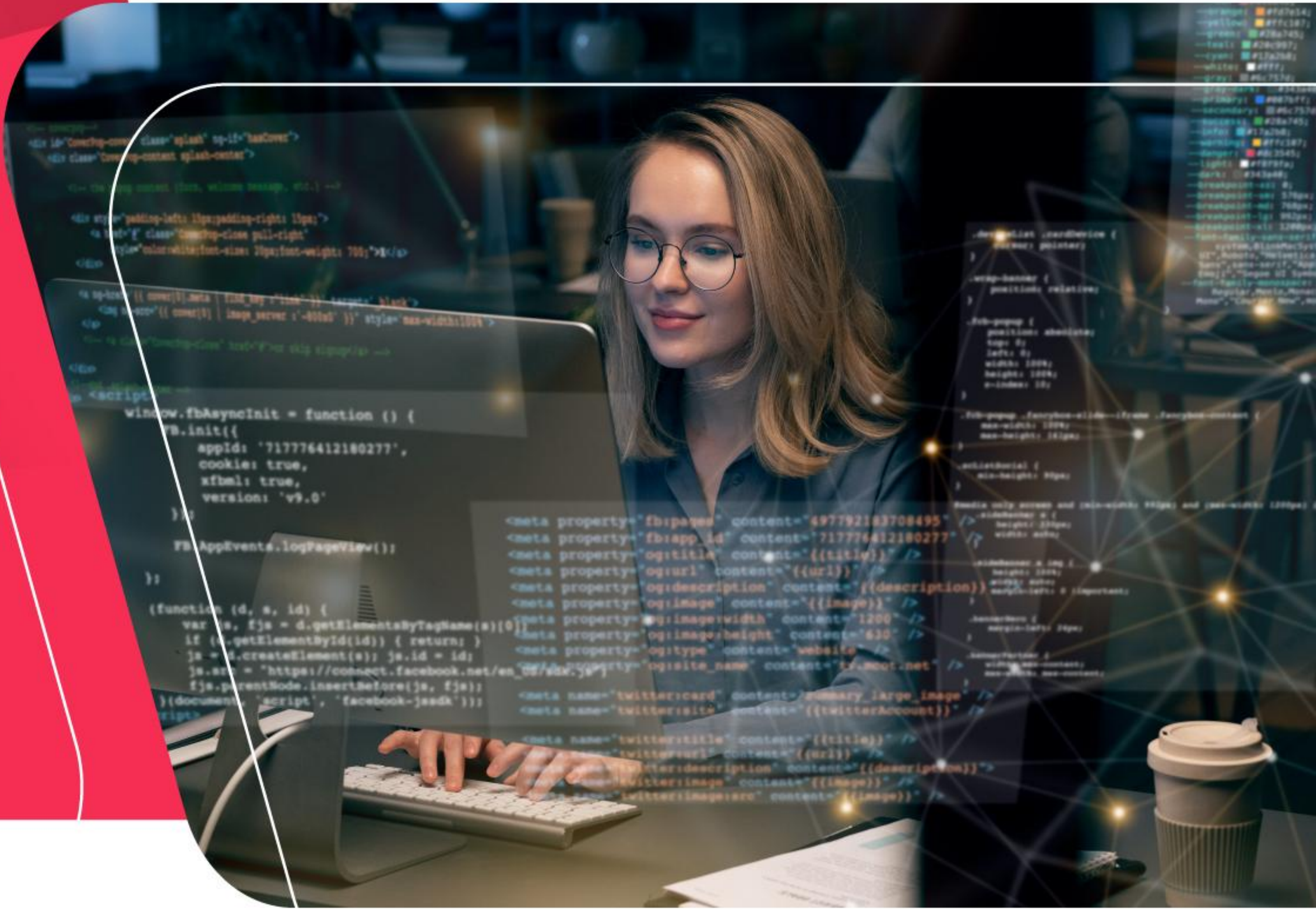
TIA-Academy

Python for Data Science

Day 03



Don’t Repeat Yourself (DRY)

Don’t Repeat Yourself - DRY

DRY (Don’t Repeat Yourself) is a principle of software development. The focus of DRY is to avoid repetition of information.

Why?

One main reason is that when you write code that performs the same tasks over and over again, any modification of one task requires the same change to be made to every single instance of that task! Editing every instance of a task is a lot of work.

By implementing DRY code approaches, you can make your code:

- easier to follow and read (for yourself as well as others), thereby supporting reproducibility
- easier to update because you only have to update your code once, rather than everywhere that code block is used

DRY – Tips and Trick

- **Write Functions To Document and Simplify Repeated Tasks**
- **Write Loops in Python To Simplify Iterative Repetitive Tasks**
- **Conditional Statements**

Conditional If - Advance

Conditional If - Advance

- A conditional statement is used to determine whether a certain condition exists before code is executed.
- Conditional statements can help improve the efficiency of your code by providing you with the ability to control the flow of your code, such as when or how code is executed.

Conditional If - Advance

If the condition provided with the if statement is satisfied (i.e. results in a value of True), then a certain code will execute. If that condition is not met (i.e. results in a value of False), then the code provided with the else statement will execute.

For example:

```
if condition:
    print("Condition is true, and this statement is printed.")
else:
    print("Condition is false (i.e. not true), so a different statement is printed.")
```

Conditional If - Advance

You can use membership operators (e.g. in or not in) to write conditional statements to check whether certain values are contained within a data structure, such as a list, or even a text string.

```
# Create list of average monthly precip (inches) in Boulder, CO
avg_monthly_precip = [0.70,  0.75, 1.85, 2.93, 3.05, 2.02,
                      1.93, 1.62, 1.84, 1.31, 1.39, 0.84]

# Check for value 0.70 in list
if 0.70 in avg_monthly_precip:
    print("Value is in list.")
else:
    print("Value is not in list.")
## Value is in list.
```

Conditional If - Advance

The condition above could also be checked in the opposite manner using not in to check that the value is not in the list:

```
# Check that value 0.71 not in list
if 0.71 not in avg_monthly_precip:
    print("Value is not in list.")
else:
    print("Value is in list.")
## Value is not in list.
```

Conditional If - Advance

You can also use membership operators to check for specific words within a text string.

```
# Check for string "precip" within text string "avg_monthly_temp"
if "precip" in "avg_monthly_temp":
    print("This textstring contains the word precip.")
else:
    print("This textstring does not contain the word precip.")
## This textstring does not contain the word precip.
```

Conditional If - Advance

You can also use identity operators (e.g. is or is not) to write conditional statements to check whether an object is of a certain type (e.g. int, str, list).

```
# Set x to 0
x = 0

# Check if x is type integer
if type(x) is int:
    print(x, "is an integer.")
else:
    print(x, "is not an integer.")
## 0 is an integer.
```

Conditional If - Advance

Logical operators (e.g. and, or, not) allow you to create conditional statements that can check for combinations of conditions. You can use:

- and to execute code if all specified conditions have been met
- or to execute code if at least one specified condition has been met
- not to execute code only if the specified condition has not been met (note that you can use not in combination with and or or to check whether multiple conditions are not met)

Conditional If - Advance

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

p	$\neg p$
F	T
T	F

Looping - Advance

Looping - Advance

- A loop is a sequence of operations that are performed over and over in some specified order. Following the DRY (Don’t Repeat Yourself) principle of clean coding, loops can help you to eliminate repetition in code by replacing duplicate lines of code with an iteration, meaning that you can iteratively execute the same code line or block until it reaches a specified end point.
- Instead of copying and pasting the same code over and over to run it multiple times (say, on multiple variables or files), you can create a loop that contains one instance of the code and that executes that code block on a range or list of values that you provide.

Challenges

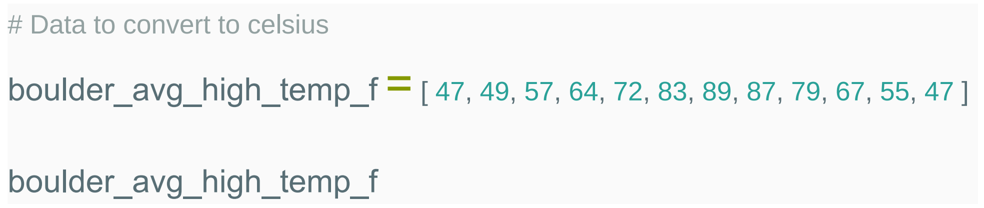
Challenge 1: Print Numbers in a list

The list below contains temperature values for a location in Boulder, Colorado. Create a for loop that loops through each value in the list and prints the value like this: `

temp: 47

HINT: you can print a string and a variable together using the syntax:

print("temp:", variable_name_here)



```
# Data to convert to celsius
boulder_avg_high_temp_f = [ 47, 49, 57, 64, 72, 83, 89, 87, 79, 67, 55, 47 ]

boulder_avg_high_temp_f
```

Challenge 2: Modify Numeric Values in a List

Below is a list of values that represents the average monthly high temperature in Boulder, CO., collected by NOAA. They are currently in Fahrenheit, but can be converted to Celsius by subtracting 32, and multiplying by 5/9.

celcius = (fahrenheit - 32) * 5/9

Create a **new list** with these same temperatures converted to Celsius using a for loop. Call your new list: boulder_avg_high_temp_c

HINT: to complete this challenge you may want to create a new empty list first. Then you can use list_name.append() in each loop iteration to add a new value to your list.

```
# Add your code here
[8.333333333333334, 9.444444444444445, 13.88888888888889, 17.77777777777778, 22.22222222222222,
28.333333333333336, 31.666666666666668, 30.555555555555557, 26.11111111111111, 19.444444444444446,
12.777777777777779, 8.333333333333334]
```