

MSAS Tutorial Sequence -Module Three-

WRITTEN BY ADVAY MUCHOOR

**LET'S
GET
READY
TO
WORK**



Goals of Today

1. Finish up the basics of Python
 1. Booleans
 2. If/Else
 3. Lists
 4. Loops
 5. Dictionaries
2. Understand Pandas
3. Getting some I/O with dataframes
 1. Reading from Excel
 2. Webscraping
 3. Basic dataframe operations



More Python

Booleans

- Booleans represent a “True” or “False” variable (can only hold these two values)
- Often used for comparison purposes
- Declared just like any other variable
 - `is_starter = True`
 - `temp = False`
- Can also be used in inequalities and expressions
 - More on this in a bit

Operators

- The only three operators you should need to know for python are the following:
 - “and”
 - “or”
 - “not”
 - All three operators are used in their plaintext form (no need for “&&” or “| |” or “!”)

- Truth tables:

A	Operand	B	Result
True	and	True	True
True	and	False	False
False	and	False	False
True	or	True	True
True	or	False	True
False	or	False	False

Conditions

- Recap: Python supports the usual logical conditions from mathematics:
 - Equals: `a == b`
 - Not Equals: `a != b`
 - Less than: `a < b`
 - Less than or equal to: `a <= b`
 - Greater than: `a > b`
 - Greater than or equal to: `a >= b`
- “If” statements allow us to check “if” a condition is true or false
 - If the statement evaluates to True, we will go into to that “if” block and run what is inside it

```
In [1]: if 2 < 3:  
        print("hello")
```

hello

```
In [2]: brady_touchdowns = 25  
        rivers_touchdowns = 20  
  
        if brady_touchdowns > rivers_touchdowns:  
            print("Brady is better")
```

Brady is better

```
In [4]: home = True  
        sun = True  
  
        if home and sun:  
            print("We should win!")
```

We should win!

```
In [5]: home = False  
        sun = True  
  
        if home or sun:  
            print("We have a chance")
```

We have a chance

```
In [6]: road = False  
  
        if not road:  
            print("We are playing at home")
```

We are playing at home

Else statements

- Must be directly paired with an “if” statement
- The “else” keyword catches anything that isn’t caught by the if statement
- If a condition is not hit, the else statement will run. Otherwise it will not.
 - Yes, an if/else clause is literally an if/else clause

```
In [1]: lebron = 27.4  
        harden = 36.1
```

Harden averaged more ppg than lebron

```
In [2]: # Don't do this
```

```
if lebron > harden:  
    print("Lebron averaged more ppg than harden")  
if harden > lebron:  
    print("Harden averaged more ppg than lebron")
```

Harden averaged more ppg than lebron

```
In [3]: # Do this
```

```
if lebron > harden:  
    print("Lebron averaged more ppg than harden")  
else:  
    print("Harden averaged more ppg than lebron")
```

Harden averaged more ppg than lebron

```
In [4]: goals_scored = 3

if goals_scored == 0:
    print("shutout")
elif goals_scored == 1:
    print("scored")
elif goals_scored == 3:
    print("Hat Trick!")
else:
    print("multiple scored")
```

Hat Trick!

```
In [5]: goals_scored = 2

if goals_scored == 0:
    print("shutout")
elif goals_scored == 1:
    print("scored")
elif goals_scored == 3:
    print("Hat Trick!")
else:
    print("multiple scored")
```

multiple scored

If/Elif/Else

- “Elif” clauses allow for multiple if statements to be checked (short for “else if”)
- First, the “if” statement is checked
- Next, each “elif” clause will be checked until one is hit
- Finally, if none of the “elif” clauses are hit, the “else” clause is run

INDENTATION MATTERS

```
In [1]: a = 10  
        b = 5  
  
        if a > b:  
            print("here")
```

```
File "<ipython-input-1-c0905ae9b858>", line 5  
    print("here")  
    ^
```

IndentationError: expected an indented block

Lists

- Lists are a collection of items that is ordered and can be changed
- Lists are indicated by square brackets: []
- We can declare a list in three different ways:
 - An empty list `my_list = []`
 - Fill it with elements `my_list_2 = ["Ravens", "Saints", "Patriots"]`
 - This is a list of strings, but we can have a list of any datatype
 - You can mix and match your datatypes within the list, but this is considered bad practice
 - Set it equal to another list `my_list_3 = my_list_2`
- You can get the number of elements in the list by calling the “len” function
 - `my_list = [1, 2, 3, 4]`
 - `print(len(my_list))` 4
- Clear the list by calling the “clear function”
 - `my_list.clear()`
 - `print(my_list)` ???

Accessing elements of a list

- You can access list items by referring to the index number
- Lists in Python are “0-index based”
- Example:
 - `stats = [“pts”, “reb”, “ast”]`
 - `print(stats[0])` “pts”
 - `print(stats[1])` “reb”
 - `Print(stats[2])` “ast”
 - `Print(stats[3])` Error! Why?
 - `Print(stats[-1])` ???
- You can access a range of indexes by specifying where to start and where to end (called slicing)
 - Use the following format: `list[start:end]`
 - `qbs = [“Brady”, “Rivers”, “Jackson”, “Stafford”, “Cousins”]`
 - `slice = qbs[2:4]`
 - `print(slice)` [“Jackson”, “Stafford”, “Cousins”]

Updating a list element

- To update a list element, simply index into the list and reassign its value

```
In [9]: points = [10, 7, 13, 21, 15]
print(points)
print(points[3])
points[3] = 19
print(points)
print(points[3])
```

```
[10, 7, 13, 21, 15]
```

```
21
```

```
[10, 7, 13, 19, 15]
```

```
19
```

Adding to a list

```
In [1]: ppg = [27.4, 31.5, 21.5, 18.9, 29.7]
```

```
In [2]: # append to the end of a list  
ppg.append(15.7)  
ppg
```

```
Out[2]: [27.4, 31.5, 21.5, 18.9, 29.7, 15.7]
```

```
In [3]: # add another list  
more_ppg = [11.5, 20.1, 16.7]  
ppg += more_ppg  
ppg
```

```
Out[3]: [27.4, 31.5, 21.5, 18.9, 29.7, 15.7, 11.5, 20.1, 16.7]
```

```
In [4]: # insert at a specific position  
ppg.insert(3, 10.1)  
ppg
```

```
Out[4]: [27.4, 31.5, 21.5, 10.1, 18.9, 29.7, 15.7, 11.5, 20.1, 16.7]
```

Range-based for loops (one way to loop)

- A “for” loop is used for iterating over a sequence
 - For our purposes, this will usually be a list
- We will only cover range based loops in this module
 - This is a brief sneak peak... for more information
 - Other loops aren't used traditionally in data science unless you are exploring a niche
 - Don't need to index into array
- Use cases:
 - When you want to view every element of a list in sequential order
 - Save time from indexing into specific elements

• Syntax:

```
for element in list:  
    do something
```


Basic loop examples

```
In [6]: simple_list = [0, 3, 2, 5]
        for number in simple_list:
            print(number)
```

```
0
3
2
5
```

```
In [5]: AFC_West = ["Broncos", "Chargers", "Chiefs", "Raiders"]

        for team in AFC_West:
            print(team)
```

```
Broncos
Chargers
Chiefs
Raiders
```

Basic loop applications

```
In [7]: rebounds = [1, 4, 3, 2, 6, 1, 3]
sum = 0

for game in rebounds:
    sum = sum + game

print(sum / len(rebounds))
# What does this value represent?
```

2.857142857142857

```
In [18]: rebounds = [1, 4, 3, 2, 6, 1, 3]
good_games = []
bad_games = []

for game in rebounds:
    if game <= 3:
        bad_games.append(game)
    else:
        good_games.append(game)

print(bad_games)
print(good_games)
```

[1, 3, 2, 1, 3]
[4, 6]

Dictionaries

- Arguably one of the most important data structures in all of computer science
- A dictionary is a collection of items which is unordered and indexed.
- In Python, dictionaries are written with curly brackets and have keys and values
 - Keys are how you access values within the data structure
 - In a list, you index into the elements by numerical index
 - In a dictionary, you index into values with a key!
 - Every key is unique, but keys can have the same value
- Why are dictionaries useful?

Example Dictionaries

```
players = {  
    "LeBron James": "PF",  
    "Anthony Davis": "C",  
    "Steph Curry" : "PG",  
    "Luka Doncic"  : "SF",  
    "Trae Young"   : "PG",  
    "Pau Gasol"    : "C"  
}
```

```
team_names = {  
    "DET" : "Lions",  
    "GB"  : "Packers",  
    "CHI" : "Bears",  
    "MIN" : "Vikings"  
}
```

Accessing elements in a dictionary

```
In [1]: players = {  
    "LeBron James": "PF",  
    "Anthony Davis": "C",  
    "Steph Curry" : "PG",  
    "Luka Doncic"  : "SF",  
    "Trae Young"   : "PG",  
    "Pau Gasol"    : "C"  
}
```

```
In [2]: players["LeBron James"]
```

```
Out[2]: 'PF'
```

```
In [3]: players["LeBron James"] = "C"  
players
```

```
Out[3]: {'LeBron James': 'C',  
        'Anthony Davis': 'C',  
        'Steph Curry': 'PG',  
        'Luka Doncic': 'SF',  
        'Trae Young': 'PG',  
        'Pau Gasol': 'C'}
```

```
In [4]: players["Kevin Love"]
```

```
-----  
KeyError                                Traceback (most recent call last)  
<ipython-input-4-b16d7fa527bf> in <module>  
----> 1 players["Kevin Love"]  
  
KeyError: 'Kevin Love'
```

```
In [14]: mlb_teams = {  
    "NYY" : "Yankees",  
    "BAL" : "Orioles",  
    "BOS" : "Red Sox",  
    "TB"  : "Rays"  
}
```

```
In [15]: if "TB" in mlb_teams:  
    print("The value of TB is: " + mlb_teams["TB"])
```

The value of TB is: Rays

```
In [16]: if "TOR" not in mlb_teams:  
    print("Forgot that one")  
    mlb_teams["TOR"] = "Blue Jays"  
    print(mlb_teams)
```

Forgot that one

```
{'NYY': 'Yankees', 'BAL': 'Orioles', 'BOS': 'Red Sox', 'TB': 'Rays', 'TOR': 'Blue Jays'}
```

Checking if an element is in a dictionary (try it on lists too!)

Questions about anything we talked about?

IF NOT, TAKE A COUPLE MINUTES TO WORK WITH THESE CONCEPTS

A solid yellow horizontal bar at the bottom of the slide.



Pandas

Welcome to REAL Data Science

Libraries in Python

- A python library is a collection of functions that allows you to perform specific actions with your code
- More often than not, a library is designed for a specific feature. There are libraries to...
 - Move and modify files on your computer
 - Create graphical interfaces for your code
 - Webscrape (more on that in the next module)
 - Perform data science operations 😊
 - And much more!
- Libraries can be imported with the following syntax:
 - `import library` (imports entire library)
 - `import library as lib` (imports entire library as an alias, usually used to not clutter up code)
 - `from library, import function a, b` (imports functions a and b from the library, saves memory)
- To access functions of your imported library, you will need to use the dot operator
 - We will see this throughout the module sequence

What is Pandas?

- From their documentation:

“pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.”

- Very powerful data science library
- The basis of the rest of this module, as well as the next three!
- Import pandas at the top of your jupyter notebook
 - `import pandas as pd`

Series vs DataFrames

- The primary two components of pandas are Series and DataFrames
- Series is a column in your table, and a DataFrame is a multidimensional table comprised of a collection of Series

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3	+	0	0	=	0	3	0
1	2		1	3		1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

Reading in Files

```
import urllib.request
urllib.request.urlretrieve('https://github.com/gfolbe318/MSAS_tutorials/raw/master/3_
Module/nfl_stats.xlsx', 'nfl_stats.xlsx')
```

```
In [1]: import pandas as pd
```

```
In [5]: # Change the filename if you aren't in the same directory

file_name = "nfl_stats.xlsx"

# Read in the file from excel
df = pd.read_excel(file_name)
```

Printing a DataFrame

```
In [6]: # This makes things ugly
print(df)
```

	Rk	Player	Tm	Age	Pos	G	GS	Cmp	Att	Cmp%	Yds	\
0	1	Ben Roethlisberger	PIT	36	QB	16	16	452	675	67.0	5129	
1	2	Andrew Luck*	IND	29	QB	16	16	430	639	67.3	4593	
2	3	Matt Ryan	ATL	33	QB	16	16	422	608	69.4	4924	
3	4	Kirk Cousins	MIN	30	QB	16	16	425	606	70.1	4298	
4	5	Aaron Rodgers*	GNB	35	QB	16	16	372	597	62.3	4442	
5	6	Case Keenum	DEN	30	QB	16	16	365	586	62.3	3890	
6	7	Patrick Mahomes*+	KAN	23	QB	16	16	383	580	66.0	5097	
7	8	Eli Manning	NYG	37	QB	16	16	380	576	66.0	4299	
8	9	Tom Brady*	NWE	41	QB	16	16	375	570	65.8	4355	
9	10	Jared Goff*	LAR	24	QB	16	16	364	561	64.9	4688	
10	11	Matthew Stafford	DET	30	QB	16	16	367	555	66.1	3777	
11	12	Derek Carr	OAK	27	QB	16	16	381	553	68.9	4049	
12	13	Dak Prescott*	DAL	25	QB	16	16	356	526	67.7	3885	
13	14	Philip Rivers*	LAC	37	QB	16	16	347	508	68.3	4308	
14	15	Deshaun Watson*	HOU	23	QB	16	16	345	505	68.3	4165	
15	16	Drew Brees*	NOR	39	QB	15	15	364	489	74.4	3992	
16	17	Baker Mayfield	CLE	23	QB	14	13	310	486	63.8	3725	
17	18	Cam Newton	CAR	29	QB	14	14	320	471	67.9	3395	
18	19	Mitchell Trubisky*	CHI	24	QB	14	14	289	434	66.6	3223	

Displaying a DataFrame

```
In [7]: # This makes things pretty  
df
```

Out[7]:

	Rk	Player	Tm	Age	Pos	G	GS	Cmp	Att	Cmp%	Yds	TD	Int	Y/A	Y/C	Y/G	Rate	QBR	Sk
0	1	Ben Roethlisberger	PIT	36	QB	16	16	452	675	67.0	5129	34	16	7.6	11.3	320.6	96.5	71.0	24
1	2	Andrew Luck*	IND	29	QB	16	16	430	639	67.3	4593	39	15	7.2	10.7	287.1	98.7	69.4	18
2	3	Matt Ryan	ATL	33	QB	16	16	422	608	69.4	4924	35	7	8.1	11.7	307.8	108.1	68.5	42
3	4	Kirk Cousins	MIN	30	QB	16	16	425	606	70.1	4298	30	10	7.1	10.1	268.6	99.7	58.2	40
4	5	Aaron Rodgers*	GNB	35	QB	16	16	372	597	62.3	4442	25	2	7.4	11.9	277.6	97.6	54.4	49
5	6	Case Keenum	DEN	30	QB	16	16	365	586	62.3	3890	18	15	6.6	10.7	243.1	81.2	45.5	34
6	7	Patrick Mahomes*+	KAN	23	QB	16	16	383	580	66.0	5097	50	12	8.8	13.3	318.6	113.8	80.4	26
7	8	Eli Manning	NYG	37	QB	16	16	380	576	66.0	4299	21	11	7.5	11.3	268.7	92.4	48.7	47
8	9	Tom Brady*	NWE	41	QB	16	16	375	570	65.8	4355	29	11	7.6	11.6	272.2	97.7	66.6	21
9	10	Jared Goff*	LAR	24	QB	16	16	364	561	64.9	4688	32	12	8.4	12.9	293.0	101.1	63.1	33
10	11	Matthew Stafford	DET	30	QB	16	16	367	555	66.1	3777	21	11	6.8	10.3	236.1	89.9	48.4	40
11	12	Derek Carr	OAK	27	QB	16	16	381	553	68.9	4049	19	10	7.3	10.6	253.1	93.9	46.9	51

Displaying the head of a DataFrame

In [8]: `# Print out the head(start) of your DataFrame`
`df.head()`

Out[8]:

	Rk	Player	Tm	Age	Pos	G	GS	Cmp	Att	Cmp%	Yds	TD	Int	Y/A	Y/C	Y/G	Rate	QBR	Sk
0	1	Ben Roethlisberger	PIT	36	QB	16	16	452	675	67.0	5129	34	16	7.6	11.3	320.6	96.5	71.0	24
1	2	Andrew Luck*	IND	29	QB	16	16	430	639	67.3	4593	39	15	7.2	10.7	287.1	98.7	69.4	18
2	3	Matt Ryan	ATL	33	QB	16	16	422	608	69.4	4924	35	7	8.1	11.7	307.8	108.1	68.5	42
3	4	Kirk Cousins	MIN	30	QB	16	16	425	606	70.1	4298	30	10	7.1	10.1	268.6	99.7	58.2	40
4	5	Aaron Rodgers*	GNB	35	QB	16	16	372	597	62.3	4442	25	2	7.4	11.9	277.6	97.6	54.4	49

Display a Specific Number of Rows

```
In [10]: # Print out n number of rows
num_rows = 10
df.head(num_rows)
```

Out[10]:

	Rk	Player	Tm	Age	Pos	G	GS	Cmp	Att	Cmp%	Yds	TD	Int	Y/A	Y/C	Y/G	Rate	QBR	Sk
0	1	Ben Roethlisberger	PIT	36	QB	16	16	452	675	67.0	5129	34	16	7.6	11.3	320.6	96.5	71.0	24
1	2	Andrew Luck*	IND	29	QB	16	16	430	639	67.3	4593	39	15	7.2	10.7	287.1	98.7	69.4	18
2	3	Matt Ryan	ATL	33	QB	16	16	422	608	69.4	4924	35	7	8.1	11.7	307.8	108.1	68.5	42
3	4	Kirk Cousins	MIN	30	QB	16	16	425	606	70.1	4298	30	10	7.1	10.1	268.6	99.7	58.2	40
4	5	Aaron Rodgers*	GNB	35	QB	16	16	372	597	62.3	4442	25	2	7.4	11.9	277.6	97.6	54.4	49
5	6	Case Keenum	DEN	30	QB	16	16	365	586	62.3	3890	18	15	6.6	10.7	243.1	81.2	45.5	34
6	7	Patrick Mahomes*+	KAN	23	QB	16	16	383	580	66.0	5097	50	12	8.8	13.3	318.6	113.8	80.4	26
7	8	Eli Manning	NYG	37	QB	16	16	380	576	66.0	4299	21	11	7.5	11.3	268.7	92.4	48.7	47
8	9	Tom Brady*	NWE	41	QB	16	16	375	570	65.8	4355	29	11	7.6	11.6	272.2	97.7	66.6	21
9	10	Jared Goff*	LAR	24	QB	16	16	364	561	64.9	4688	32	12	8.4	12.9	293.0	101.1	63.1	33


```
In [11]: # Get dimensions of dataframe in the format of (rows, columns)
df.shape
```

```
Out[11]: (106, 19)
```

```
In [12]: # Get the names of of the columns
df.columns
```

```
Out[12]: Index(['Rk', 'Player', 'Tm', 'Age', 'Pos', 'G', 'GS', 'Cmp', 'Att', 'Cmp%',
               'Yds', 'TD', 'Int', 'Y/A', 'Y/C', 'Y/G', 'Rate', 'QBR', 'Sk'],
              dtype='object')
```

```
In [13]: # Get the tyeps of each column
df.dtypes
```

```
Out[13]: Rk          int64
Player    object
Tm        object
Age       int64
Pos       object
G         int64
GS        int64
Cmp       int64
Att       int64
Cmp%     float64
Yds       int64
TD        int64
Int       int64
Y/A      float64
Y/C      float64
Y/G      float64
Rate     float64
QBR      float64
Sk       int64
dtype: object
```

Get Information about the DataFrame

Extract Certain Columns

```
In [15]: # Pick out specific columns that you want
new_df = df[["Player"]]
new_df.head()
```

Out[15]:

	Player
0	Ben Roethlisberger
1	Andrew Luck*
2	Matt Ryan
3	Kirk Cousins
4	Aaron Rodgers*

```
In [17]: # Get more than one column
new_df = df[["Player", "Tm", "TD", "Yds"]]
new_df.head()
```

Out[17]:

	Player	Tm	TD	Yds
0	Ben Roethlisberger	PIT	34	5129
1	Andrew Luck*	IND	39	4593
2	Matt Ryan	ATL	35	4924
3	Kirk Cousins	MIN	30	4298
4	Aaron Rodgers*	GNB	25	4442

Write to Excel

```
In [36]: # Write back to an excel sheet  
# Be careful, you can only write to a sheet once  
new_df.to_excel("important.xlsx")
```

Any Questions?
