Computer Science

Summer Springboard

SUMMER SPRINGBOARD
Look Inward. Go Upward

Day 5: Libraries

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

But first... Project 1!

- It is now time for everyone to present their project 1.
- Each person should take a turn presenting their project.
  - Show a demo of your code and your presentation.
  - After presenting, open the floor to questions / comments.
- Good luck and have fun!

Try / Except Blocks

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Wrapping Up From Last Time

- Let’s wrap up our discussion of debugging and errors with Try / Except blocks.
- Note that even though we are “wrapping up” this section, debugging is always a part of coding and it is something we will continue to practice.

Mastering Error Handling with Try / Except

- There are times when certain pieces of code may be more error prone than others.
- Sometimes, instead of crashing when errors occur, we want the program to do something else.
- This is where **try** and **except** come in.
    - These are tools for gracefully handling errors, preventing crashes and managing unforeseen issues.

The ‘try’ Block: Testing the Waters

- Wrap code that might cause an error in a **try** block.
  - Simply indent the code that you want to have inside of the **try** block, similar to putting code in a loop or function.
- If an error occurs, the **try** block is exited, and Python looks for an **except** block to handle the error.

The ‘except’ Block: Catching the Errors

- Use specific except blocks for different error types, like **except ValueError:** or **except TypeError:**.
  - This specificity helps in targeting particular kinds of errors.
- You can have multiple **except** blocks to handle different exceptions separately, ensuring a tailored response to various error conditions.

Finishing Up

- An optional **else** block can be used to execute code when the **try** block raises no errors.
- The **finally** block is executed no matter what – whether an error occurs or not. It's ideal for clean-up actions, like closing files.
    - Also optional.

Best Practices in Error Handling

- Avoid Generic Catches: Don’t use a bare **except**: as it can catch unexpected errors and hide programming mistakes.
- Resource Management: Utilize **finally** or context managers (**with** statement) for reliable resource management, like file handling.
    - More info on **with** here: https://docs.python.org/3/reference/compound_stmts.html#with

‘try’ and ‘except’ Blocks are Like Baseball

- Imagine a batter ready to hit the ball - this is like a **try** block in Python. The batter (your code) takes a swing (executes), not knowing if it'll be a hit or a miss (error).
- If the batter hits the ball heads towards a fielder, think of the fielder as an **except** block. Just like the fielder is prepared to catch specific types of hits, the **except** block is ready to catch specific errors. For example, if it's a high fly ball (a specific error like **ZeroDivisionError**), the outfielder (a specific **except** clause) is ready to catch it.
- If the batter successfully hits the ball and makes it to base without the ball being caught (i.e., no errors occur), it's like the **else** block in Python. This block runs when the code in the **try** block executes without any errors, just like the batter safely reaching base signifies a successful hit.

‘try’ and ‘except’ Blocks are Like Baseball (Cont’d)

- Regardless of whether the batter hits the ball or not, the play eventually comes to an end. This is like the **finally** block in Python, which runs no matter what - whether the batter hits the ball, misses, or even if an unusual play (exception) occurs. It's the wrap-up of the play, ensuring that everything resets for the next batter, similar to how **finally** might be used to clean up or close resources in a program.

SUMMER SPRINGBOARD
Look Inward. Go Upward

Example: Error Handling in Division

- Below is a practical example of using **try**, **except**, **else**, and **finally**:

```
70  def safe_divide(a, b):
71      try:
72          result = a / b
73      except ZeroDivisionError:
74          print("Error: Cannot divide by zero.")
75      except TypeError:
76          print("Error: All inputs must be numbers.")
77      else:
78          print("Result:", result)
79      finally:
80          print("Execution completed, whether an error occurred or not.")
81
82  # Test cases
83  safe_divide(10, 2)   # Valid division
84  safe_divide(5, 0)    # Division by zero
85  safe_divide("5", "2")  # Invalid types
```

Example: Error Handling in Division (Cont'd)

In this example:

- The **try** block contains the division operation, which might raise a **ZeroDivisionError** (if **b** is 0) or a **TypeError** (if **a** or **b** are not numbers).
- The **except ZeroDivisionError** block catches and handles the case where division by zero is attempted.
- The **except TypeError** block handles the case where the inputs are not numbers.
- The **else** block executes if there are no exceptions, and it prints the result.
- The **finally** block executes in all cases, indicating the end of the operation.
- This example is included in today's Google Colab notebook so that you can *try* (haha) it out on your own.

Enough Talk...

- Let's get into some practice problems! Navigate to today's Google Colab notebook and work until the first "***PAUSE***"

SUMMER SPRINGBOARD
Look Inward. Go Upward

Libraries

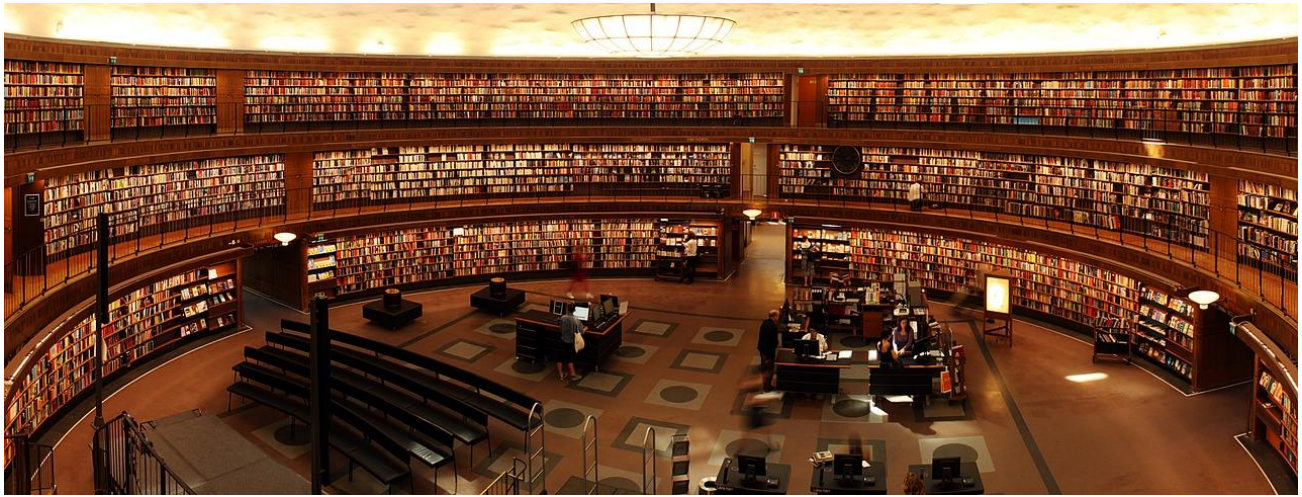
SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Introduction to Python Libraries

- Just as a carpenter uses different tools for different tasks, a programmer uses various libraries for different functionalities.
- Think of a Python library adding books to a physical library.
    - The physical library already houses a wealth of knowledge and resources, and by bringing in more books, you expand its repository of information.
    - Built-in Python has a lot of functionality already, but through libraries, we can do so much more
- Libraries save time and energy.
    - If you are having difficulty working through a specific problem, chances are someone else has had that same problem before and that they have already made a library that can help in solving your problem.

Definition

- Libraries in Python are collections of pre-written code that users can utilize to add functionality to their own programs without having to write code from scratch.



Types of Libraries in Python

- Standard Libraries: Included with Python, like **math**, **datetime**, and **os**.
- Third-Party Libraries: Developed by the community, such as **NumPy**, **Pandas**, and **Requests**.

Accessing Libraries

- To gain access to a library, simply use the **import** statement to make a library's functionality available in your code.
- For example, here we import the **Pandas** library:

```
87    import pandas
```

Importing Libraries

- As we just saw, using **import library_name** imports the library named **library_name**.
- What if we only want to import a specific section of a library?
  - To stick with our physical library example, what if we only want to add selected chapters of a book to our library?
  - Use the following syntax: **from library_name import specific_function1, specific_function2, ...**
- What if we want to name our library something else?
  - Programmers are lazy and do not always want to type out a library name, so they use a nickname / alias.
  - Syntax: **import library_name as alias**

Importing Libraries Big Example

- The below code snippet shows an example of everything we covered in the previous slide, using some Python libraries (don't worry about what these specific libraries do yet).

```
87    import pandas
88    import numpy as np
89    from matplotlib import pyplot
90    from math import sqrt, pi, cos
```

Practical Library Examples

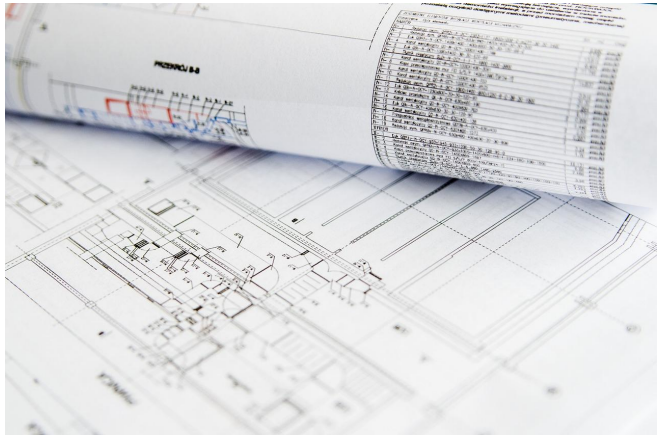
- Use **Pandas** for data manipulation and reading / processing CSV files.
- Use **NumPy** for scientific computing with complex mathematical operations on large datasets.
- Use **BeautifulSoup** or **Scrapy** for web scraping and extracting data from web pages.
- Etc etc etc the list goes on and on.

Exploring Library Functions

- How do we know what a specific library is capable of doing?
- How can we figure out what all the functions included in a library are?

Exploring Library Functions (Cont'd)

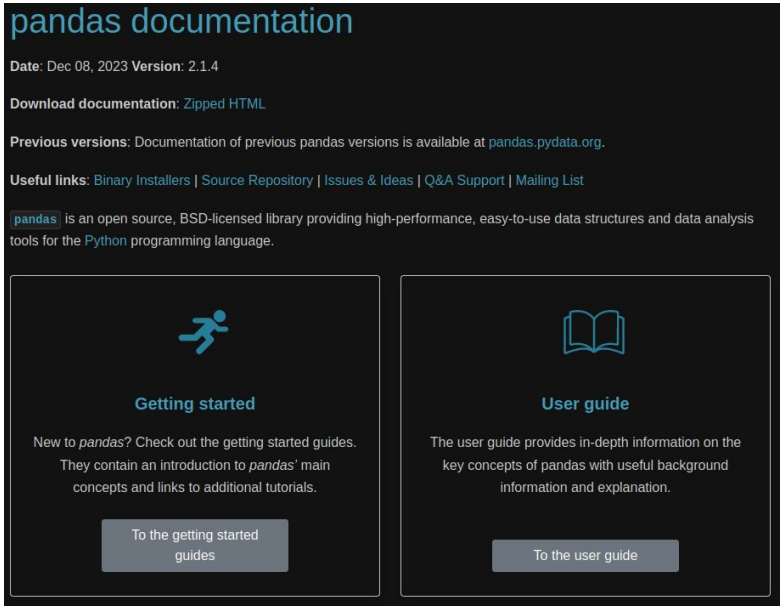
- Documentation!!!
- Documentation for a piece of code is a comprehensive descriptive guide that explains functionality, usage, and other helpful information.
- While documentation may seem dry and tedious to read through, it is our best friend when trying to figure out how to use a new tool.



SUMMER SPRINGBOARD
Look Inward. Go Upward

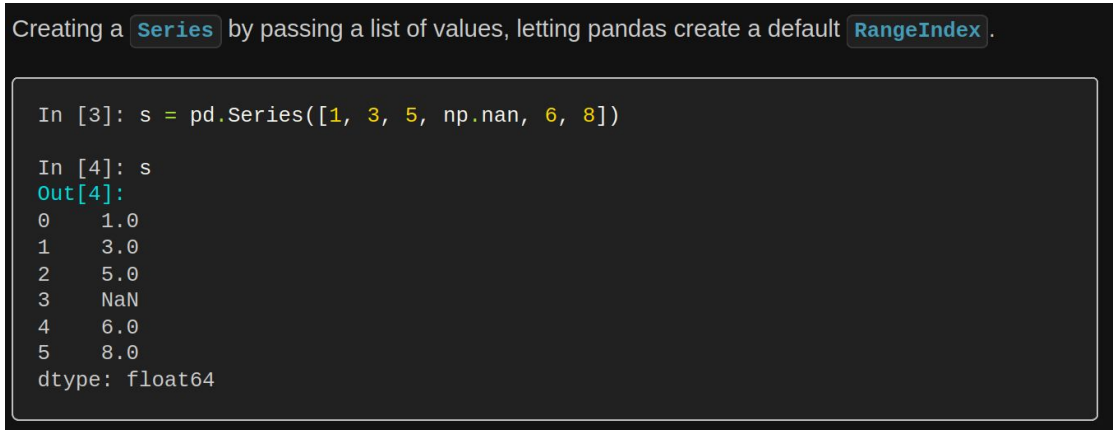
Documentation Example

- Let's take a look at an example of documentation.
- This link will bring you to the documentation for the Python library **Pandas**: https://pandas.pydata.org/pandas-docs/stable/



Documentation Example (Cont'd)

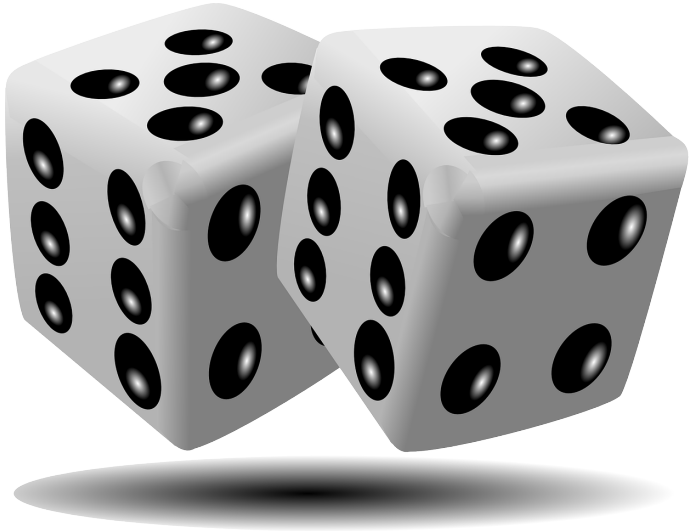
- Here is an example of reading through part of the **Pandas** documentation:



- This snippet walks you through how to create something called a **Series** in **Panda**
- The documentation will give you examples, definitions, and walkthroughs of everything related to the library you are using.

Simple Library Example

- Let's start with the **random** library.
- This library is simple and practical; it is included with Python and offers a simple way to generate random numbers, choose random elements from a list, and more.



The ‘random’ Library

- Let’s say you’re working on a program and at some point you want to pick a random element from a list.
- With our current toolset, coming up with a decently random way of selecting an item would be difficult...
- So, let’s import the **random** library!

```
92
93    import random
94
```

The ‘random’ Library (Cont’d)

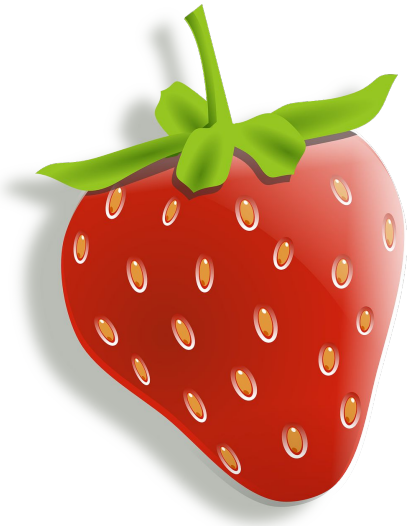
- Now, let’s make our list, **fruits**, and select a random item from it.
  - To do this, we will make use of **random**’s **choice()** function.

```
93    import random
94
95
96    fruits = ["apple", "strawberry", "cherry"]
97    print(random.choice(fruits))
```

- This piece of code will print out a random element (either **apple**, **strawberry**, or **cherry**) from **fruits**.

Other ‘random’ Examples

- Maybe we want to shuffle a list around in a random order.
  - For this, we would use the **shuffle()** function.
- For example: **random.shuffle(fruits)** takes our **fruits** list from earlier and randomly reorders the fruits that we have listed inside.



Other ‘random’ Examples (Cont’d)

- Or maybe we simply want to choose a random number in a certain range
  - For this, we would use **random**’s **randint()** function.
- In this case, we would write **x = random.randint(1,100)** to pick a random integer between 1 and 100 and store it to a variable **x**.
- Want to learn more about what **random** can do?
  - Read the documentation :)
  - https://docs.python.org/3/library/random.html

Key Takeaways

- With the **random** library, we have now seen how a few lines of code with a library can accomplish what would otherwise be complex to program
  - This is great because, as stated before, programmers are very lazy!
- Libraries are powerful when it comes to simplifying coding tasks and increasing functionality.

I wonder what could be coming up next...

- That's right! Practice problem time!
- Work through the rest of the practice problems in our Google Colab notebook.

SUMMER SPRINGBOARD
Look Inward. Go Upward

Project 2 Introduction

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Project 2

- Use at least two custom functions, include try / except blocks to handle potential errors, and use at least one basic standard library (e.g. **random**, **math**, **datetime**, etc.).
- Minimum of 30 lines of code.
- Incorporate previous concepts, like collections and conditionals.
- Prepare a Google Presentation showcasing your project and a Google Colab notebook to demo your code.
    - Include the same sections as you did for Project 1.
- As with Project 1, you will present your Project 2.
    - Present on Monday

SUMMER SPRINGBOARD
Look Inward. Go Upward

Brainstorming Time!

- Take some time to brainstorm with a partner or group what you could do for this project.
- One example of something that you could implement right now given your current knowledge is a game or quiz that uses the **random** library to generate questions or scenarios so that when different people play the game / quiz, they get different questions.
    - What collections would this program use?
    - What would the control flow look like?
    - How could you make useful functions for this program?

Final Project Introduction

SUMMER
SPRINGBOARD
Look Inward. Go Upward.

Final Project

- Since the weekend is coming up, you will hopefully have time to work on your Project 2 and get a head start on the final project.
- For the final project, you will need to develop a comprehensive Python application that demonstrates your knowledge of the course material.
- Here are the list of requirements:
  - Minimum of 50 lines of code
  - Ensure your project is reflective of the cumulative knowledge gained in the course
    - Make sure to use conditionals, loops, variables, collections, error handling, and functions
    - Next week, we will start to cover objects; try to fit this in, too.

Final Project (Cont'd)

- Requirements (cont'd)
  - Use at least one non-standard Python library that enhances or is essential to your project
    - Research some libraries that may be interesting / fun to use
      - Read the documentation.
      - Watch YouTube tutorials to get a better sense of the library.
      - Use AI tools to help you get started or to help solve a problem you are having (do not have the AI model do everything!).
      - Some suggestions: **Pygame**, **Plotly**, **FastAPI**, **Moviepy**.
- You can work individually OR in a group of 2.
  - If working in a group, make sure to put in an equal amount of effort.
  - Also, if working in a group, make sure to share your Google notebook together.

Final Project (Cont'd)

- Finally, the presentation...
- Prepare a Google Presentation showcasing your project and a Google Colab notebook to demo your code.
    - Include a title slide with your name and project title.
    - Have a brief description of your project and its functionality.
    - Tell us about your experience working on the project; what challenges did you face? What did you learn? How could you build upon your project in the future?

More Brainstorming Time!

- Take some time to brainstorm for your final project.
- What will you build?
- What library might you want to use?
- Some project ideas:
  - A data dashboard that displays trends and statistical insights from a dataset of your choosing.
  - A small website (running locally) using **FastAPI**.
  - A 2D platformer game using **Pygame**.
- It is highly encouraged to make your project as visual as possible.