**Name:** Brayden Fox

**Date:** 07/06/25

**Program Description:**

This program allows the user to specify a number of student data entries and then enter data for each student to be stored in a CSV file.

# Functions

1. **Function Name:** `main()`

**Description:** runs the main program

**Parameters:** none

**Variables:**

* 1. `file\_name`: stores the file path to write to
  2. `field\_names`: stores the field headers to write with the CSV file.
  3. `field\_types`: helps to enforce the expected type for each corresponding column of CSV data (stores callables which do type conversion and raise `ValueError` when input is bad.
  4. `data`: list which stores the data for the CSV file.
  5. `nof\_entries`: stores the number of expected entries specified by the user.
  6. `entry\_idx`: looping variable, stores current working entry.
  7. `field\_idx`: looping variable, stores current field index; used to index into `field\_types` to call the proper function for input validation.
  8. `field\_name`: stores the current field to be gathered from the user.
  9. `item`: stores raw text data gathered from user, later validated.
  10. `writer`: stores the return of `csv.DictWriter()` when called on the CSV file.
  11. `csvfile`: stores the `TextIO` object assigned by the context manager block when the save file is created.

**Logical Steps:**

1. Initialize all the program variables which remain constant (`file\_name`, `field\_names`, etc.)
2. Begin data entry.
   1. Query user for number of entries and engage in a loop to collect the specified number of entries.
   2. For each entry, append an empty working dictionary to the end of `data`.
   3. For each field type, collect an entry from the user and attempt to validate for type.
      1. The corresponding callable in `field\_types` is passed the user’s input in a try-expcept block
      2. If a `ValueError` or `TypeError` are raised, the expected value type is displayed and the user is prompted to re-enter.
      3. Otherwise, the data is considered “validated”
   4. Valid data is assigned to the corresponding dictionary key in the current (last) dictionary element of `data`.
3. Data entry ends, and a file is created at the file path given by `file\_path`.
4. A `with` block is used to manage the file, and the `csv` module is used to write the headers and dictionary data from `data` to the CSV file.
5. The file is automatically closed by the context manager, and the program exits.

**Returns:** None

1. **Function Name:** …

**Description:** …

**Parameters:** …

**Variables:** …

**Logical Steps:** …

**Returns:** …

1. **Function Name:** …

**Description:** …

**Parameters:** …

**Variables:** …

**Logical Steps:** …

**Returns:** …

# Logical Steps

1. The `csv` module is imported
2. The if block which calls the main function is entered.
3. The main function is called, and when completed the program terminates.

# Repository

<https://github.com/fox-2-4/COP2373>

# Screenshots

**Data entry:**

