerical (number) values that are positive or negative whole numbers without decimals

Examples

type(20)

<class 'int'>

type(2)

<class 'int'>

type(2021)

<class 'int'>

Float data type

Float (floating point) numbers are numbers with a decimal point.

Examples

type(3.142)

<class 'float'>

type(11.124)

<class 'float'>

type(-2021.2)

<class 'float'>

String data type

Characters or texts are known as strings in Python. A string object must be inside a single or double quotation. stringswise, Python will throw an error.

Example 1

"I am from Lagos."

'I am from Lagos.'

type("I am from Lagos")

<class 'str'>

As you just learnt, this is a string (str) data type.

Example 2

```
  File "<ipython-input-59-eeefa5526eaf>", line 1
    I am from Kenya
      ^
SyntaxError: invalid syntax
```

I am from Kenya

There is a SyntaxError because we don't put 'I am from Kenya in a quotation. Now, Let's correct the error by putting the texts in double quotation marks.

"I am from Kenya."

'I am from Kenya.'

type("I am from Kenya")

<class 'str'>

Strings can be represented by both single quotes '' and double quotes " ". However, you can't mix them, i.e., " ' or ' ".

Example 3

"Python is so simple."

'Python is so simple.'

It is common to see most Python users using the function print() in their line of code. For now, all you need to know is that it instructs python on what to print.

print("Hello, I am from Lagos and I can speak Yoruba.")

Hello, I am from Lagos and I can speak Yoruba

print(type("Hello, I am from Lagos and I can speak Yoruba."))

<class 'str'>

print(4+6)

10

print(type(14.6))

<class 'float'>

Boolean data type

A Boolean or bool, is an object in python that can take two values i.e. True or False.

Values of a Boolean must always start with a capital letter and then be followed by small letters. This allows the programming language to identify the value as a Boolean. When we check the type of True or False, the result is a bool, which represents the Boolean data type.

Example 1

Check for the data type of a Boolean True

To check the data type, type

type(True)

<class 'bool'>

Example 2

Check for the data type of a Boolean False

To check the data type, type

type(False)

<class 'bool'>

Some operations can also result in a Boolean data type. For example:

8 > 6

True

6 < -10

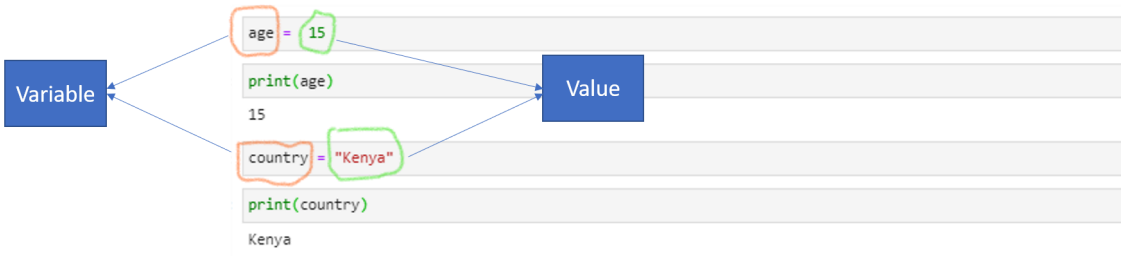
False

"Lagos" != "Abuja"

True

Variable Assignment

In the simplest terms, a variable is a way in which a programmer stores a value so that it can be used over and over without having to repeat oneself. A variable is just like a container that stores a value. For example, In the image below, age is a variable that stores the value of 15 or the value 15 is stored into the variable age. Also, country is a variable that stores or holds the value of "Kenya". The equal operator (sign) is called an assignment.



Example 1

age = 15

To access what is in the variable age, you can do that by putting the name of the variable in the print() function or simply typing the name of the variable without a print() function.

print(age)

15

age

15

We can also access the kind of data type in the variable age.

```
type(age)
<class 'int'>
```

The value stored in the variable age is an integer (int) data type

Example 2

```
country = "America"
print(country)
America
type(country)
<class 'str'>
```

The value stored in the variable 'country' is a string (str) data type

Class Activity 4:

- What is your name? Assign the value to the variable name
- How old are you? Assign the value to the variable age
- Are you a male or female? Assign the value to the variable gender
- What is the name of your country? Assign the value to the variable country
- What language can you speak? Assign the value to the variable language
- True or False: do you smile today? Assign the value to the variable smile

The list of the variables to create and assign values to is:

- name
- age
- gender
- country
- language
- smile

Programming is about automating mundane tasks that otherwise would have taken ages! It also makes sense for the programmer to make their work easier by having a single representation of knowledge at only one point of the programme. This in turn makes it easier to understand code and reduces the number of lines one has to write and hence saves time!

When a value is stored in a variable, you can then use that variable to do further calculation

Example 1

```
# Age now
age = 15
Age = 15
# Next year age =
age + 1
16
# Age last two years:
age - 2
13
```

Example 2

```
# Assigning values to Variables
x = 2,
y = 3,
z = 4
x + y
5
z - x - y
-1
x * z
8
x ** 2
4
z % y
1
divmod(z, x)
(2, 0)
```

Example 3: We can assign multiple values to multiple variables at the same time in Python

```
name, age, gender, country = "Jamal", 16, "Male", "Nigeria"
print(name)
Jamal
print(age)
16
```

print(gender)

Male

print(country)

Nigeria

Rules for Naming Variables

- All variables must begin with a letter of the alphabet
- After the initial letter, variable names can also contain an underscore (_), a period (.) and/or a number. No spaces or special characters, however, are allowed
- Uppercase strings are different from lowercase strings in Python

Example

	Sample of acceptable variable names	Unacceptable variable names
1	grade.test	Grade(Test)
2	test_grade	test grade
3	term2	2 term
4	Sale_price_2021	2021sales_price
5	sudan_students	sudan&students

It is a good practice to give meaningful variable names so that someone else or you in the future can understand what you were trying to accomplish. The best naming convention is to choose a variable name that will tell the reader of the program what the variable represents.

Class Activity 5: Discuss among your colleagues why some variable names in the examples above are acceptable and why some are not acceptable in Python.

How to get information about a function?

If you want to become a good programmer, it is essential to read function documentations. This is like a user guide or manual about a specific function. There are many ways to get help in Python, but the simplest way is to question the function by using a question mark. For example, to read the help file or documentation of the print() function, we shall use ? print

?print

```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

It prints the value of an object.

Let's see what the float() function does by using ? float

?float

```
Init signature: float(x=0, /)
Docstring:      Convert a string or number to a floating point number, if possible.
Type:           type
Subclasses:
```

It converts a string or number to a floating-point number, if possible.

Figure 4.3 Warning

Don't use parentheses in the function when seeking for help in Python. For example, if you need help on what the int() function does, just type ?int without parentheses. help

?int

```
Init signature: int(self, /, *args, **kwargs)
Docstring:
int([x]) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given.  If x is a number, return x.__int__().  For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base.  The literal can be preceded by '+' or '-' and be surrounded
by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
Type:           type
Subclasses:     bool, IntEnum, IntFlag, _NamedIntConstant, Handle
```

If you include parentheses, you will have the following:

```
?int()
```

```
Object `int()` not found.
```

```
?print()
```

```
Object `print()` not found.
```

As you can see, that is not what you are looking for.

Data Type Conversions

Python allows us to convert objects from one data type to another. For example, you may need to convert a float to an integer (int). In programming this is called typecasting and has a lot of very many useful applications, as will be shown later.

When we convert an integer into a float, we don't really change the value (i.e., the significand) of the number. However, if we cast a float into an integer, we could potentially lose some information. For example, if we cast the float 1.1 to an integer, we will get 1 and therefore lose the decimal information (i.e. 0.1)

Example 1

Let us convert an integer 9 into a float, i.e., int to float. First assign the value of 9 to a variable nigeria_naira.

```
nigeria_naira = 9
```

Let us check the data type in the variable nigeria_naira by using the type() function

```
type(nigeria_naira)
<class 'int'>
```

Now that we are sure that the value in the variable nigeria_naira is an integer (int) data type. So, we can convert it to a float data type by using a float() function. Save the result as nigeria_naira_float

```
nigeria_naira_float = float(nigeria_naira)
type(nigeria_naira_float)
```

<class 'float'>

As you can see, we have cast the integer (int) to a float data type

Example 2

Let us try and convert a float to an integer. As we said above, we will end up losing information, so you should be careful about this type of conversion.

Convert 9.345 to an integer

int(9.345)9

The result of the above code is 9, i.e., we lose 0.345 which is a lot of information. For example, if it was an amount of money transfer, it would be potentially wrong to report that the amount was 9 and not 9.345.

Example 3

We can also convert a pure number that is recognised as a string (str) due to one reason or another to a float or an integer. This is only possible when the string looks like an int or a float otherwise python will throw an error

Convert a string into an integer

age = "20"

type(age)

<class 'str'>

Because the value of age is in a quotation, therefore, Python sees this as a string. We can convert the value in age to an integer by using the int() function.

Converting an imperfect string to an integer

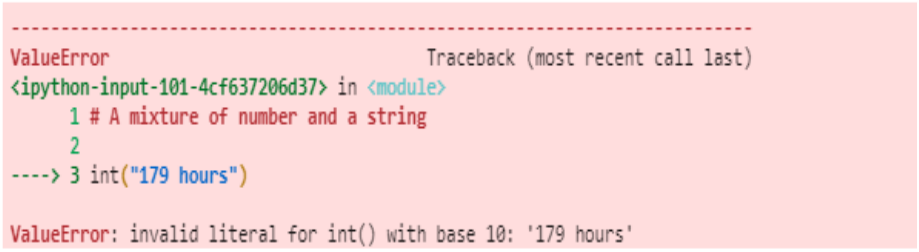
```
new_age = int(age)

type(new_age)

<class 'int'>

# A mixture of a number and a string

int("179 hours")
```



```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-101-4cf637206d37> in <module>
      1 # A mixture of number and a string
      2
----> 3 int("179 hours")

ValueError: invalid literal for int() with base 10: '179 hours'
```

This throws an error because this is not a pure number. Other data types can also be converted to strings by using the str() function, e.g.

```
# Convert a float to a string:

str(9.7)

'9.7'

# Convert an int to a string:

str(6)

'6'
```

Example 4

In our introduction to the Boolean data type, we said that Boolean can take two values: True or False. It is often common in practice to represent 1 as True and 0 as False. This can be seen when we convert Booleans to integer (int) or float.

```
# Convert False to integer

int(False)
```

```
0
# Convert True to integer
int(True)
1
# Convert integer 1 to bool
bool(1)
True
# Convert integer 0 to bool
bool(0)
False
```

Logical Operators

In the last section we talked about some mathematical operators such as +, -, *, /, **, and % and comparison operators such as ==, !=, >, >=, <, and <=. In this section, you will learn about logical operators.

Logical operators test whether an expression is True or False. There are three types of logical operators namely and, or, not.

- and - This operator returns a False when it encounters a False condition
- or - This operator returns a True when it encounters at least a True condition
- not- This operator returns True if the condition is False and True if the condition is False

Example 1: Using and operator

We define a variable 'mins' to represent the number of minutes a money transfer transaction in GTB takes to be completed.

mins = 3

Run this cell to see the outcome

mins < 7 and mins < 1

False

Since the two expressions are both False i.e.

mins < 7 is False

mins < 1 is False

Therefore, mins < 7 and mins < 1 will be False

To cross-check

False and False

False

Example 2: Using the 'or' operator

We define a variable 'year' to represent the number of years Panam Pancy Paul, a Jos man, has been living in Abuja

year = 12

Run this cell to see the outcome

year =integersr year < 10

True

Since one expression is True i.e.

year == 12 is True

year < 10 is False

Therefore, year == 12 or year < 10 will be True

Class Activity 5 (Peer to peer review activity)

What will be the result of the following expressions?

minute = 18

minute < 7 and minute < 1

Example 3: Using the not operator

We define a variable 'mins' to represent the number of minutes a money transfer transaction in Zenit bank takes to be completed.

mins = 10

mins > 5

True

not mins > 5

False

As you know, anything that is not True is False! For example:

not True

False

not False

True

Summary of Python Programming

In this study session, you have learnt that:

1. Python can be used as a calculator to do some arithmetic operations
2. Data type includes integer, float, string, and Boolean
3. The Boolean data type can either be True or False
4. String data type can be created by using either single quotes (‘’) or double quote “ ”.
5. Python makes use of # to make a comment
6. Comparison operators include ==, !=, >, <, >=, and <=.

Session 5: R Programming language

Lesson Outline:

- History and Overview of R
- R for Data Science
- R as a Calculator
- Atomic Data Type and Variable Assignment in R
- Basic Data Structure in R

History and Overview of R

R is a programming language and free software environment for statistical computations, data cleaning, data analysis, and graphical representation of data. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. In the year 2020, the R community celebrated the 20th year of R version 1.0.0. The current version of R programming is 4.3.1 with the nickname Beagle Scouts.

Why R programming?

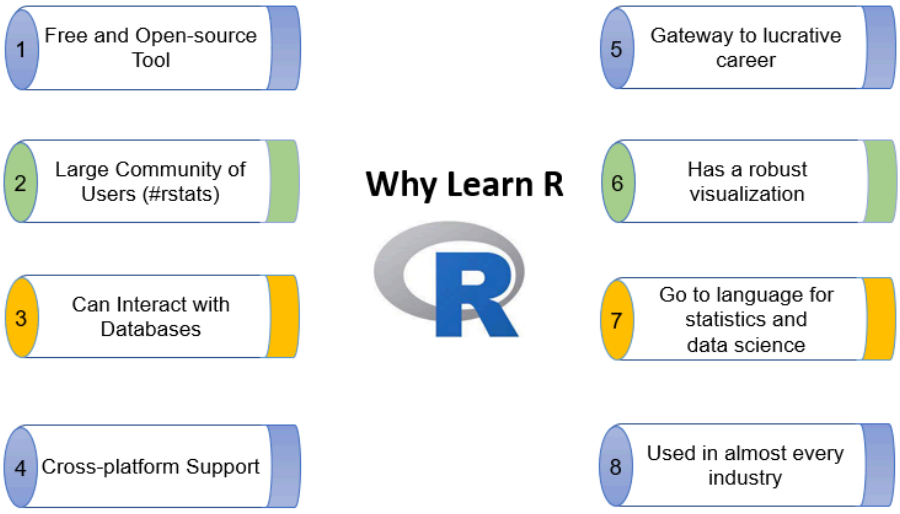


Figure 5.1 LearnR, useR, and thinkR

Figure 5.2 Companies that use R for analytics

R Learning Curve

As many have said, R makes hard things easy. The learning curve for R is pretty steep for a beginner. Though R is a bit difficult in the beginning, data science enthusiasts still prefer to learn it due to the amazing features of R.

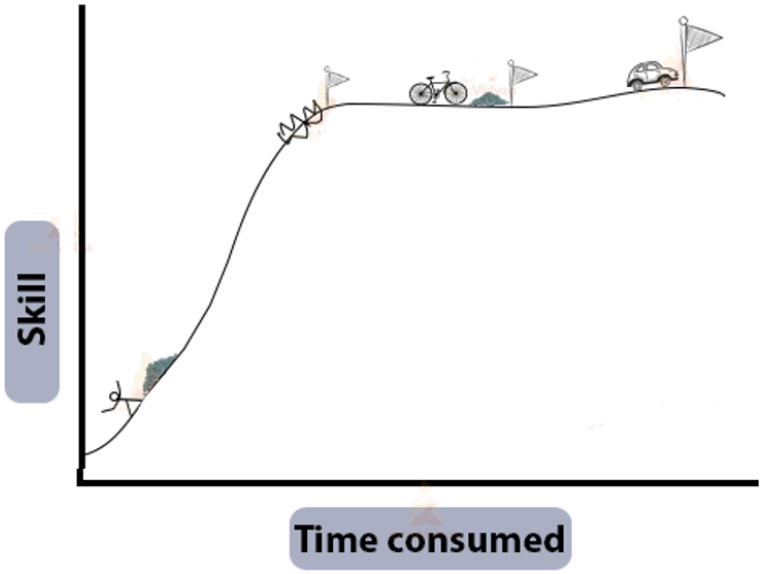


Figure 5.3: R learning curve

R programming Interface

RStudio

RStudio is an integrated development environment (IDE) for R programming. RStudio makes programming easier and friendlier in R.

The current version of RStudio is 1.4.1717

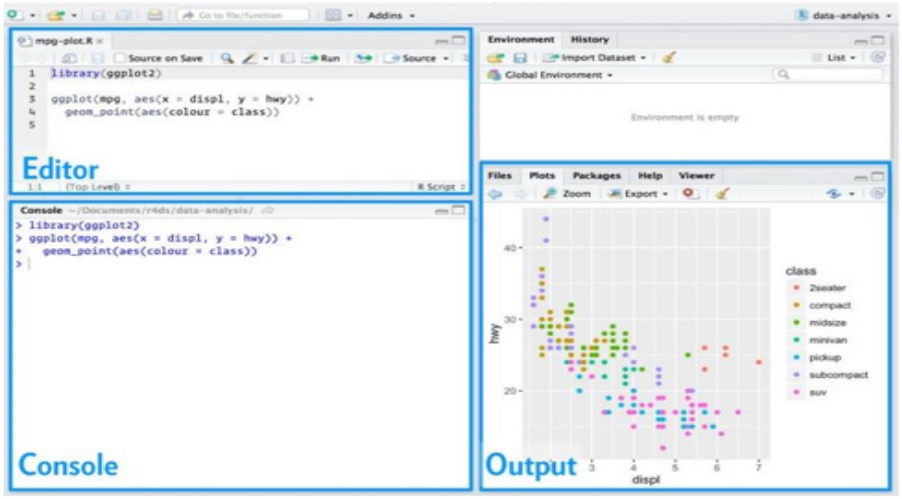


Figure 5.4 R programming interface

Installing R and RStudio

You can install R on any operating system (OS).

For Windows:

- Download R for Windows OS from this link
- Open the .exe file and install it.

For Mac:

Download R for Mac via this link. Please make sure you click on the first one. pkg link,

- Open the .pkg file and install it.

For Linux:

For complete R System installation in Linux, follow the instructions on this link.

For Ubuntu with Apt-get installed, execute sudo apt-get install r-base in the terminal.

You can install RStudio via this link, choose the appropriate installer file for your operating system, download it, and then run it to install RStudio.

R for Data Science

What is Data Science?

Data science is an exciting discipline that allows you to turn raw data into understanding, insight, and knowledge.

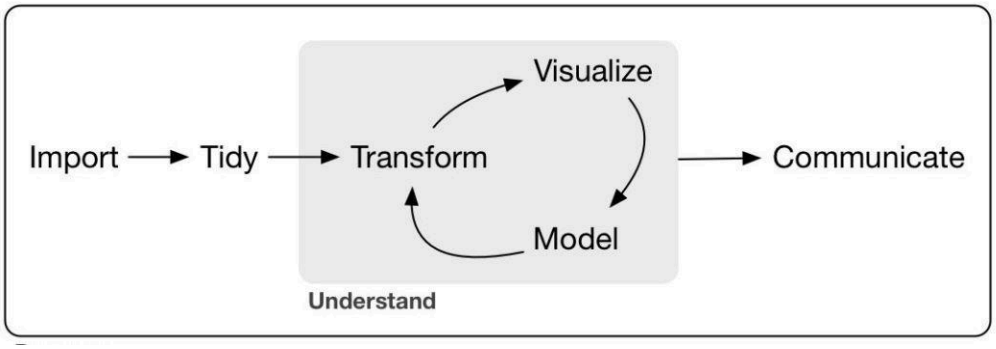


Figure 5.5 What is data science?

Data Science Phases – Inspired by Hadley

R is learning R Materials

You can learn R through the following useful selected materials:

- YaRrr! The Pirate's Guide to R
- R for Data Science
- R for Data Science: Exercise Solutions
- Big Book of R
- Posts you might have missed.

Introduction to R

In this section, you will take your first steps with R. You will learn how to navigate through the four windows in Rstudio, use R as a calculator and assign values to variables. You will also get to know the basic data types in R.

The four panes of RStudio:

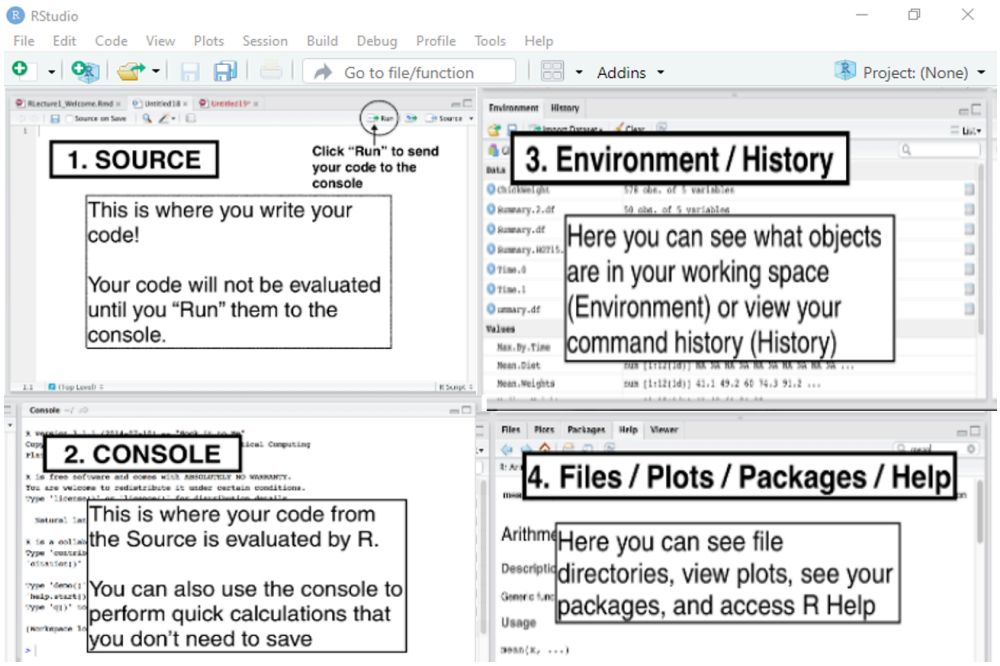


Figure 5.6 The four panes of Rstudio

Let's open our RStudio!

R as a calculator

Figure 5.7 R as a calculator

R as a Calculator

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operations:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Exponentiation (^)
- Modulo (%%)
- Parenthesis ()

Addition and Subtraction

6 + 12 - 8

[1] 10

56 - 14 + 100

[1] 142

1 + 3 + 2

[1] 6

7 + 15 - 10 + 18 - 6 + 11 - 14

[1] 21

2 + 3 + 6

[1] 11

Multiplication and Division

```
2 * 3
```

```
[1] 6
```

```
100 / 50
```

```
[1] 2
```

```
3 * 5 / 3
```

```
[1] 5
```

```
-2 * 4
```

```
[1] -8
```

```
8 * -5
```

```
[1] -40
```

Exponentiation

```
3^2
```

```
[1] 9
```

```
2 * 2^3
```

```
[1] 16
```

```
2^3 - 9
```

```
[1] -1
```

```
2 - 2^3
```

```
[1] -6
```

```
3^2 - 1 * 4 + 2
```

```
[1] 7
```

Modulus

The modulo (or “modulus” or “mod”) is the remainder after dividing one number by another. For example, 9 mod 2 = 1. Because 9/2 = 4 with a remainder of 1. In mathematics, we write that as 9 mod 2 = 1 and in R, we write it as 9 %% 2 = 1.

```
9 %% 2
```

```
[1] 1
```

```
11 %% 3
```

```
[1] 2
```

```
11 %% 3 - 4
```

```
[1] -2
```

```
16 %% 3
```

```
[1] 1
```

Parenthesis or Brackets

Parentheses are used to denote the grouping of operations in mathematics. It denotes modifications to the normal order of operations. Remember BODMAS in mathematics? We shall use BEDMAS in programming:

- B - Bracket
- E - Exponentiation
- D - Division
- M - Multiplication
- A - Addition
- S - Subtraction

In an expression like this, the part of the expression within the parentheses is evaluated first, and then this result is used in the rest of the expression, i.e., $3 \times 5 = 15$.

Examples

$3 \times (2 + 3)$

3 * (2 + 3)

[1] 15

$(3 + 2) \times (6 - 4)$

(3 + 2) * (6 - 4)

[1] 10

$6 - (5 \times 1) + 2^3$

6 - (5 * 1) + 2^3

[1] 9

Operations involving square root

Use R to evaluate $\sqrt{125}$

sqrt(125)

[1] 11.18034

Use R to evaluate $\frac{19}{\sqrt{19}}$

19 / sqrt(19)

[1] 4.358899

Evaluate $\frac{2\sqrt{3}}{3\sqrt{10}}$

(2 * sqrt(3)) / (3 * sqrt(10))

[1] 0.3651484

Comment in R

R makes use of the # sign to add comments. Adding comments to your code is a common practice in all programming languages. This is important because:

1. It makes you remember a line of code you have written after leaving it for a while.
2. It makes others who want to check your code understand it line by line.
3. It is good for documentation purposes.

Comments are not run as R code, so they will not influence your result and R ignores anything that has a # sign.

Examples:

```
# We use * for multiplication in R

2 * 8
```

```
[1] 16
```

It is important to put a space after the # for easy readability.

```
3 + 6 # a space after #
```

```
[1] 9
```

Comparison Operators

We can compare two or more values by using comparison operators. These types of operators compare one value to another based on a condition. The result can either be TRUE or FALSE, known as logical. The following are the most common comparison operators in R:

- double equal to (==)
- not equal to (!=)
- greater than ⊙
- less than (<)
- greater than or equal to (>=)
- less than or equal to (<=)

Examples

```
5 == 3
```

```
[1] FALSE
```

```
25 != 10
```

```
[1] TRUE
```

```
100 > 30

[1] TRUE

-6 < -5

[1] TRUE

60 >= 45

[1] TRUE

100 <= 1000

[1] TRUE

0 < -1

[1] FALSE

-1 > 1

[1] FALSE
```

Atomic Data Type in R

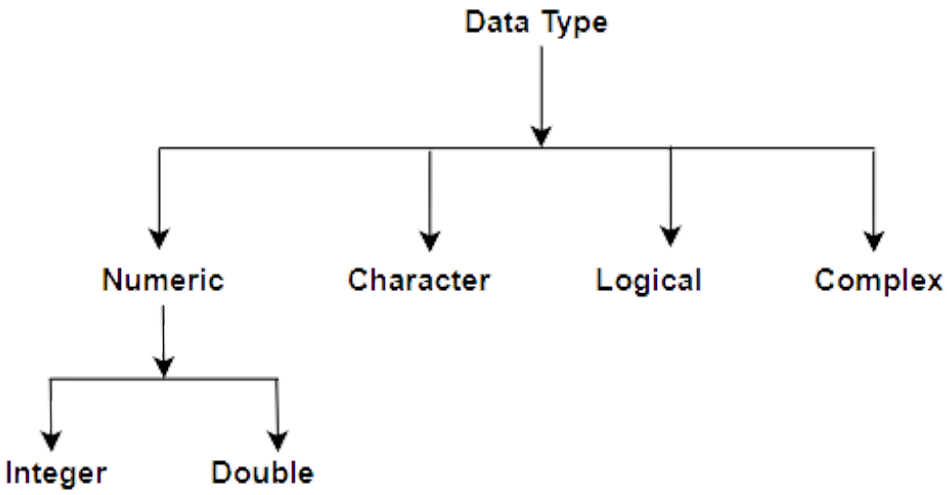


Figure 5.8 Atomic data type in R

Classes of Objects in R

R works with numerous atomic classes of objects. The most basic atomic data types are:

- Numeric, such as integer (4, -2) or float (4.7, -0.26)
- Characters such as texts (or strings) must be enclosed within either single or double quotes. e.g., “Nigeria”, “COVID-19”, “baby”, “Hello world”.
- Logical, such as TRUE or FALSE

We use the class() function to check or determine the type of any object in R.

Examples

```
class(2)
[1] "numeric"
class(3.145)
[1] "numeric"
```

```
class(TRUE)

[1] "logical"

class("female")

[1] "character"

class("I am from Abuja")

[1] "character"

class(FALSE)

[1] "logical"
```

Variable Assignment

A variable allows you to store a value (e.g., 5) or an object (e.g., a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored with it. You can create a variable with the help of an assignment operator ← or =.

Examples

```
ezekiel ← 4
ezekiel
```

```
[1] 4
```

You can access the type of object in ezekiel:

```
class(ezekiel)
```

```
[1] "numeric"
```

```
state ← "Lagos"
```

```
state
[1] "Lagos"
```

```
class(state)
```

```
[1] "character"
```

Variable Assignment

```
# Age now
```

```
age ← 15
```

```
age
```

```
[1] 15
```

```
class(age)
```

```
[1] "numeric"
```

```
# Next year age =
```

```
age + 1
```

```
[1] 16
```

```
# Age two years ago:
```

```
age - 2
```

```
[1] 13
```

Other examples

```
name ← "Gift"
```

```
age ← 30
```

gender ← "Female"

country ← "Nigeria"

smile_face ← TRUE

Rules for Naming Variables

- All variables must begin with a letter of the alphabet.
- After the initial letter, variable names can also contain (_ or .) and numbers. No spaces or special characters, however, are allowed.
- R is case sensitive. Uppercase characters are different from lowercase characters.

Samples of acceptable variable names	**Samples of unacceptable variable names**
health.status	health (status)
covid_19_cases	covid-19-cases
budget2021	2021 budget
sales_price_2021	sales price 2021

The best naming convention is to choose a variable name that will tell the reader of the program what the variable represents.

Exercise

What is the class of the following data types?

age ← 15

diabetic_status ← "No"

five_less_than_2 ← FALSE

weight ← "60.4 kg"

smile_face ← "FALSE"

Data Type Conversions

R allows us to convert objects from one data type to another. For example, you may need to convert your character variable to a numeric. In programming this is called typecasting and has a lot of very many useful applications, as will be shown later.

Consider the following scenario: You have a variable called "weight" that you assign "64.45" to instead of 64.45. You may need to convert this to numeric/double so that you can use it appropriately in your code.

In order to convert the data, you need to use the as.datatype_name function. As part of this function, you specify the data type you are converting to. The following table shows examples of those functions:

Data type converting to	How to do it
numeric	as.numeric(variable_name)
character	as.character(variable_name)
logical	as.logical(variable_name)
complex	as.complex(variable_name)

Examples

weight ← "64.45"

smiling_face ← "FALSE"

height ← "161.5 cm"

As you can see, some of the variables are not in the right data type.

weight_before ← "64.45"

class(weight_before)
[1] "character"

weight_now = as.numeric(`weight_before`)

weight_now

```
[1] 64.45

class (weight_now)

[1] "numeric"

Examples

smiling_face_old ← "FALSE"

smiling_face_old

[1] "FALSE"

class(smiling_face_old)
[1] "character"

smiling_face_new ← as.logical(`smiling_face_old`)

smiling_face_new
[1] FALSE

class(smiling_face_new)

[1] "logical"
```

The NA Result

If R cannot convert an object, then it will return an NA (Not Available). This is its way of saying that it tried to convert the value, but it couldn't find anything that would logically match with the conversion. The following scenarios will return NA values:

- Converting mixed characters to numeric, e.g., "334cm"
- Converting any character to logical, except TRUE (T) or FALSE (F).

```
height ← "161.5 cm"
as.numeric(`height`)
```

```
[1] NA
```

```
smiling_face ← "No"
```

```
as.logical(`smiling_face`)
```

```
[1] NA
```

```
smiling_face ← "Yes"
```

```
as.logical(`smiling_face`)
```

```
[1] NA
```

Summary of R Programming

In this study session, you have learnt that:

1. R is another programming language and is open source
2. arithmetic operators include +, -, /, *, ^, %%, and ()
3. R makes use of # to make a comment
4. Comparison operators include ==, !=, >, <, >=, and <=.
5. Atomic data types include numeric, character, logical, and factor