CMDB Bootcamp

Table of Contents

# 1 Homepage

This is the course homepage and digital textbook for CMDB Bootcamp.

#### 1.0.0.1 Instructors

Rajiv McCoy, rajiv.mccoy[at]jhu.edu Mike Sauria, mike.sauria[at]jhu.edu Fred Tan, tan[at]ciwemb.edu

#### 1.0.0.2 Schedule & Logistics

Class is **Tuesdays from 3-3:50PM**, in **UTL G89**.

Please bring your laptop with you to every class.

| Session | Content |
| --- | --- |
| **Session 1:** | The reference genome & genome browsers |
| **Session 2:** | *De novo* mutations |
| **Session 3:** | Linkage disequilibrium |
| **Session 4:** | Simulating evolution |
| **Session 5:** | Population structure – part I |
| **Session 6:** | Population structure – part II |
| **Session 7:** | Genome-wide association studies – part I |
| **Session 8:** | Genome-wide association studies – part II |
| **Session 9:** | Scans for selection – part I |
| **Session 10:** | Scans for selection – part II |
| **Session 11:** | Archaic admixture |
| **Session 12:** | Gene expression |
| **Session 13:** | Coronavirus phylogenetics |

# 2 The Unix Shell

## 2.1 Introducing the Shell

## 2.2 Navigating Files and Directories

## 2.3 Working with Files and Directories

# 3 Python

## 3.1 Data Types

**Integers** are whole numbers. For example:

* 2
* -3
* 0

**Floats** are numbers with a decimal point. For example:

* 1.2
* -3.0
* 26/3 (evaluates to 8.666)

**Strings** are characters enclosed by single ' or double " quotation marks. Any text enclosed by quotes will be treated as a string.

* "My Grandpa's deck has no pathetic cards"
* '85.3'

Note the second example – 85.3 is a float; '85.3' is a string.

**Booleans** have two possible values: True and False. These can also be expressed as 1 (true) or 0 (false).

## 3.2 Variables

Variables are assigned using the = sign:

variable\_name = value

For example:

* composer = "buxtehude"
* year = 1637

The variable name can be almost anything. Here are some general rules to consider when naming a variable:

* The name must start with a letter or underscore
* The name can only consist of letters, numbers, or underscores
* Variables are case sensitive (i.e. Python interprets my\_number, MY\_NUMBER, and My\_Number as different variables)
* Python has a set of “reserved words” that cannot be used as variable names. These are words that already have a set meaning in Python, such as True, False, for, and if. A full list can be found [**here**](https://www.programiz.com/python-programming/keywords-identifier).

## 3.3 Math

### 3.3.1 Mathematical operations

A lot of mathematical operations in Python are straightforward. Here are some of the basic operations we can perform:

* + and -: addition and subtraction
* \* and /: multiplication and division
* \*\*: exponentials

We can perform mathematical operations on values directly:

print(2 + 3)

## 5

Or we can operate on variables:

myValue = 4   
print(myValue\*\*2)

## 16

Python will automatically convert integers to floats when appropriate:

print(3 + 2.2)

## 5.2

We can save the output of an expression as a variable:

my\_product = 2 \* 10  
print(my\_product)

## 20

And likewise we can perform mathematical operations on variables, if these variables store numeric data:

number1 = 7  
print(number1 / 2)

## 3.5

### 3.3.2 Order of operations

Python follows the usual mathematical order of operations. And like in math, we can use parentheses () to enforce a specific order.

print(2 \* (2 + 2))

## 8

### 3.3.3 Comparisons

In Python, comparisons will always return a Boolean, i.e. either True or False. We can use the following syntax to compare values:

* < and >: greater than, less than
* <= and >=: greater than or equal to, less than or equal to

For example:

print(5.3 < 17)

## True

print(4. >= (8/2))

## True

* == checks if two values are equal. != checks if two values are different.

print(2 == (10 - 8))

## True

Note that a single equals sign = is used to assign values. However, a double equals sign == is used to compare values.

* var1 = 3 sets the variable var1 to 3
* var1 == 3 checks whether the value of var1 is 3

## 3.4 Built-In Functions and Methods

### 3.4.1 Functions

A **function** is a block of code that performs a task. Python comes with a substantial set of pre-written functions.

***Text on how a function is formulated - name, parentheses, optionally arguments***

***Is it worth talking about named arguments or positional arguments. Be introspective about this***

For example, the print() function displays ***FINISH ME***

***OTHER Built-In FUNCTIONS***’

### 3.4.2 Methods

Every data type that we use in Python (strings, integers, etc.) is associated with a set of functions unique to the data type. These functions are called **methods**.

The syntax for using a method is:

<objectName>.<methodName>()

#### 3.4.2.1 Example

The string method .upper() is used to convert a string into uppercase letters. We can use it either directly on the string itself:

"peppa".upper()

## 'PEPPA'

or by operating on a variable:

pig = "peppa"  
pig.upper()

## 'PEPPA'

#### 3.4.2.2 Modifying objects

Using a method may or not modify the object you run it on. For example, the .upper() method shows you the uppercase version of a string but **does not** modify the actual object that you are operating on. Observe the following code block:

pig = "peppa"  
pig.upper()  
print(pig)

## peppa

pig has not been updated. To actually save the result of .upper(), we have to assign it to a variable:

pig\_caps = pig.upper()  
print(pig\_caps)

## PEPPA

Some methods **do** modify underlying variables. For example, the list .append() method (which we will learn about later) *does* change the list if operates on:

myList = []  
print(myList)

## []

myList.append('apple')  
print(myList)

## ['apple']

myList is altered by myList.append('apple'). There’s no need to type something like myList = myList.append('apple') - this would in fact be an error. Whether methods modify the objects they operate on is something you’ll have to keep track of on a case-by-case basis as you learn new methods.

## 3.5 Type Conversion

## 3.6 String Methods

### 3.6.1 .upper() and .lower()

The .upper() and .lower() methods take a string and convert it to uppercase and lowercase, respectively.

print("out on the wily, windy moors".upper())

## OUT ON THE WILY, WINDY MOORS

aria = "Piangerò La Sorte Mia"  
print(aria.lower())

## piangerò la sorte mia

### 3.6.2 .split()

The .split() method takes a string and splits it into a list, dividing the list on a **delimiter** (i.e., separator). The delimiter is provided as an argument:

print("Newt eye, frog toe".split(','))

## ['Newt eye', ' frog toe']

If no argument is provided, then the string is split on whitespace (that is, it is split whenever a space or tab is encountered).

print("Eye of newt and toe of frog".split())

## ['Eye', 'of', 'newt', 'and', 'toe', 'of', 'frog']

### 3.6.3 .join()

The .join() method is the inverse of .split(): converts a list into a string, with list elements separated by a delimiter. The general syntax is:

"<delimiter>".join(<list>)

For example:

" ".join(["I", "found", "a", "fox", "caught", "by", "dogs"])

## 'I found a fox caught by dogs'

If we do not provide a delimiter, then the strings are directly concatenated:

"".join(["I", "found", "a", "fox", "caught", "by", "dogs"])

## 'Ifoundafoxcaughtbydogs'

### 3.6.4 .rstrip(), .lstrip(), .strip()

These three methods remove unwanted characters on the right, left, or both sides of a string. You can provide the characters you want to remove as an argument:

"ricercar........,,,,,,".rstrip(",.")

## 'ricercar'

Without an argument, the methods remove spaces:

" ricercar ".lstrip()

## 'ricercar '

Note that in the above example we strip the spaces to the left of the main text, but we **do not** remove the spaces from the middle or right end of the text.

## 3.7 Lists

Lists allow us to store multiple objects together.

A list is a sequential group of variables, denoted in Python by square brackets [], with individual entries separated by commas. A few of the neat properties of lists are:

* **Ordered**: The list [1, 5, 3, 7] will always store those numbers in the same order.
* **Mixed data types**: ["mercury", 13, 5.3, False] is a valid list which contains every data type we’ve seen so far.
* **Can contain other lists**: [[2, 3], "sulfur", 12, 18]
* **Can contain repeat values**: ["tomato", "tomato", "tomato", "sulfur"]

### 3.7.1 Indexing

What if we want to extract a specific value from a list? We can use **indexing**. To index in Python, we use the following syntax:

variable\_name[index]

where index is the number of the item we wish to extract.

alchemists = ["Zosimos", "Oresme", "Flamel", "pseudo-Aristotle"]  
print(alchemists[1])

## Oresme

Notice that when we printed the item at position 1, we printed out the *second* entry in alchemists. This is because in Python, **indexing begins at 0**. To print out the first entry, we would use alchemists[0].

A couple interesting things we can do with indexing:

* To print multiple consecutive items, we can provie two numbers separated by a colon :.

print(alchemists[0:2])

## ['Zosimos', 'Oresme']

Note that the first number is **inclusive** and the second number is **exclusive**: we include the item at position 0 ('Zosimos'), but not the item at position 2 ('Flamel').

* We can index in reverse. To index from the end of a list, we use negative numbers.

print(alchemists[-1])

## pseudo-Aristotle

Nested lists

How would we extract the number 3 from the list below?

my\_list = [1, 2, [3, 4], 5]

First, we extract the [3, 4] list. This is the third item of the outer list, so it is at position 2 (remember, indexing in Python starts at 0). So we can access the interior list with my\_list[2]:

print(my\_list[2])

## [3, 4]

Within the interior list, 3 is the first item, so it is at position 0. It can be accessed with the syntax my\_list[2][0]:

print(my\_list[2][0])

## 3

#### 3.7.1.1 Indexing Strings

We can also apply indexing to extract substrings from within a string. This is done identically to how we index a list:

creature = 'stingray'  
print(creature[0:5])

## sting

### 3.7.2 Adding to lists

Finally, we can add entries to the end of a list. We do this with the append() method, which is used with the following syntax:

list\_name.append(item)

For example:

florilegium = ['marigold', 'thistle', 'wormwood']  
florilegium.append('tansy')  
print(florilegium)

## ['marigold', 'thistle', 'wormwood', 'tansy']

## 3.8 For Loops

In Python, we will often want to perform an action more than once. For example, if we have a list, we might want to operate on every item within the list one by one.

One way to do this is to make use of a for loop, which is structured like this:

for <temporary\_variable> in <thing to loop through>:  
 {do something}

For example:

stations = ["Three Note Oddity", "Radio Londres", "Cherry Ripe", "Swedish Rhapsody"]  
  
for i in stations:  
 print(i)

## Three Note Oddity  
## Radio Londres  
## Cherry Ripe  
## Swedish Rhapsody

Here is how the loop works:

In the line for i in stations: we are defining the temporary variable i (the name of this variable is arbitrary). We are also saying that we are looping through stations. So in this first iteration of the for loop, i takes on the value of the first item in stations, that is “Three Note Oddity”. Now we perform all of the indented code, which here is just a single print() statement.

Now we go back and set the value of i to that of the second entry in stations, or “Radio Londres”. We execute all of the indented code, printing out “Radio Londres”.

Now we go back and set the value of i to that of the third value in stations, or “Cherry Ripe”. And we keep on doing this until there is nothing left in my\_list.

In the above example, there was a single indented line in the bodyo of the loop, but the for loop can be arbitrarily long. Here is an example of a for loop structured in the same way, but with more going on in the body of the loop:

my\_list = [1, 4, 6, 9, 10, 2]  
  
for i in my\_list:  
 i = i + 3  
 i = i \*\* 2  
 print(i)

## 16  
## 49  
## 81  
## 144  
## 169  
## 25

We can also use a for loop to perform an action a set number of times, even when we don’t have a list to loop through. To do so, we can use the range() function. We’ll use this function a ton throughout the course. When we run the range() function with a single integer inside the paranthesis, it generates a sequence of numbers from 0 up to and not including the number provided. So to run a function 3 times, we would provide the for loop with range(3) (i.e. [0, 1, 2]).

for i in range(3):  
 print(i)

## 0  
## 1  
## 2

So far, the body of our for loops has always referenced the temporary variable i. We can also use a for loop to run a block of code repeatedly without actually manipulating the temporary variable within that block:

for i in range(6):  
 print("All work and no play")

## All work and no play  
## All work and no play  
## All work and no play  
## All work and no play  
## All work and no play  
## All work and no play

## 3.9 Reading in Data and Text Parsing

***FILL***

## 3.10 Modules

Often, we need to make use of functions beyond those that are packaged with Python. To do this, we can import a **module**, or a collection of pre-written functions. A module is imported with the following syntax: import <module name>. For example, to import the popular plotting module Matplotlib, we write:

import matplotlib.pyplot

To use a function from this module, we need to reference both the package name and the function name, with the general syntax: moduleName.functionName(). To user Matplotlib’s show() function, we would write:

matplotlib.pyplot.show()

What if you don’t want to write out matplotlib.pyplot in full each time that you run a Matplotlib function? To simplify this, we can give our modules a shorthand name. For example:

import matplotlib.pyplot as plt

Now, instead of writing out matplotlib.pyplot, we can just write plt. The previous matplotlib.pyplot.show() command is shortened to:

plt.show()

## 3.11 Plotting

In this course, we will use the library Matplotlib for plotting. For concision, we will import the matplotlib.pyplot module with the name plt, as such:

import matplotlib.pyplot as plt

## 3.12 Errors

# 4 Git

## 4.1 Tracking Changes

## 4.2 Ignoring Things

# Authors

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Instructor | [Rajiv McCoy](https://mccoy-lab.org/) |
| Content Author | [Stephanie Yan](https://stephaniemyan.github.io/) |
| Content Author | [Kate Weaver](https://kweav.github.io/) |
| **Website** |  |
| Template | [Jeff Leek](https://jtleek.com/) & [The Johns Hopkins Data Science Lab](https://jhudatascience.org/index.html) |
| Design Inspiration | [Ali Madooei](https://engineering.jhu.edu/faculty/ali-madooei/) & [JHU Data Structures](https://cs226sp22.github.io/) |
| **Funding** |  |
| JHU Center for Educational Resources | Techology Fellowship Grant |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.5 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2023-08-09   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.5)   
## bookdown 0.24 2023-03-28 [1] Github (rstudio/bookdown@88bc4ea)   
## cachem 1.0.7 2023-02-24 [1] CRAN (R 4.0.2)   
## callr 3.5.0 2020-10-08 [1] RSPM (R 4.0.2)   
## cli 3.6.1 2023-03-23 [1] CRAN (R 4.0.2)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.20 2023-01-17 [1] CRAN (R 4.0.2)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## fastmap 1.1.1 2023-02-24 [1] CRAN (R 4.0.2)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.4.2 2020-08-27 [1] RSPM (R 4.0.5)   
## here 1.0.1 2020-12-13 [1] CRAN (R 4.0.2)   
## hms 0.5.3 2020-01-08 [1] RSPM (R 4.0.0)   
## htmltools 0.5.5 2023-03-23 [1] CRAN (R 4.0.2)   
## jsonlite 1.7.1 2020-09-07 [1] RSPM (R 4.0.2)   
## knitr 1.33 2023-03-28 [1] Github (yihui/knitr@a1052d1)   
## lattice 0.20-41 2020-04-02 [2] CRAN (R 4.0.2)   
## lifecycle 1.0.3 2022-10-07 [1] CRAN (R 4.0.2)   
## magrittr 2.0.3 2022-03-30 [1] CRAN (R 4.0.2)   
## Matrix 1.2-18 2019-11-27 [2] CRAN (R 4.0.2)   
## memoise 2.0.1 2021-11-26 [1] CRAN (R 4.0.2)   
## ottrpal 1.0.1 2023-03-28 [1] Github (jhudsl/ottrpal@151e412)   
## pillar 1.9.0 2023-03-22 [1] CRAN (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgconfig 2.0.3 2019-09-22 [1] RSPM (R 4.0.3)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## png 0.1-8 2022-11-29 [1] CRAN (R 4.0.2)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.4.0 2020-10-07 [1] RSPM (R 4.0.2)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## rappdirs 0.3.3 2021-01-31 [1] CRAN (R 4.0.2)   
## Rcpp 1.0.10 2023-01-22 [1] CRAN (R 4.0.2)   
## readr 1.4.0 2020-10-05 [1] RSPM (R 4.0.2)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## reticulate 1.28 2023-01-27 [1] CRAN (R 4.0.2)   
## rlang 1.1.0 2023-03-14 [1] CRAN (R 4.0.2)   
## rmarkdown 2.10 2023-03-28 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.0.2)   
## rstudioapi 0.11 2020-02-07 [1] RSPM (R 4.0.0)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2023-03-28 [1] Github (R-lib/testthat@e99155a)   
## tibble 3.2.1 2023-03-20 [1] CRAN (R 4.0.2)   
## usethis 1.6.3 2020-09-17 [1] RSPM (R 4.0.2)   
## utf8 1.1.4 2018-05-24 [1] RSPM (R 4.0.3)   
## vctrs 0.6.1 2023-03-22 [1] CRAN (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2023-03-28 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library