Learning Outcomes: Throughout this HW, you will work to:

- Understand the distinction between an ADT and its interface, versus the data structure used in the underlying implementation.
- Understand one good use of a stack.
- Learn how HTML — and more generally, XML — is structured.
- Learn about use of command-line arguments in Python.

Some Setup: Python Command-Line Arguments

- When you run your program at the command line, you can specify so-called command-line arguments to your program. To do so, you will need to `import sys` and then leverage the included `sys.argv` list.
- For example, given the simple program below:

  ```
  import sys

  def main() -> None:
      print(f"sys.argv = {sys.argv}")
      print(f"name of program = {sys.argv[0]}")
      for i in range(1, len(sys.argv)):
          print(f"arg {i} = {sys.argv[i]} \t type = {type(sys.argv[i])}")

  if __name__ == "__main__":
      main()
  ```

  when you execute the program, you will see how command-line arguments work in Python.
- Try the above simple program. Run the program with a varying number of arguments to your program, as in:

  ```
  % python cla.py
  sys.argv = ['cla.py']
  name of program = cla.py

  % python cla.py 2 blue
  sys.argv = ['cla.py', '2', 'blue']
  name of program = cla.py
  arg 1 = 2       type = <class 'str'>
  arg 2 = blue    type = <class 'str'>
  ```

Assignment:

1. Create a new file named `dcs229_hw_stacks.py`.
2. Write a new function named `readFile` that accepts a single string argument corresponding to a filename. Include code that will open and read a file whose name is provided as the argument. For opening and reading the file, use Python's "`with open() as`" pattern (see https://realpython.com/lessons/with-open-pattern/) and `read()` to read the HTML file. Return the contents of the files as a single string. In your docstring, include a "Raises" block indicating that your function may raise a `FileNotFoundException`.

3. In `main`, pass `sys.argv[1]` as the first argument to `readFile` so that the user can provide the filename as a command-line argument. In the example execution below:

   ```
   % python dcs229_laby6.py little_boat.html
   ```

   the user will be asking the program to read and process the file named `little_boat.html`.

   Include the above logic within a try-except block, looking for two different possible exceptions:

   (a) An `IndexError` exception that will allow you to handle the case when the user has not provided an appropriate number of command-line arguments. Print a useful error message (detailing how to use your program), and call `sys.exit(2)` — see `https://docs.python.org/3/library/sys.html#sys.exit`.

   (b) A `FileNotFoundError` exception — which `readFile` may raise – that will allow you to handle the case when the user provides the name of a file that does not exist. Print a useful error message, and call `sys.exit(1)` — again, see `https://docs.python.org/3/library/sys.html#sys.exit`.

4. Write a new function named `parseHTML` having a single `str` parameter corresponding to the entire HTML text read and returned by `readFile`. Your function must return `True` if the HTML consists entirely of properly-matched tags, or `False` if there is any improperly-matched tag. If the the HTML is not properly matched, you must also print a description of what went wrong. Below are two different examples of ways a file could fail, and appropriate printed messages:

   ```
   mismatched <html> to </i>

   unmatched tags: <head>,<html>
   ```

   You must use the (provided) `ArrayStack` class in your solution. (You are not allowed to use `BeautifulSoup` or similar libraries to parse the HTML — rather, use native-Python string methods.)

   Include appropriate type hinting and a complete docstring.

5. Test your solution carefully on a variety of HTML files. On Lyceum, I have provided the HTML file `little_boat.html` having properly matched tags. Copy and modify this file to test your implementation on different valid and invalid tag pairings.

Submitting: When done, upload your `dcs229_hw_stacks.py` solution to Lyceum.