Data Wrangling with Python - Day 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Welcome!

Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

Best practices for virtual classes

1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Stay on mute unless you are speaking.
3. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
4. Participate in activities and ask questions. This will be interactive!
5. Give your honest feedback so we can troubleshoot problems and improve the course.

Welcome: Icebreaker!

In the chat box, answer one of the following questions:

1. Why do you think it's important to learn about data wrangling?

2. What would you like to learn in this class?

Best practices for virtual classes

1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Stay on mute unless you are speaking.
3. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
4. Participate in activities and ask questions. This will remain interactive!
5. Give your honest feedback so we can troubleshoot problems and improve the course.

Prerequisites

You should be familiar with: - Anaconda and Jupyter Notebook - Defining, swapping, printing and deleting variables - Data types including: numbers, logicals, and strings - Basic data structures including: lists, sets, dictionaries, and tuples - Control flow structures and modular code including: - Conditional statements: if-else - For/While loops, break/continue statements, and list comprehensions

Course structure

- **Lecture slides:**
  - 2 hours

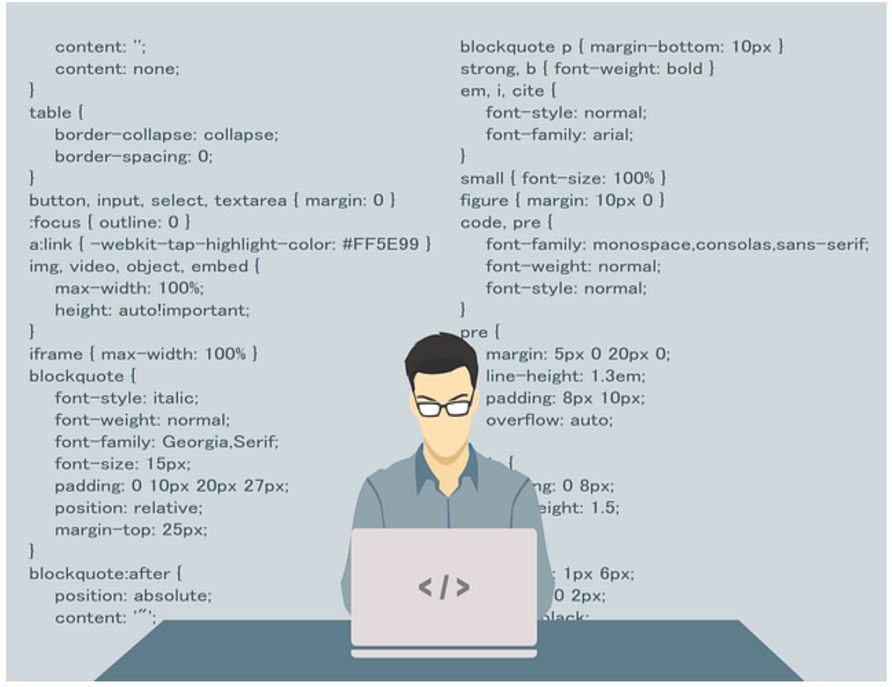
- **Knowledge checks and exercises:**
  - 30 minutes

Module completion checklist

Objective	Complete
Discuss how programming is used across industries and define core functions of data scientists	
Explain the data science life cycle and summarize data science use cases for Python	
Refresh on coding in Python and introduce functions	

Why are we learning to program?

- I'm a *[...]* major, why do I need to learn programming?



- Programming is not just for CompSci people, programming is **becoming a more universal skill** like typing in Word, or making eye-catching presentations in PowerPoint was in the 90's or early 2000's
- Programming facilitates **performing the same operations on a large scale multiple times**
- Programming is a **necessary component of reproducible research**
- Programming makes you **think through your problem from a very specific viewpoint that requires clear formulation of your end goal and methods you would like to utilize**
- The list goes on ...

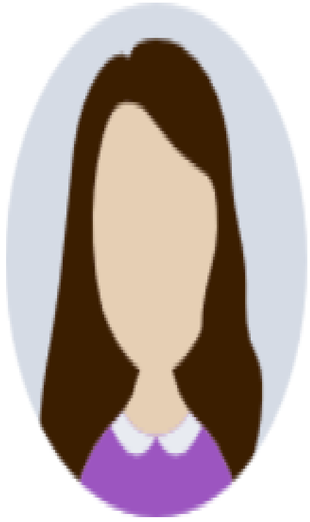
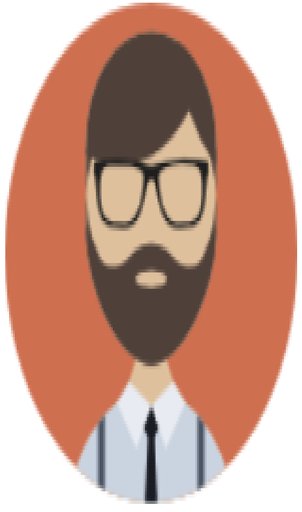
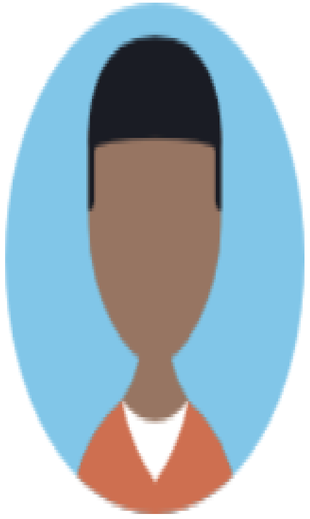
What level of proficiency do I need?

- To use programming as a tool in your professional toolkit, you don't need to be a computer scientist or have a similar level of knowledge as one
- The level of proficiency will depend on
  - **the problems** you are trying to solve on daily basis
  - **the subject matter area** you are in
  - **the level of sophistication of your solutions** that you would like to implement
- Most of the time, people who are subject matter experts who also use various programming tools and languages are known as *data analysts* or *data scientists*

What are the problems you are trying to solve? What is your area of expertise? What level of complexity would you like your programmatic solution to have?

A data scientist can

1. **Pose** the right question
2. **Wrangle** the data (gather, clean, and sample data to get a suitable data set)
3. **Manage** the data for easy access by the organization
4. **Explore** the data to generate a hypothesis
5. **Make predictions** using statistical methods such as regression and classification
6. **Communicate** the results using visualizations, presentations, and products


A data scientist needs to be able to

Use programming languages and tools to

1. **Wrangle** the data (gather, clean, and sample data to get a suitable dataset)
2. **Manage** the data for easy access by the organization
3. **Explore** the data to generate a hypothesis
4. **Make predictions** using statistical methods such as regression and classification

Stemming from the list above, the programming skills should cover knowing a programming language (or two, or three, or ...) to a degree that allows you to perform these operations!

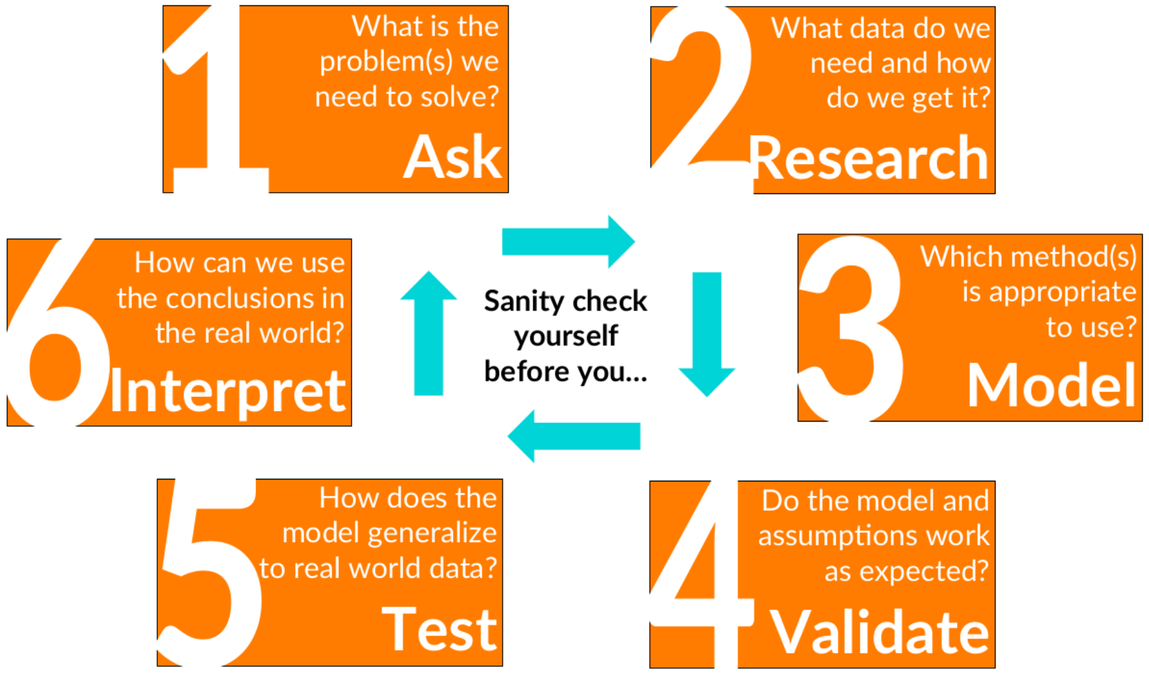
Module completion checklist

Objective	Complete
Discuss how programming is used across industries and define core functions of data scientists	✔
Explain the data science life cycle and summarize data science use cases for Python	
Refresh on coding in Python and introduce functions	

Data science control cycle: framework for data

- **There is a protocol or standard for working with data that most data scientists follow**
- The cycle involves everything from asking the right questions and being knowledgeable about the data you're studying, to optimizing your model's performance

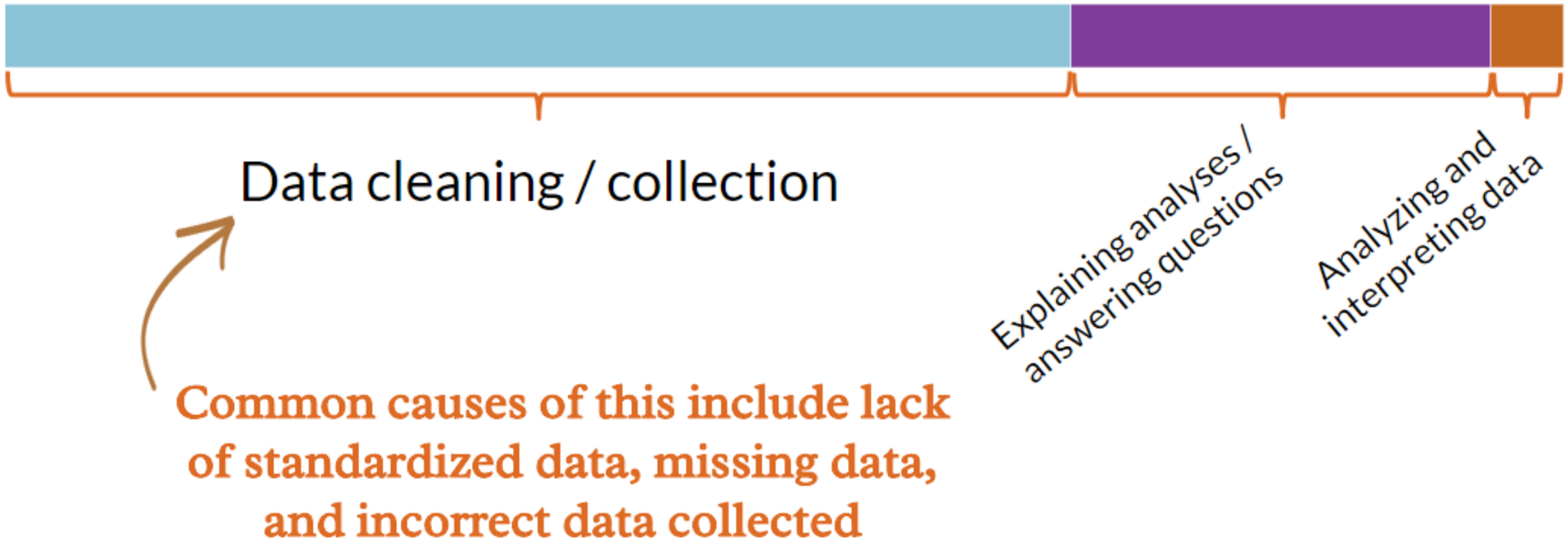
Data science control cycle (DSCC)



- Which part of the cycle do you think takes up the most time?

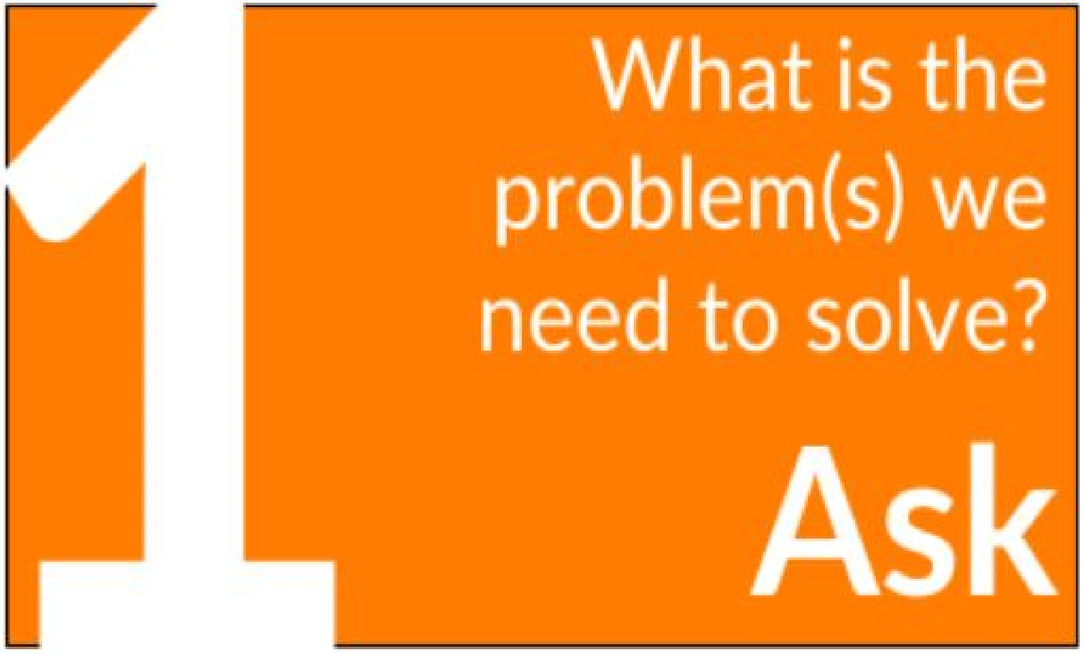
How you think data scientist spend their time

How data scientists actually spend their time



DSCC: smart questions

SPECIFIC - How are you framing the question? - What specific variables?
MEASURABLE - What metrics are you using? - What is the success criteria?
ACHIEVABLE - Scope your analysis well - Use data that is available to you
RELEVANT - Who will use this analysis? - Is it interesting or usable?
TIMEBOUND - Reference time frame of analysis - If predicting, in next year? next month? ever?

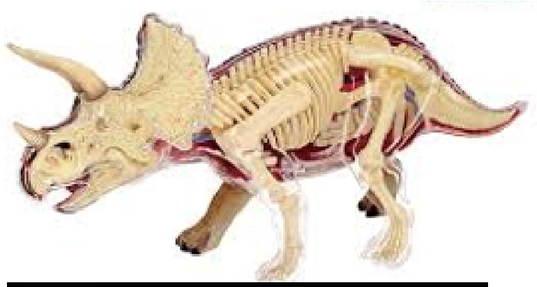


DSCC: research

- Data is key to quality results
- *Garbage in - garbage out* is the famous programming mantra that stands true for data science as well!
- It should always be on your mind when working with any dataset
- Whether it is **suitability of the data for your research** or its **quality**, it must never be overlooked

DSCC: modeling

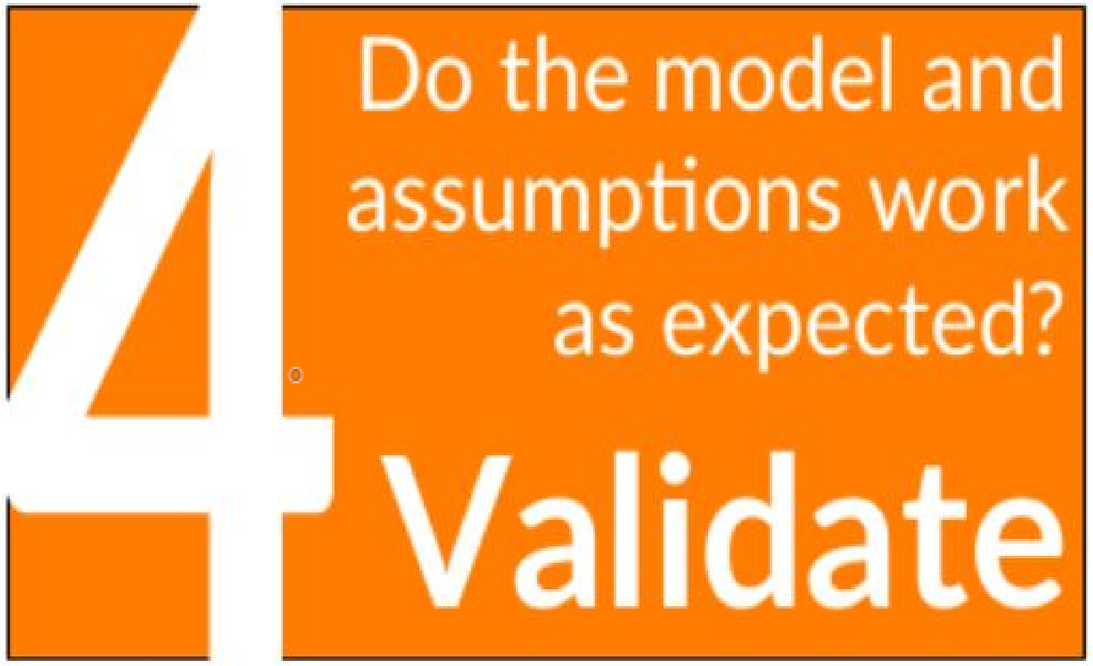
- Model by definition is a replica of a real thing
- Select a **model that suits your problem/data** or **simulates the real-life situation** in the closest possible way



$$\min_{w,b,\varepsilon} \frac{1}{2}\sum_{p=1}^{d} w_p^2 + \gamma \sum_{i=1}^{n} \varepsilon_i$$

$$\text{subject to} \begin{cases} y_i \left[\sum_{p=1}^{d} w_p x_i^p + b \right] \geq 1 - \varepsilon_i, \ \forall i = 1, \ldots, n \\ \varepsilon_i \geq 0, \qquad \forall i = 1, \ldots, n. \end{cases}$$

DSCC: steps 4 - 5



- We all have **prior knowledge** that sometimes **makes us pre-conditioned to make incorrect assumptions**
- Don’t let your extensive experience get in the way, always start as if you know nothing about the problem!
- **Validate and test your assumptions** before delivering results to stakeholders!
- Adjust the model if necessary
- **Have you ever made incorrect assumptions about a problem you were trying to solve?**

DSCC: step 6

- Interpretation of the results is as important as the results themselves
- Use your best judgment and expertise to deliver **actual information that the data carries** to stakeholders
- Make your **conclusions actionable**, so that stakeholders know what the next steps should come from your results

What is Python?

- **Python** is a powerful programming language that data scientists love because of its:
  - inherent readability and simplicity compared to lower level languages
  - number of dedicated analytical libraries to work with numerical, text, and image data
  - Over 72,000 libraries and growing constantly
  - ***Click here*** for more about Python

What can you do with Python?

Natural Language Processing - Sentiment analysis - Twitter analysis using live Twitter feeds

Deep Learning - Object recognition - Facial recognition

Visualization - Interactive visualization deployable to websites - 3D visualizations

Automation, Big Data, and Predictive Modeling - Web scraping to automate the collection of data from websites - Data wrangling with fast and efficient functions - Big Data modeling through integration with Apache Spark - Machine learning on structured data - Data ingestion from various sources

Knowledge Check 1

Module completion checklist

Objective	Complete
Discuss how programming is used across industries and define core functions of data scientists	✔
Explain the data science life cycle and summarize data science use cases for Python	✔
Refresh on coding in Python and introduce functions	

Refresh on basic coding in Python

```
# Store variables.
name = 'Mike'
a, b, c = 1, 2, 3

# Print a variable out.
print(a)
```

```
1
```

```
# Swap variable values.
a, b = b, a

# Delete a variable.
del a
```

```
# Slice a string
letters = 'ABCDEFGH'
print(letters[2:5])
```

```
CDE
```

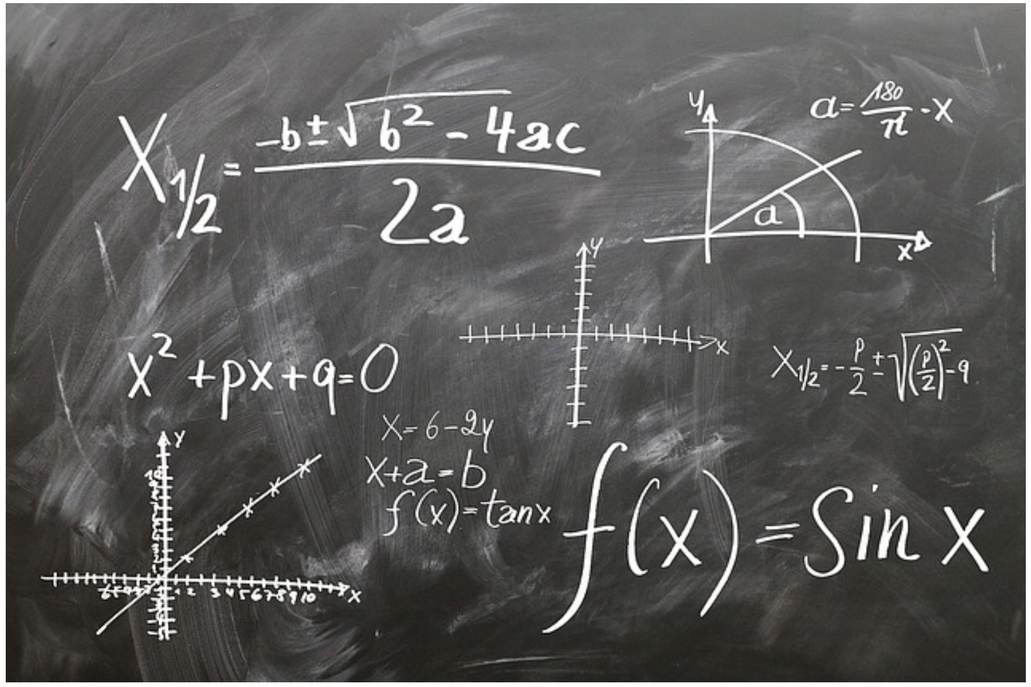
```
# Length of a string
print(len(letters))
```

```
8
```

```
print(type(letters))
```

```
<class 'str'>
```

Functions are fun!



- When you hear a word **function**, you might imagine a blackboard full of math symbols
- **Why do you need to deal with them here?**

Types of functions in Python

There are 2 basic types of functions in Python: - Built-in functions:

- These are part of the Python language that have already been built and are available for end-users
- Example: `print()`, `len()`, `abs()`
- User-defined functions:
  - These are functions that a user can create according to a specific use case or functionality
- We will now see how we can write our own functions in Python

How functions apply to programming

- In programming, functions serve a very similar purpose as in mathematics
- They help us *abstract from operations on actual numbers and let us define those operations through a set of variables*
- The **logic of operations stays stationary**, but the **value of variables changes**
- This makes it **easy to re-use such blocks** and only pass new values to those variables

User-defined functions in Python

- Functions can
  - Have `0` or more arguments
  - Perform `1` or more actions
  - Return some value(s) or not return anything
- The simplest possible function will have no arguments and will not return any value

- We can **define** a function, using a `def` construct that includes:
  - a `name` for function
  - an `action` or `actions` to perform when the function is triggered

```
# Define a function that prints the value of `Pi`.
def PrintPi():      #<- function name
    print(3.14)     #<- action to perform
```

- To "trigger" the function, we can **call** it by its name

```
PrintPi()
```

```
3.14
```

Functions in Python with 1 or more arguments

- Most of the time, you will create or use a function that has at least `1` **argument**
- When we need to define a function with argument(s), the `def` construct includes:
  - a `name` for function
  - at least `1` **argument** in parentheses
  - an `action` or `actions` to perform when the function is triggered

```
# Define a function that prints the value of `Pi`
# and takes a number of decimal points to which we want to round the number.
def PrintPi(num_decimals):                  #<- function name + argument(s)
    pi = 3.14159265359                      #<- action to perform
    rounded_pi = round(pi, num_decimals)    #<- action to perform
    print(rounded_pi)                       #<- action to perform
```

```
# Print value of pi rounded to 4 decimal points.
PrintPi(4)
```

```
3.1416
```

Functions in Python that return a value

- Functions can also **return value(s)** in addition to some `actions` they perform
- When we need to define a function with or without argument(s) that **returns value(s)**, we use a `def` construct that includes
  - A `name` for function
  - At least `0` or more `arguments` in parentheses
  - An `action` or `actions` to perform when the function is triggered
  - A `return` **value** or **values**

```
# Define a function that prints the value of `Pi`,
# takes a number of decimal points to which we want to round the number,
# and returns the value back to us.
def GetPi(num_decimals):                    #<- function name + argument(s)
    pi = 3.14159265359                      #<- action to perform
    rounded_pi = round(pi, num_decimals)    #<- action to perform
    return rounded_pi                       #<- value to return
```

Functions in Python that return a value (cont'd)

- Let's print the value that `GetPi()` returns

```
# Return a value of pi rounded to 4 decimal points.
print(GetPi(4))
```

```
3.1416
```

- A function that returns a value **can also be assigned to a variable**
- Now the value returned and assigned to a variable can be re-used throughout the code!

```
# Return a value of pi rounded to 4 decimal points.
# Assign it to a variable.
pi_4 = GetPi(4)
print(pi_4)
```

```
3.1416
```

Functions in Python: general structure

- Here is a general outline of a function that takes arguments and returns something in Python
- The arguments, actions, and returned values are all up to you to define based on what you are trying to achieve!

```
def FunctionName(argument1, argument2, ...):
    action1
    action2
    ...
    return something
```

Functions in Python: MakeFullName

- Let's define a function that concatenates the *first name* and *last name* into a **full name**

```
# Define a function that concatenates
# first and last names.
def MakeFullName(first_name, last_name):
    full_name = first_name + ' ' + last_name
    return full_name
```

- Here are the components of the function definition broken down:
  - `def` defines a function
  - `MakeFullName` is the name of our function
  - The two arguments it takes are specified in the parentheses: `first_name` and `last_name`
  - The `return` statement controls what gets returned when someone uses the function (i.e. `full_name`)

Functions in Python: calling a function

- After we have defined a function, we need to test it
- Running a function in programming is usually known as *calling* a function
- Let's call `MakeFullName()` - to do that, we need to call the function by its name and pass any necessary values we want for the arguments of the function

- Let's print out the function results

```
# Call the function.
print(MakeFullName("Harry", "Potter"))
```

```
Harry Potter
```

- If the function returns a value (like a string with full name in this case), we can assign the output of that function to a variable

```
# Call the function and save
# output to variable.
output_name = MakeFullName("Harry", "Potter")
print(output_name)
```

```
Harry Potter
```

Functions in Python: EvaluateCarPrice

- Let's define another function that evaluates the price of a car and gives us the action to do based on that
- Here are the components of the function definition broken down:
  - `EvaluateCarPrice` is the name of our function
  - The argument it takes is specified in the parentheses: `price`
  - The `return` statement controls what gets returned when someone uses the function (i.e. `action`)

```
def EvaluateCarPrice(price):
    if price > 38000:
        action = "Leave the dealership immediately, this is a rip off!"
    elif price > 22000 and price <= 38000 :
        action = "Take the car and go celebrate, you can afford it!"
    else:
        action = "Leave the dealership immediately, this is a scam!"
    return action
```

Functions in Python: calling a function

```
# Let's set the price of a car to $45,000.
price = 45000
action1 = EvaluateCarPrice(price)
print(action1)
```

```
Leave the dealership immediately, this is a rip off!
```

```
# Let's set the price of a car to $32,000.
price = 32000
action2 = EvaluateCarPrice(price)
print(action2)
```

```
Take the car and go celebrate, you can afford it!
```

```
# Let's set the price of a car to $5,000.
price = 5000
action3 = EvaluateCarPrice(price)
print(action3)
```

```
Leave the dealership immediately, this is a scam!
```

Functions that return two or more values

- Let's define another function that evaluates the sales data (i.e. a dictionary with `month: amount` key-value pairs)
  - `MostProfitableMonth` is the name of our function
  - The argument it takes is a dictionary `sales_data`
  - It **returns a tuple of the two values**: `biggest_month` and `biggest_amt` for the month with the most profit

```
def MostProfitableMonth(sales_data):

    biggest_amt = 0
    biggest_month = None

    for month, amt in sales_data.items():
        if amt >= biggest_amt:
            biggest_amt = amt
            biggest_month = month

    return (biggest_month, biggest_amt)
```

Functions that return two or more values (cont'd)

```
# Let's define a dictionary with sales data.
year_sales = {'January': 1045, 'February': 1008, 'March': 1025,
                    'April': 1080, 'May': 1100, 'June': 1050, 'July': 1050,
                    'August': 950, 'September': 1010, 'October': 1500,
                    'November': 1450, 'December': 1380}
```

- To save the output of a function that returns 2 or more values, assign the function call output to the same number of variables as there are outputs in the function in the same order

```
# Assign output of function to variables in correct order.
best_sales_month, best_sales_amt = MostProfitableMonth(year_sales)
print("Best month:", best_sales_month)
```

```
Best month: October
```

```
print("Best amount:", best_sales_amt)
```

```
Best amount: 1500
```

Passing a list as an argument

- Can we write a function to operate on a list?
  - Most certainly! Let's see how!

```
# Define a function that returns the length of all character strings in a list.
def calculate_lengths(list_of_strings):
  # Define an empty list to store the result.
  lengths = []
  # Use a for loop to access items in the input list.
  for item in list_of_strings:
    lengths.append(len(item))

  return lengths
```

```
result = calculate_lengths(["Monday", "Tuesday", "Saturday"])
print(result)
```

```
[6, 7, 8]
```

Functions that have default arguments

- Let's modify another `MostProfitableMonth` and add another argument
  - This argument will have a **default** value
  - We are not required to provide the other argument, if it has a default value
- Let's set the second argument to be `verbose`, which in programming usually signals to print some message along with returned value
- Let's set the **default value** to `True`

```
def MostProfitableMonth(sales_data, verbose = True):

    biggest_amt = 0
    biggest_month = None

    for month, amt in sales_data.items():
        if amt >= biggest_amt:
            biggest_amt = amt
            biggest_month = month

    if verbose:
        print('The best sales month was', biggest_month, 'with amount sold equal to', biggest_amt)

    return (biggest_month, biggest_amt)
```

Functions that have default arguments (cont'd)

- Let's call the function `MostProfitableMonth()` and provide only the first (required) argument

```
# Assign output of function to variables in correct order.
best_sales_month, best_sales_amt = MostProfitableMonth(year_sales)
```

```
The best sales month was October with amount sold equal to 1500
```

- Let's call the function `MostProfitableMonth()` and set the `verbose` argument to `False`

```
# Assign output of function to variables in correct order.
best_sales_month, best_sales_amt = MostProfitableMonth(year_sales, False)
```

- The output is still the same, but the "verbose" message is no longer printed, so if we need to see it, we would have to do it manually

```
print("Best month:", best_sales_month, "Best amount:", best_sales_amt)
```

```
Best month: October Best amount: 1500
```

Anonymous functions: lambda

- Anonymous functions are **small nameless functions**
- They work the same way as conventional functions, but are **used as shorthand within other pieces of code**
- They are often used as a part or as an argument in other functions

Anonymous functions: Syntax

left: 60%

- To write an anonymous function in Python, we use the `lambda` command followed by an **argument** and a `:`
- The expression that follows represents an **action** we would like to perform using those arguments

```
print((lambda v: v + " the 5th of November!")("Remember"))
```

```
Remember the 5th of November!
```

- As we can see above, the lambda function **does not require** to be tied to any name and can be directly called on the argument “Remember”

Anonymous functions with multiple arguments

- We can also assign a lambda function to a name for reusability

```
remember = lambda v: v + " the 5th of November!"
print(remember("Remember"))
```

```
Remember the 5th of November!
```

- Anonymous functions can take several arguments separated by a comma

```
y = lambda a, b: a + b
print(y(765, -987))
```

```
-222
```

Function within another function

- In order to make different parts of our code interact with each other, we can also call a function from within another function

```
# Define a function to check if a number is even
or odd.
def even_odd(num):
  if num%2 == 0:              #<- % is the Modulo
operator.
    result = "Even"
  else:
    result = "Odd"
  return result
```

```
# Define a function to print if each element of a
list is even or odd.
def even_odd_list(numbers):
  for num in numbers:
    result = even_odd(num)    #<- Call function on
num an assign its result
    print(num, ": ", result)
```

```
even_odd_list([22,41,16,13])
```

```
22 :  Even
41 :  Odd
16 :  Even
13 :  Odd
```

Knowledge check 2

Exercise 1

Module completion checklist

Objective	Complete
Discuss how programming is used across industries and define core functions of data scientists	✔
Explain the data science life cycle and summarize data science use cases for Python	✔
Refresh on coding in Python and introduce functions	✔

Next Step

In the next session, you will learn how to: - use the `numpy` package - reshape a numpy array - work with data with Pandas, another Python package - operate Pandas series

This completes our module

Congratulations!