Learning Objectives

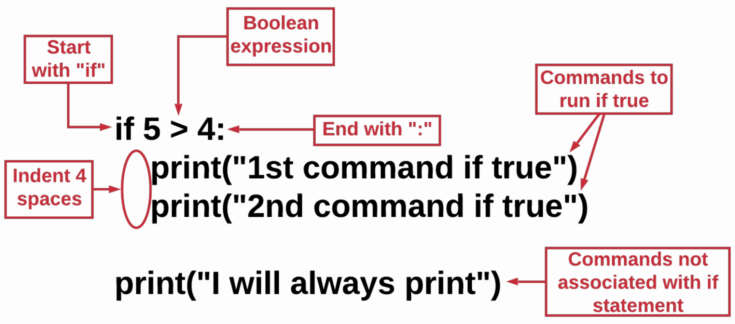
In the third module of this course, you will learn how to handle different scenarios in order to better manipulate your data.

Learning Objectives:

1. Learn about conditionals
2. Applying ‘if else’ statements
3. Using compound conditionals to better filter through our Data Frame

Conditionals

If Statement Syntax


Start with "if"
Boolean expression
Commands to run if true
if 5 > 4:
End with ":"
Indent 4 spaces
print("1st command if true")
print("2nd command if true")
print("I will always print")
Commands not associated with if statement

images/if-statement-syntax

Conditionals are pieces of code that make a decision about what the program is going to do next. The most common conditional is the if statement.

If statements in Python must contain the following items:
* the keyword `if`
* a boolean expression
* a colon
* 4 spaces of indentation for all lines of code that will run if the boolean expression is true.

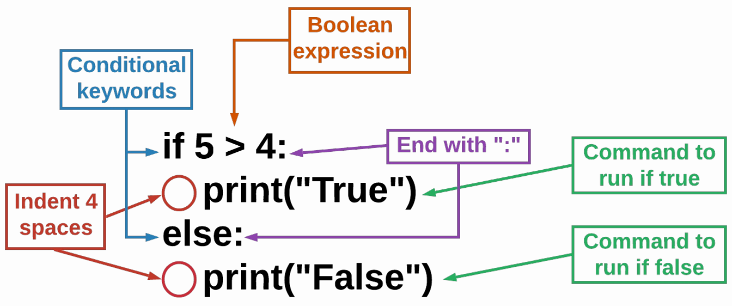
`p ython if 5 > 4:     print("1st command if true")     print("2nd command if true")`

If statements test to see if a certain condition is true. If yes, then specific commands are run. The simple if statement does not do anything if the boolean expression is false.

```
if 7 != 10:
    print("The above statement is true")
print("This is not related to the if statement")
```

If-Else Syntax

The if-else statement checks to see if a condition is true, and then has specific actions that take place. But it also provides a specific set of actions if the boolean expression is false. Use the `else` keyword to introduce the code to run when false. Notice, `else` is aligned with the `if` keyword (no indentation) and has a `:`. You do not write another boolean expression with `else`.



images/if-else-statement-syntax

```
if 5 > 4:
    print("The boolean expression is true")
else:
    print("The boolean expression is false")
```

Code Visualizer

The if-else statement is used when you want something to specific to happen if the boolean expression is true and if you want something else to happen if it is false.

```
my_bool = True

if my_bool:
    print("The value of my_bool is true")
else:
    print("The value of my_bool is false")
```

Compound Conditional

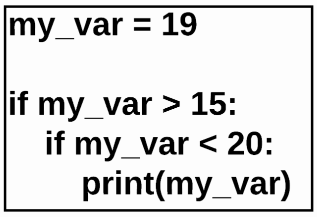
Compound Conditional Syntax

A compound conditional is a conditional (an if statement) that has more than one boolean expression. You need to use the `and` or the `or` keywords to link these boolean expressions together. You can use the `not` keyword, but only in combination with `and` or `or`.

```python
if True and True:
    print("True")
```

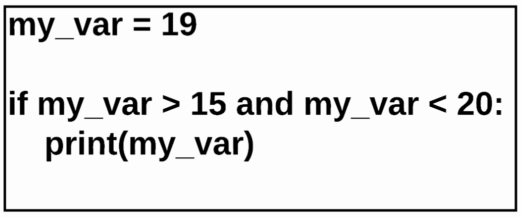
In data science you can use a compound statement to help narrow down search fields. For example, if you want to test that a number is even and greater than 15, you will need two conditionals.

Both code snippets below do the same thing — ask if `my_var` is greater than 15 and if `my_var` is less than 20. If both of these are true, then Python will print the value of `my_var`.



```
my_var = 19

if my_var > 15:
    if my_var < 20:
        print(my_var)
```



```
my_var = 19

if my_var > 15 and my_var < 20:
    print(my_var)
```

For example, let’s open up the homerun file as data2.

1. First we’ll check what our data looks like using `.info()`
2. Show the first couple rows
3. Print out the values where `homerun` greater than 650
4. Print out the values that for `homerun` greater than 650 and less than 700 ***Hint: Use () to group your boolean vector to remove ambiguity.**

```python
#Load our Libraries
import pandas as pd
import matplotlib

data2= pd.read_csv('/home/codio/workspace/csv/home_runs.csv')

# review our data

#print(data2.info())

# print out the head of our data2

#print(data2.head(3))

# print all homeruns betweeen 650 and 700
print(data2[(data2['Home Runs']>=650) & (data2['Home Runs']
        <=700)])
```

Frequency Count

It will often come up that you will need to get the count based on column values.

We have added a new csv file called `states.csv`. Import the states csv then use `info` to get some information about our new data.

```
data= pd.read_csv('/home/codio/workspace/csv/states.csv')
print(data.info())
```

From the information, we have gotten from using `.info` we are not going to check all the possible values for `location` column and their count using the `value_counts` function.

```
data= pd.read_csv('/home/codio/workspace/csv/states.csv')
#print(data.info())
print(data.value_counts(["location"]))
```

The `value_counts` is especially efficient because it enables user to get the count of multiple columns at the same time. Remember our csv had multiple columns so we can add an extra argument.

```
print(data.value_counts(["location","gender"]))
```